CUE Live

User Guide

3.2.4-2

Table of Contents

1 Introduction

CUE Live is a liveblogging application based on the CUE Content Store and CUE. CUE Live extends the CUE editor to provide bloggers, journalists and editors with a purpose-built, streamlined liveblogging platform. Since CUE is a browser-based app, all you need to use it is a web-capable device – anything from a laptop down to a smartphone – and a network connection.

CUE Live is designed to support continuous news coverage of **events**. An event can be a football match, a royal wedding, a disaster, an election, a conference – something that is of limited duration and of great interest to your public. An event is represented in CUE Live by a special type of content item, also called Event. An Event content item consists of a reverse-chronological series of **entries** containing updates about the event.

An event is published as a self-updating story: whenever a new entry is added to the event and published, it is effectively pushed to all readers currently viewing the event; they do not need to actively refresh the story. Since events are published in reverse-chronological order, the new entry appears at the top of the event page.

Entries can contain pretty much anything: text, images, links, media clips, tweets and so on, and can come from a variety of sources:

- Any CUE Live user working on the event
- Agency feeds
- Social platforms such as Twitter, Instagram and YouTube

The exact structure of an entry can vary from event to event, and you can configure your own entry types. A football entry, for example, could have an "incident" field that can be set to "goal", "penalty", "offside" and so on. An entry designed for an election event might have a "constituency" field for specifying where the report is coming from.

Once CUE Live is installed and configured, it is very easy to use. Events are created and published in exactly the same way as other content items. When an event is opened in CUE, its **Live Editor** tab can be used to create and edit entries. Multiple contributors can open and work in the same event simultaneously.

> CUE Live is the new name for the CUE plug-in previously known as CUE Live Center. The primary difference between CUE Live and CUE Live Center is that CUE Live is now an integrated part of the CUE editor, rather than a stand-alone web application.

2 Using CUE Live

CUE Live is a plug-in or extension to the CUE editor, so to use it you simply log in to CUE in the usual way and open or create an **event**. An event is just a special type of content item designed to be used with CUE Live. It contains a special editor that you can use to create and edit blog entries. In general, CUE Live is very straightforward to use, like the CUE editor itself, and does not require much description. This section is intended to provide just enough information to get you started.

2.1 Creating an Event

To get started you need an event to work with. If the event already exists, then you just open it in the same way as any other content item in CUE. If it doesn't already exist and you have sufficient rights, then you can create an event in the same way as any other content item, by clicking on the **+** button in the top left corner of the CUE window. When the **Create new** dialog is display, select **Events** from the **Search for other options** drop-down, and then select the event type you want from the **Choose options from group** drop-down. In the example shown below, there's only one event type to choose, called **Event**:

You then need to fill a few fields in the event's **Main** tab as shown below. In particular note the **Entry type** field – you may have several different types of entry to choose from as is the case here:

When you have filled out the **Main** tab, click the **Save** button in the bottom left corner. When you save the new event, several new tabs are created, and the **Live Editor** tab is automatically displayed. You can start creating entries straight away (see section 2.2), but you will probably want to set up the event to support the inclusion of content from social media and other external sources first (see section 2.3), and possibly change some of the event's settings (see section 2.4).

2.2 Blogging with CUE Live

To start blogging, select **Live Editor**. For a new event, the Live Editor tab is almost empty, but you'll see a small entry editor at the bottom of the page:

The entry editor works more or less in the same way as an ordinary CUE content editor: fill in the displayed fields with the content you want, and when you are satisfied with the content, select **Save** to submit it for approval to your editor. Selecting **Save** will apparently cause the entry to just disappear, but in fact it is submitted to the **Live Inbox**, a holding area for unpublished entries. You can display the Live Inbox by selecting from the column of panel buttons on the left:

The entry in the editor shown above only contains one field, because all the other entry fields are hidden. You can display the other fields and edit them by selecting the button:

The main entry field is usually a rich text field as shown in the screenshots above, with a palette of formatting buttons above it that you can use for simple text styling and the insertion of hyperlinks. As well as formatting text you can drag in content from social media. Clicking on the button, for example, will open a Twitter feed in the panel on the left, and from there you can drag a tweet into the rich text field in the entry editor. You can add YouTube videos and RSS links to your entries in exactly the same way.

If your editor approves of an entry you have submitted to the inbox, then she can publish it. It will then be published online and also appear on the **Live Editor** page, in a list above the entry editor. Note that this list of published entries is displayed in **chronological order**, with the most recent at the bottom. This is most likely the opposite order to the published live blog, where it is normal to publish entries in reverse chronological order.

2.2.1 Adding Images

You can post images by dragging image content items from the CUE search panel into an entry's rich text field.

2.2.2 Adding Social Content

You can include content from social media such as Twitter, YouTube and so on in your entries. There are two different ways of doing this:

- Directly from feeds displayed in CUE Live
- By means of **embed codes**

2.2.2.1 Using Social Media Feeds

An event can include social feeds of various kinds. Currently Twitter and YouTube feeds are supported, in addition to generic RSS feeds. An event's feeds (if it has any) can be displayed using buttons displayed in the column of panel buttons on the left. Clicking on the button, for example, displays a panel containing a Twitter feed. If you see an item in the feed that you want to include in the event blog, then you can do so by dragging it into a rich text field in the entry editor. You can then fill out any other fields in the entry as required and **Save** it in the usual way.

The feeds available in an event will have been set up by the event designer – for details see section 2.1.

2.2.2.2 Using Embed Codes

You can also include content from social media services by inserting **embed codes** in an entry. You can insert embed codes in any rich text field (that is, any field that displays a formatting palette). To insert an embed code, select the </> button near the right hand end of the formatting palette, and then paste the URL of the social media content you want to include in your entry into the displayed dialog box. For a YouTube video, for example, you might enter something like:

```
https://www.youtube.com/watch?v=yVwAodrjZMY
```


To insert the code, click on **OK**. The video then appears in your entry.

You can include several social media items in a single entry if you wish, along with your own content.

> Note that all you need to enter in the **URL** field is the URL of the required content. You do not need to get any special embed code from the social media site - the required code is constructed by CUE Live's embed code function.

The delivered system includes full support for the following social media services:

- YouTube
- Vimeo
- Instagram
- Vine
- Twitter

However, many other sites are also supported via an open standard for embedding called Oembed. There is a list of Oembed providers here. You should be able to embed content from any of the sites in this list.

For information about how to add full embed code support for additional sites and services, see chapter 5.

2.2.3 Pinning Entries

Click on the button (on the bar at the bottom of the **Live Editor**) to **pin** an entry or make it "sticky". A sticky entry is typically displayed permanently at the top of the published event feed. Unlike an ordinary entry, it will not be pushed down the feed as new entries are added. Sticky entries are marked with a pin in CUE Live:

and displayed permanently at the bottom of the published entries list.

2.2.4 Tagging Entries

Some events have **taggable** entries, in which case a button is displayed on the bar at the bottom of the **Live Editor**. To tag the entry you are editing, click on the button and select a tag from the displayed list. You can add as many tags as you like in this way.

To remove a tag from an entry, click on its **x** button.

How tags are displayed in the published blog is determined by your publication designers.

For information about how you make entries taggable, see section 4.4.1.

2.2.4.1 Auto Tagging

CUE Live can be set up to auto-tag entries, in which case you may see that tags are added to your entries without you doing anything. You can still add tags of your own in addition to those added by the system. For more information about auto-tagging and how to set it up, see chapter 6.

2.2.5 Editing Entries

You can edit existing entries as long as they have not yet been published. To edit an entry, select to open the Live Inbox panel, right-click or long-press the entry you are interested in and select **Edit**. The entry is then moved back into the entry editor, where you can make changes to it. When you have finished, select **Update** to save your changes. Note that you can edit any entries in the Live Inbox panel, not just ones that you created yourself.

2.2.6 Posting to Twitter

The entries in some events may have a "post to Twitter" button ( ) displayed on the bar at the bottom of the **Live Editor**. If select this button while editing an entry, then the entry is marked with a Twitter icon, and when it is published, it is also simultaneously posted to Twitter, using a predefined account.

To cancel your selection, just select the button a second time.

2.3 Including External Content

It is important to be able to easily include content from external source such as social media in a live blog. You can easily set up a CUE Live event to enable streamlined inclusion of content from a wide range of social media platforms. You can also set it up to include external video feeds. There are three different ways of enabling the inclusion of external content:

- Social media feeds
- Video streams

2.3.1 Social Media Feeds

You can define social media searches related to the subject of your event, and display the search results in CUE Live. Once you have set up such searches, the results can be displayed as real-time feeds, allowing you to monitor content related to your event, and easily drag interesting posts from the feeds into your blog entries.

To define such a search:

1. Select **External Content**.
2. Select the **+** button under **Subscriptions**.
3. Select the social media service you are interested in from the **Subscriptions** pull-down.
4. Enter the string you want to search for in the **Query** field. Exactly which services are available will depend on how CUE Live has been configured (section 4.1.1). You can search for hashtags, users and free text in Twitter. For all other services only free text searches are supported.
5. Select **Save**.

You can add as many different searches as you want in this way.

To view the resulting feeds, select one of the social feed buttons on the left - the Twitter button ( ), for example.

2.3.2 Video Stream

You can also add a live video stream to an event. The video stream is not displayed in the CUE Live webapp, but can be displayed in the published live blog if your publication has been configured to support it.

To add a video stream:

1. Select **External Content**.
2. Select the + button under **Video Stream**.
3. Fill in the displayed fields:

   Mime type
   The MIME type of the video stream

   URL
   The URL of the video stream
4. If required, click the + button again and repeat. The purpose of the + button in this case is not to allow the addition of many different video streams, but to allow the addition of different versions of the same stream, for display on different devices.
5. Select **Save**.

2.4 Event Settings

An event has only two settings:

Disable event
Check this option to disable or conclude an event. The event will then have the status **Concluded** in CUE Live. Concluded events cannot be edited. A concluded event remains published however, so it is still accessible on your web site.

Clear author field on submit
By default, the **Author** field in event entries is **persistent**: if you set it when you submit an entry, then your choice is remembered and applied to any subsequent entries you add, until you explicitly change it or clear it. Check this option if you would prefer the **Author** field to be cleared every time an entry is submitted.

2.5 Editorial Controls in CUE Live

If you have editor access rights then you can carry out all the editing operations described in section 2.2, but you can also do a few other things as well:

Publish your entries immediately
If you create an entry as a user with editor access rights, then instead of submitting it for approval, you can just publish it: a **Publish** button is displayed next to the **Submit** button.

Publish entries from the Live Inbox
Right-click/long press any entry displayed in the Live Inbox panel and select **Publish** from the displayed menu to publish it.

Edit published entries

As an editor, you can change entries after they have been published. Right-click/long press any entry displayed in the published entries list and select **Edit** from the displayed menu. Make your changes in the entry editor and click on **Update** to save them.

Delete entries

To delete an entry, right-click/long press on it and select **Delete** from the displayed menu. Note that deleted entries are not physically deleted, simply marked as deleted. They are still displayed in the published entry list and / or inbox panel, but are grayed out.

3 Installation

The following preconditions must be met before you can install CUE Live 3.2.4-2:

- CUE Content Store version 7.10.0-7 and CUE assembly tool have been installed as described in the CUE Content Store Installation Guide and are in working order.
- The CUE editor has been installed and as described in the CUE User Guide and configured to work with the Content Store installation.
- You have the credentials needed to access CCI Europe's SW repositories.

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

Placeholder	Path
engine-installation	**`/usr/share/escenic/escenic-content-engine-3.2.4-2`**
assemblytool_installation	**`/usr/share/escenic/escenic-assemblytool`**

In a multiple-server environment:

- CUE Live must be installed on **all** servers.
- The CUE event mechanism must be working correctly.

After installing CUE Live you must also activate CUE's CUE Live plugin. For details, see section 3.7.

3.1 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 Store components into an enterprise archive or .EAR file.

engine-host
: The machine(s) used to host application servers and Content Store instances.

editorial-host
: **engine-host**(s) that are used solely for (internal) editorial purposes.

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.

3.2 Installing

To install CUE Live 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-live
   ```

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-live-version.x86_64.rpm
```

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

3.3 Re-assembling Applications

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

3.4 Verifying The Installation

To verify the status of CUE Live, open the CUE Admin web application (usually located at **`http://`***server*`/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.

3.5 Updating The Database Schema

CUE Live 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 four scripts in each folder:

- **`constants.sql`**
- **`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/live/misc/database/mysql`**).
3. Run the scripts as follows
   - For MySql:

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

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

3.6 Upgrading

To upgrade CUE Live:

1. Read the release notes for your planned upgrade. Make a note of any special tasks that need to be carried out in connection with the upgrades.
2. Log in as **`root`** on your **assembly-host** and on each of your **engine-hosts**, and enter the following commands:

   ```
   # apt-get update
   # apt-get upgrade
   ```
3. Carry out any required upgrade tasks.

3.7 CUE Plug-in Activation

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

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/Live.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
   ```

4 Configuration

After installing CUE Live (see chapter 3) you need to configure it to meet your requirements. This involves the following tasks:

- Configure CUE Live itself
- For each publication in which you want to publish live events:
  - Add an event content type definition to your publication **`content-type`** resource
  - Create an **`entry-type`** publication resource (an additional resource required by CUE Live) and upload it to your publication

4.1 CUE Live Configuration

This set-up task consists of copying example configuration files from the CUE Live 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. > This step is only required if you installed CUE Live using the old-style installation method. If you installed it using the new package-based method, then the required configuration files will already have been installed in **`/etc/escenic/engine/common/com`**.

   Copy all the supplied common configuration layer files as follows:

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

   > If you installed CUE Live 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 copied files as described in the following sections.

4.1.1 General Settings

Open **`/etc/escenic/engine/common/com/escenic/livecenter/Configuration.properties`** for editing and set the following properties:

`autoPopulateAuthors`
: If set to **`true`** (the default), then when a CUE Live webapp user creates an entry, his name is automatically added to the entry's **`Authors`** field. If set to **`false`**, then this does not happen. The user's name is **always** added to the entry's **`Creator`** field, irrespective of how this property is set.

`webserviceUri`
: The name of the CUE Content Store web service (**not** the CUE Live web service). The default setting is **`/webservice`**, which will work if the web service is running on the same host as CUE Live, and if it has been deployed with the default name **`webservice`**. If the service has been renamed then you will need to set this property accordingly. If the service is running on a different host then you will need to specify the service's absolute URL.

editorialWebserviceUri
The URI of the CUE Content Store editorial web service. The URI must be specified as an absolute URI even if it is deployed on the same host as CUE Live. The default setting is **http://localhost:8080/live-center-editorial**. This setting will work if:

- The editorial web service is running on the same host as CUE Live
- The editorial web service has been deployed with the default name **live-center-editorial**
- The host IP address is mapped to **localhost** (usually the case)

If any of these conditions are not met, then you need to set this property with the correct absolute URL.

twitterAPIKey
The API key CUE Live is to use for connecting to Twitter (so that events can be configured to monitor Twitter for relevant tweets). The value of this key can be found in the "Keys and Access tokens" tab in your Twitter settings. If defined, then **twitterAPISecret** must also be defined.

twitterAPISecret
The API secret CUE Live is to use for connecting to Twitter. The value of this secret can be found in the "Keys and Access tokens" tab in your Twitter settings. If defined, then **twitterAPIKey** must also be defined.

googleApiKey
The Google API key for CUE Live is to use for connecting to Google. To obtain a **googleApiKey** you need to follow the following steps.

- You need a Google Account to access the Google Developers Console, request an API key, and register your application.
- Go to the Google Developers Console.
- Select a project, or create a new one.
- In the sidebar on the left, expand APIs & auth. Next, click APIs. In the list of APIs, make sure the status is ON for the YouTube Data API v3.
- In the sidebar on the left, select Credentials.
- The API supports two types of credentials. For CUE Live we will use **API key**. After selecting **API key** choose **Server key**. Then use create to generate the key.

For details of how to create api key visit Obtaining authorization credentials

presentationWebserviceUri
The URI of the CUE Live Presentation web service. The default setting is **/live-center-presentation-webservice**, which will work if the presentation web service is running on the same host as CUE Live, and if it has been deployed with the default name **live-center-presentation-webservice**. If the service has been renamed then you will need to set this property accordingly. If the service is running on a different host then you will need to specify the service's absolute URL.

twitterProfiles
A comma-separated list of references to Twitter profiles that may be used for **posting** entries to Twitter (not for monitoring - see **twitterApiKey** and **twitterApiSecret** above). Each referenced Twitter profile must be defined in its own **.properties** file. If, for example, you specified:

```
twitterProfiles=./client/DefaultTwitterProfile,./client/
JournalistTwitterProfile,./client/EditorTwitterProfile
```

then you would also need to create three corresponding properties files in the **`client`** subfolder: **`DefaultTwitterProfile.properties`**, **`JournalistTwitterProfile.properties`** and **`EditorTwitterProfile.properties`**. For further information about this, see section 4.1.3.1.

`presentationSseProxyUri`
Set this property to enable Server-sent Events (SSE) **in the presentation layer**. SSE is always used for communication between CUE and the editorial web service, but it is only used in the presentation layer if you enable it by setting this property. You must set it to the URI of the SSE Proxy to be used by the presentation layer. For example:

```
http://my-cuelive-presentation-proxy/
```

4.1.2 Back-end User Configuration

Certain CUE Live functions need a back-end user to have been set up. The back-end user is a standard CUE user with a high level of access (at least section editor privileges for all sections and write access for all content types). CUE Live uses the back-end user to publish:

- Images submitted by CUE Live users without publishing rights
- Entries created from trusted Twitter account tweets

If you do not configure a back-end user, then users with journalist roles will not be able to upload images, so for most installations this is a required configuration.

You can either set up a common back-end user for all publications, or one back-end user for each publication.

To set up a common back-end user, open **`/etc/escenic/engine/common/com/escenic/livecenter/BackendUser.properties`** for editing and set the following properties:

`userName`
The user name of your back-end user. The user must have at least section editor rights and write access for all content types.

`password`
The back-end user's password.

To set up separate back-end users for each of your publications:

1. Make several copies of **`/etc/escenic/engine/common/com/escenic/livecenter/BackendUser.properties`**, one for each publication, and name them accordingly. For example:

   `DailyNewsBackendUser.properties`
   `WeeklyGossipBackendUser.properties`
2. Set the required **`username`** and **`password`** properties in each file.

3. Open **`/etc/escenic/engine/common/com/escenic/livecenter/PublicationWiseBackendUser.properties`** for editing and set a **`backendUsers`** property of the form

   ```
   backendUsers.publicationName=configuration
   ```

   for each publication, where configuration references one of the BackendUser configurations you created in step 1. For example:

   ```
   backendUsers.DailyNews=./DailyNewsBackendUser
   backendUsers.WeeklyGossip=./WeeklyGossipBackendUser
   ```

4.1.3 Twitter-related Configurations

CUE Live can make use of Twitter in a number of different ways, which makes the configuration options related to Twitter potentially confusing. The different ways in which Twitter can be used are:

Twitter monitoring
To set up CUE Live to support Twitter monitoring, you simply need to set the **`twitterAPIKey`** and **`twitterAPISecret`** properties in **`Configuration.properties`**, as described in section 4.1.1. CUE Live then logs in to the specified Twitter account and events can be configured to monitor Twitter for relevant tweets, and the user can drag tweets into entries as described in section 2.2.2.1.

Posting entries to Twitter
If you want users to be able to post CUE Live entries to Twitter, then you can enable this functionality by creating **Twitter profiles**. A Twitter profile is defined in a **`.properties`** file as described in section 4.1.3.1. You also need to select the Twitter profile to use for posting to Twitter when defining your events in CUE (see section 2.1) and specify which entry fields to use by adding **`com.escenic.live-center.content-type.field.twitter-tweet`** parameters to the required entry definitions (see section 4.4.4.1).

When CUE Live posts an entry to a Twitter account, it appends a pingback URL to the tweet. For details of how to control the format of the URL, see section 4.1.3.2.

4.1.3.1 Creating Twitter Profiles

A Twitter profile contains details of a Twitter account that can be used by CUE Live. You can define a number of different Twitter profiles, each connected to a different Twitter account.

Twitter profiles can be used for two different purposes:

Posting to Twitter
Events can be defined to include a Post to Twitter (section 2.2.6) option. When defining such an event in CUE, you can choose which profile is to be used for this purpose. (see section 2.1). If the CUE Live user checks this option when submitting an entry, then the entry will be posted to the Twitter account defined in the profile you selected.

To create Twitter profiles:

1. Copy or rename **`/etc/escenic/engine/common/com/escenic/livecenter/client/SampleTwitterProfile.properties`**, to the required number of files, giving the files suitable names. You might, for example, create three profiles called **`DefaultTwitterProfile.properties`**, **`JournalistTwitterProfile.properties`** and

EditorTwitterProfile.properties. All these files can reside in the same **com/escenic/livecenter/client** folder.

2. Open each file for editing and set the following properties:

   profileName
   The name of the profile. This name will appear as an option when users create new events in CUE.

   twitterAPIKey
   The API key of the account CUE Live is to use for posting to Twitter when this profile is selected. See section 4.1.3.1.1 for details of how to obtain this value.

   twitterAPISecret
   The API secret of the same Twitter account. See section 4.1.3.1.1 for details of how to obtain this value.

   twitterAccessToken
   The access token of the same Twitter account. See section 4.1.3.1.1 for details of how to obtain this value.

   twitterAccessTokenSecret
   The access token secret of the same Twitter account. See section 4.1.3.1.1 for details of how to obtain this value.

3. Each profile you added in step 2 must be referenced in the **twitterProfiles** property in **Configuration.properties**, as described in section 4.1.1. Once you have added these configurations, then you can enable posting to Twitter by adding a **com.escenic.live-center.content-type.field.twitter-profile** parameter and corresponding field to the event definition in your publication **content-type** resource as described in section 4.3.1.5 and adding a "post to Twitter" option to the relevant entry definitions as described in section 4.4.3.2.

4.1.3.1.1 Getting Twitter Account Credentials

In order to get the access credentials you need to create a Twitter profile (**twitterAPIKey**, **twitterAPISecret**, **twitterAccessToken** and **twitterAccessTokenSecret**) you must log in to Twitter as the appropriate user, create a Twitter **application** using the Twitter development console (https://dev.twitter.com) and then generate an access token. In detail:

1. Open a browser window.
2. Log in to Twitter.
3. Go to https://dev.twitter.com/apps/new.
4. Fill in all required fields in the displayed form and click the **Create your Twitter application** button.
5. A new application page is displayed. Display the **Keys and Access Tokens** tab.
6. Click the **Create my access token** button at the bottom of the page.

You should now have everything you need to create a profile displayed on this page: the **Consumer Key** and **Consumer Secret** fields hold the values for the **twitterAPIKey** and **twitterAPISecret** properties, and the **Access Token** and **Access Token Secret** fields hold the values for the **twitterAccessToken** and **twitterAccessTokenSecret** properties.

4.1.3.2 Controlling the Pingback URL Format

When CUE Live posts an entry to a Twitter account, it appends a pingback URL to the tweet. By default, this is the URL of the entry's parent event. You can, however, configure CUE Live to append an entry-specific URL instead. You can choose both the kind of entry ID used, and how to include it in the URL (as a parameter or as a fragment ID).

To add an entry ID to the pingback URL:

1. Open **`/etc/escenic/engine/common/com/escenic/livecenter/social/filter/PostToTwitterTransactionFilter.properties`** for editing and set the following properties as required:

   `entryIdentifierFormat`
   Determines the method used to include the entry ID in the pingback URL:

   ```
   entryIdentifierFormat=parameter-name={entry-id}
   ```

   where *parameter-name* is the name you want to use for the parameter (**`entry`**, for example).

   `entryIdentifierGenerator`
   Specifies the generator that is to generate unique IDs for the entries:

   ```
   entryIdentifierGenerator=./UUIDBasedEntryIdentifierGenerator
   ```

   `UUIDBasedEntryIdentifierGenerator` is the default ID generator supplied with CUE Live.

4.1.4 Instagram Configuration

In order to be able to set up Instagram feeds in CUE Live you must first register your CUE Live installation as an Instagram client application, in order to get an Instagram **client ID**. The general procedure for doing this is described in the Instagram developer documentation. The specific values you need to specify when registering a CUE Live installation are:

Application Name
Any name you like that complies with Instagram's requirements.

Description
Your choice.

Company Name
The name of your company/organization.

Website URL
You can enter any valid URL here. It doesn't actually need to be the URL of anything related to your CUE Live installation.

Valid redirect URIs
This must be a URI of the form:

editorial-webapp-url`/thirdparty/authenticate/index.html`

where *editorial-webapp-url* is the URL specified as the **`editorialWebAppUrl`** property in **`Configuration.properties`** (see section 4.1.1). If, for example, **`editorialWebAppUrl`** has the default value **`http://`***ip-address*`:8080/`, then you must specify:

`http://`*ip-address*`:8080/live-center/thirdparty/authenticate/index.html`

here, where *ip-address* is the IP address of your CUE Live host (that is, the IP address referenced by **`localhost`**).

Privacy Policy URL
You can leave this field blank.

Contact email
A suitable contact email address (your own, for example).

Disable implicit OAuth (on the Security tab)
Make sure this option is not checked.

Enforce signed requests (on the Security tab)
Make sure this option is not checked.

Once you have completed the Instagram registration and obtained a client ID for your CUE Live installation, ppen **`/etc/escenic/engine/common/com/escenic/livecenter/social/auth/InstagramAuthConfigProvider.properties`** for editing and set **`params.client_id`** to the client ID you have obtained from Instagram.

4.1.5 Pagination Configuration

You can control how many entries are displayed at a time by setting pagination properties. There are two such properties: one for controlling page length in the CUE Live editor, and one for controlling page length in publications.

4.1.5.1 CUE Live Editor Pagination

To control how many entries are displayed at a time in the CUE Live editor:

1. Copy **`EntryResourceHelper.properties`** from the CUE Live installation into your **webapp** configuration layer (not the common configuration layer):

   ```
   $ mkdir -p /etc/escenic/engine/webapp/live-center-editorial/com/escenic/
   livecenter/webapp/helpers
   $ cp engine-installation/plugins/live/misc/siteconfig/webapp/com/escenic/
   livecenter/webapp/helpers/EntryResourceHelper.properties \
   > /etc/escenic/engine/webapp/live-center-editorial/com/escenic/livecenter/webapp/
   helpers/EntryResourceHelper.properties
   ```

2. Open the copied file for editing and set the following property as required:

   `entryListSize`
   Determines how many entries are returned at a time by the editorial web service, and therefore how many entries are displayed on a page.

   ```
   entryListSize = 20
   ```

4.1.5.2 Presentation Layer Pagination

To control how many entries are displayed at a time in a publication:

1. Copy **`PublishedEntryResourceHelper.properties`** from the CUE Live installation into your **webapp** configuration layer:

   ```
   $ cp engine-installation/plugins/live/misc/siteconfig/webapp/com/escenic/
   livecenter/presentation/webservice/helpers/PublishedEntryResourceHelper.properties
    \
   ```

```
> /etc/escenic/engine/webapp/pub-name/com/escenic/livecenter/presentation/
webservice/helpers/PublishedEntryResourceHelper.properties
```

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

2. Open the copied file for editing and set the following property as required:

   `entrySize`
   Determines how many entries are returned at a time by the presentation web service, and therefore how many entries are displayed on a page.

   ```
   entrySize = 20
   ```

4.1.6 CORS Filter Configuration

For security reasons, browsers commonly apply a same-origin restriction to network requests that prevent a web application running in one domain from retrieving data from other domains. CORS (Cross-Origin Resource Sharing) is a mechanism for circumventing this restriction in a secure way. If your CUE Live presentation web service is deployed in a different domain from the CUE Live web application then it will not be able to access any data unless you explicitly give it permission by configuring the CUE Live CORS filter.

The CORS mechanism is based on the concept of a **pre-flight request**. Before making a cross-domain request, a browser first sends a pre-flight request to find out what kinds of requests the host will respond to. The host then returns a pre-flight response in which it specifies which domains it will accept requests from, and what kinds of requests (which HTTP methods, headers and so on) it will respond to. The CUE Live CORS filter allows you to specify what is returned in CUE Live pre-flight responses.

To configure the CORS filter:

1. Copy **`CorsFilter.properties`** from the CUE Live installation into your **`webapp`** configuration layer (not the common configuration layer):

   ```
   $ cp engine-installation/plugins/live/misc/siteconfig/com/escenic/livecenter/
   presentation/filter/cors/CorsFilter.properties \
   > /etc/escenic/engine/webapp/live-center-presentation-webservice/com/escenic/
   livecenter/presentation/filter/cors/CorsFilter.properties
   ```

2. Open the copied **`CorsFilter.properties`** for editing and set the following properties:

   `allowedOrigins`
   A comma-separated list of **origins** that are to be granted access to CUE Live resources. An **origin** is a URL protocol identifier plus a domain name (for example **`http://www.w3.org`** or **`https://www.apache.org`**. The default setting of **`*`** grants access to all

domains. In order to restrict access to your presentation web service only, specify the web service's origin. For example:

```
allowedOrigins=https://mypresentationdomain.com
```

If you wish more than one presentation web service to be able to access CUE Live resources, then you can specify several origins:

```
allowedOrigins=https://mypresentationdomain.com,http://myotherdomain.org
```

allowedHttpMethods
A comma-separated list of HTTP methods such as **GET** and **POST** that can be used to access resources. The specified methods are returned in the pre-flight response's **Access-Control-Allow-Methods** header.

The default setting is:

```
allowedHttpMethods=GET,POST,HEAD,OPTIONS
```

allowedHttpHeaders
A comma-separated list of HTTP request headers such as **Origin** and **Accept** that can be used in cross-origin requests. The specified headers are returned in the pre-flight response's **Access-Control-Allow-Headers** header.

The default setting is:

```
allowedHttpHeaders=Origin,Accept,X-Requested-With,Content-Type,Access-
Control-Request-Method,Access-Control-Request-Headers
```

exposedHeaders
A comma-separated list of HTTP response headers that may be exposed to the browser. The specified headers are returned in the pre-flight response's **Access-Control-Expose-Headers** header.

There is no default setting for this parameter.

supportsCredentials
A flag indicating whether or not user credentials are supported. It helps browser to determine whether or not a request can be made using credentials.

The default setting is **true**.

preflightMaxAge
The number of seconds for which the browser is allowed to cache the result of a CORS pre-flight request. The specified value is returned in the pre-flight response's **Access-Control-Max-Age** header. Specifying a negative value prevents the CORS filter from including an **Access-Control-Max-Age** header in the pre-flight response.

The default setting is **1800**.

decorateRequest
A flag specifying whether or not CORS-specific attributes are to be added to the **HttpServletRequest** object.

The default setting is **true**.

4.2 SSE Proxy Configuration

The CUE editor communicates with the Content Store via an SSE proxy. In order to support this communication the SSE proxy is configured with a URL (**`http://editorial.example.com/webservice/escenic/changelog/sse`**, for example) and credentials allowing it to access the Content Store web service's SSE change log. The CUE Live plug-in, however, has its own change logs for broadcasting SSE events: an editorial change log for communicating with CUE, and a presentation change log that can be used by read-only client applications.

You therefore need to reconfigure your SSE proxy after installing CUE Live and add URL and credentials for the following additional SSE change logs:

`http://`*`host-ip-address`*`/live-center-editorial/changelogSSE`
: The URL of the editorial SSE change log.

`http://`*`host-ip-address`*`/live-center-presentation-webservice/changelogSSE`
: The URL of the presentation SSE change log.

> Note that you may have set up multiple SSE proxies for different purposes, so these settings might need to be added to different proxies.

4.3 Content Type Definition

In order to be able to use CUE Live, you need to add a suitably configured event 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.

An event content type needs to contain the following special elements:

- A parameter element with the name **`com.escenic.live-center.content-type`** and the value **`true`**:

  ```
  <parameter name="com.escenic.live-center.content-type" value="true"/>
  ```

  This element identifies the content type as a CUE Live event.
- A **`ui:decorator`** element with the name **`com.escenic.livecenter.LiveEventDecorator`**

  ```
  <ui:decorator name="com.escenic.livecenter.LiveEventDecorator" />
  ```
- A **`ui:list-style`** element containing the string **`event`**.

  ```
  <ui:list-style>event</ui:list-style>
  ```
- A **`field`** element with the name **`livecenter`**. You can control the visibility of the field with the **`<ui:visibility/>`** element (see section 4.4.2).
- Six parameters identifying fields in the content type that are used by CUE Live for special purposes. These parameters and their associated fields are described in section 4.3.1.

4.3.1 Special Event Definition Fields

An event content type can contain a number of special fields for controlling CUE Live functionality. They are identified by means of a content type parameter that points to the field. These fields and parameters are described in the following sections.

4.3.1.1 com.escenic.live-center.content-type.field.entry-type

An event content type must contain an entry type field, so that users can select the type of entry an event is to contain. The field is identified by including a **`com.escenic.live-center.content-type.field.entry-type`** parameter in the event content type and setting it to point to the entry type field.

The entry type field itself:

- Must contain a child **`entry-type`** element, belonging to the **`http://xmlns.escenic.com/2015/live-center`** namespace (which is usually given the namespace prefix **`cue-live`**).
- Must be defined as a collection field, and configured to retrieve content from CUE Live's **`entry-types`** web service.

```
<content-type name="event" xmlns:livecenter="http://xmlns.escenic.com/2015/live-
center">
  ...
  <parameter name="com.escenic.live-center.content-type.field.entry-type"
 value="entryType"/>
  <ui:list-style>event</ui:list-style>
  ...
  <panel name="main">
    ...
    <field id="entryType" mime-type="text/plain" name="entryType" select="content"
 src="/webservice/escenic/livecenter/publication/{publication}/entry-types"
 type="collection">
      <ui:label>Entry type</ui:label>
      <cue-live:entry-type />
      <ui:description>The entry type to use for this event</ui:description>
      <constraints>
        <required>true</required>
      </constraints>
    </field>
    ...
  </panel>
  ...
</content-type>
```

4.3.1.2 com.escenic.live-center.content-type.field.visibility-in-editor

If you want to be able to disable events by rendering them invisible in the CUE Live editor, then you can do so by including a **`com.escenic.live-center.content-type.field.visibility-in-editor`** parameter in the event content type and setting it to point to a boolean field as follows:

```
<content-type name="event">
  ...
  <parameter name="com.escenic.live-center.content-type.field.visibility-in-editor"
 value="disabledEvent"/>
  <ui:list-style>event</ui:list-style>
  ...
  <panel name="main">
    ...
    <field name="disabledEvent" type="boolean">
    </field>
    ...
  </panel>
```

```
  ...
</content-type>
```

An event can then be disabled by checking its **`disabledEvent`** option (see section 2.1).

4.3.1.3 com.escenic.live-center.content-type.field.clear-author-field-on-submit

If you want to be able to control the persistence of author settings in events, then you can do so by including a **`com.escenic.live-center.content-type.field.clear-author-field-on-submit`** parameter in the event content type and setting it to point to a boolean field as follows:

```
<content-type name="event">
  ...
  <parameter name="com.escenic.live-center.content-type.field.clear-author-field-on-
submit" value="clearAuthor"/>
  <ui:list-style>event</ui:list-style>
  ...
  <panel name="main">
    ...
    <field name="clearAuthor" type="boolean">
    </field>
    ...
  </panel>
  ...
</content-type>
```

It will then be possible to determine whether author field selections made in the CUE Live editor are persistent or the author field is cleared after an entry is submtted by checking/unchecking the **`clearAuthor`** option (see section 2.1).

4.3.1.4 com.escenic.live-center.content-type.field.subscription

In order to be able to include social media feeds in your event you must include a **`com.escenic.live-center.content-type.field.subscription`** parameter in the event content type and set it to point to a complex field as follows:

```
<content-type name="event">
  ...
  <parameter name="com.escenic.live-center.content-type.field.subscription"
 value="subscriptions"/>
  <ui:list-style>event</ui:list-style>
  ...
  <panel name="main">
    ...
    <field name="subscriptions" type="complex">
      <required>false</required>
      <array default="0"/>
      <complex>
        <field type="enumeration" name="source">
          <enumeration value="twitter"/>
          <enumeration value="rss"/>
          <enumeration value="youtube"/>
          <constraints>
            <required>true</required>
          </constraints>
        </field>
        <field mime-type="text/plain" type="basic" name="query">
          <constraints>
```

```
            <required>true</required>
          </constraints>
        </field>
      </complex>
    </field>
    ...
  </panel>
  ...
</content-type>
```

The complex field must be defined exactly as shown above (although you can omit enumerations for any service you do not wish to support). It is then possible to select the feeds to be displayed in an event by making selections in the **`subscriptions`** field (see (section 2.1)).

4.3.1.5 com.escenic.live-center.content-type.field.twitter-profile

In order to be able to post CUE Live entries to Twitter, include social media feeds in your event you must include a **`com.escenic.live-center.content-type.field.twitter-profile`** parameter in the event content type and set it to point to an enumeration field as follows:

```
<content-type name="event">
  ...
  <parameter name="com.escenic.live-center.content-type.field.twitter-profile"
 value="twitterprofile"/>
  <ui:list-style>event</ui:list-style>
  ...
  <panel name="main">
    ...
    <field type="enumeration" name="twitterprofile">
      <ui:label>Twitter Profiles</ui:label>
      <enumeration value="default">
        <ui:label>Default</ui:label>
      </enumeration>
      <enumeration value="escenic">
        <ui:label>Escenic</ui:label>
      </enumeration>
    </field>
    ...
  </panel>
  ...
</content-type>
```

The values in the enumeration field must be the names of Twitter profiles defined as described in section 4.1.3.1. It is then possible to select the Twitter profile that will be used for posting entries from the **`twitterprofile`** field (see section 2.1).

An entry is only posted to the selected Twitter account if the CUE Live user checks a "Post to twitter" option in the entry. Such an option must be included in the entry definition in order for posting to Twitter to be possible. For details, see section 4.4.3.2.

4.3.1.6 com.escenic.live-center.content-type.field.video

If you want to be able to add a live video stream to your event, then you can do so by including a **`com.escenic.live-center.content-type.field.video`** parameter in the event content type and setting it to point to an array of complex fields as follows:

```
<content-type name="event">
```

```
  ...
  <parameter name="com.escenic.live-center.content-type.field.video" value="video"/>
  <ui:list-style>event</ui:list-style>
  ...
  <panel name="main">
    ...
    <field name="video" type="complex">
      <array default="1"/>
      <complex>
        <field mime-type="text/plain" type="basic" name="mime-type">
          <constraints>
            <required>true</required>
          </constraints>
        </field>
        <field mime-type="text/plain" type="basic" name="url">
          <constraints>
            <required>true</required>
          </constraints>
        </field>
      </complex>
    </field>
    ...
  </panel>
  ...
</content-type>
```

You can then add a video stream to your event using the **`video`** option (see section 2.1).

4.3.2 Example Content Type Definition

The following listing shows a complete event content type definition from which unnecessary elements such as labels have been removed:

```
<content-type name="event" xmlns:livecenter="http://xmlns.escenic.com/2015/live-
center">
  <ui:title-field>title</ui:title-field>
  <ui:decorator name="com.escenic.livecenter.LiveEventDecorator" />
  <parameter name="com.escenic.live-center.content-type.field.entry-type"
 value="entryType"/>
  <parameter name="com.escenic.live-center.content-type" value="true"/>
  <parameter name="com.escenic.live-center.content-type.field.visibility-in-editor"
 value="disabledEvent"/>
  <parameter name="com.escenic.live-center.content-type.field.clear-author-field-on-
submit" value="clearAuthor"/>
  <parameter name="com.escenic.live-center.content-type.field.subscription"
 value="subscriptions"/>
  <parameter name="com.escenic.live-center.content-type.field.twitter-profile"
 value="twitterprofile"/>
  <parameter name="com.escenic.live-center.content-type.field.video" value="video"/>
  <parameter name="com.escenic.index.summary.fields" value="description"/>
  <panel name="main">
    <field name="title" type="basic" mime-type="text/plain"/>

    <field name="description" type="basic" mime-type="text/plain"/>

    <field id="entryType" mime-type="text/plain" name="entryType" select="content"
 src="/webservice/escenic/livecenter/publication/{publication}/entry-types"
 type="collection">
      <cue-live:entry-type />
```

```
      <constraints>
        <required>true</required>
      </constraints>
    </field>

    <field name="disabledEvent" type="boolean"/>

    <field name="clearAuthor" type="boolean"/>

    <field name="subscriptions" type="complex">
      <required>false</required>
      <array default="0"/>
      <complex>
        <field type="enumeration" name="source">
          <enumeration value="twitter"/>
          <enumeration value="rss"/>
          <enumeration value="youtube"/>
          <constraints>
            <required>true</required>
          </constraints>
        </field>
        <field mime-type="text/plain" type="basic" name="query">
          <constraints>
            <required>true</required>
          </constraints>
        </field>
      </complex>
    </field>

    <field name="video" type="complex">
      <array default="1"/>
      <complex>
        <field mime-type="text/plain" type="basic" name="mime-type">
          <constraints>
            <required>true</required>
          </constraints>
        </field>
        <field mime-type="text/plain" type="basic" name="url">
          <constraints>
            <required>true</required>
          </constraints>
        </field>
      </complex>
    </field>
    <field name="livecenter" type="basic" mime-type="application/json">
      <ui:hidden/>
    </field>
    <field type="enumeration" name="twitterprofile">
      <enumeration value="default"/>
      <enumeration value="escenic"/>
    </field>
  </panel>
</content-type>
```

4.4 The entry-type Resource

CUE Live requires an additional publication resource called **entry-type**. It is an XML resource that defines the field structure of different CUE Live entry types in the same way as the **content-type** resource defines the field structure of content types. It is in fact very similar to the **content-type** resource, and uses the same elements and namespaces. It is, however, generally simpler than a **content-type** resource since entries have simpler structures than content items.

For a description of the standard **content-type** resource format, see the CUE Content Store Resource Reference. When you are creating an **entry-type** resource rather than a **content-type** resource, then you need to take the following factors into account:

- Each **content-type** element defines a CUE Live entry type, not a content type.
- The **panel** element is required by **content-type** resource syntax, but has no meaning in CUE Live. You should therefore create just one **panel** in each **content-type** element as a container for all its **field** elements.
- Summary and relation-related elements such as **summary**, **relation-type**, **relation-type-group** have no meaning in CUE Live and are ignored if present.
- The following field types are not supported by CUE Live and will be ignored if present:

  collection
  complex
  schedule
- Some **field** elements may contain special CUE Live elements belonging to the **http://xmlns.escenic.com/2015/live-center** namespace. For details see section 4.4.3.
- The **entry-type** resource may also contain **parameter** elements for special purposes. For details see section 4.4.4.

4.4.1 Enabling Tags

To make an entry type taggable, all you need to do is add a **ui:tag-scheme** element to the entry definition. The content of the tag must be the scheme (that is URI) of the tag structure to be used for tagging entries. For example:

```
<ui:tag-scheme>tag:livelabels@escenic.com,2015</ui:tag-scheme>
```

For general information about tagging in CUE and tag scheme definition, see Manage Tag Structures.

4.4.2 Controlling Field Visibility

You can control the visibility of entry fields in the CUE Live editor in the same way as you control the visibility of content item fields, by adding **ui:visibility** elements (see http://docservices.dev.escenic.com/ece-resource-ref/7.10/ih_visibility.html) to field definitions. The **ui:visibility** values are used by CUE Live as described below:

hidden
: Hidden completely.

expert or **advanced**
: Not visible in the main entry editor. To view or edit these fields, the user must open a secondary editor by selecting the button on the bar at the bottom of the **Live Editor**.

beginner
Always visible.

The default value if **ui:visibility** is not set is **advanced**.

4.4.3 Special CUE Live Elements

Some **field** elements may contain special CUE Live elements belonging either to the **http://xmlns.escenic.com/2015/live-center** namespace (usually assigned the prefix **cue-live**) or to the **http://xmlns.escenic.com/2008/interface-hints** namespace (usually assigned the prefix **ui**). These special elements are described below:

4.4.3.1 The cue-live Element

Include this element in a field definition to ensure that it appears in the Live Inbox side panel and published entry list. It is typically included in **title**, **body**, **milestone**, **happening**, and **twitter-post** field definitions. For example:

```
<field mime-type="text/plain" type="basic" name="basic">
 <ui:label>Title</ui:label>
 <ui:cue-live>title</ui:cue-live>
</field>
```

4.4.3.2 The twitterPost Element

```
<field type="boolean" name="twitter-post">
  <cue-live:twitterPost responseFieldName="twitter-response"/>
  <ui:cue-live>twitter</ui:cue-live>
  <cue-live:entryShare />
</field>
```

This element is used to identify a Boolean field as a "share on Twitter" field. If the CUE Live user checks this option when editing an entry, then when the entry is submitted it will also be automatically posted to Twitter. The **responseFieldName** can be set to point to another field in the entry to which the JSON response returned from Twitter can be written. This response field must a plain text field, and should be marked with a twitterPostResponse (section 4.4.3.4) element.

Note that posting to Twitter will only work if CUE Live has been configured with Twitter profiles (see section 4.1.3.1) and if the event containing the entry has been configured to use one of those profiles (see section 4.3.1.5).

You should always include an **entryShare** element with a **twitterPost** element to enhance the usability of the sharing option. For details see section 4.4.3.3.

4.4.3.3 The entryShare Element

```
<field type="boolean" name="twitter-post">
  <cue-live:twitterPost responseFieldName="twitter-response"/>
  <cue-live:entryShare />
  <ui:cue-live>twitter</ui:cue-live>
</field>
```

This element is used to identify a Boolean field as a sharing option. It adds sharing-related functionality to social media sharing options: when the entry is published, a "spinner" icon is displayed

while the entry is being submitted to the social service, and ensures that a success/failure message is displayed. You should always include this element in the definition of any Boolean field used to define a sharing option. (Currently, CUE Live only supports the creation of sharing options for Twitter, via the twitterPost (section 4.4.3.2) element.)

4.4.3.4 The twitterPostResponse Element

```
<field mime-type="text/plain" type="basic" name="twitter-reponse">
  <cue-live:twitterPostResponse/>
</field>
```

This element is used to identify a plain text field as a "Twitter response" field. If an entry is submitted to Twitter (by means of a **twitterPost** (section 4.4.3.2) element, then the JSON response returned from Twitter can be written to a Twitter response field identified by the **twitterPost** element's **responseFieldName** attribute. A Twitter response field can be useful for diagnostic purposes but is usually configured to be hidden.

4.4.3.5 The milestone Element

```
<field type="boolean" name="milestone">
  <ui:label>Milestone</ui:label>
  <ui:description>Milestone</ui:description>
  <ui:visibility>expert</ui:visibility>
  <ui:cue-live>milestone</ui:cue-live>
  <cue-live:milestone/>
</field>
```

This element is used to identify a boolean field as a "Milestone" field. If the CUE Live user checks this option when editing an entry, then entry will be marked as milestone entry.

4.4.4 entry-type Parameters

An **entry-type** resource may contain the following **parameter** elements.

4.4.4.1 com.escenic.live-center.content-type.field.twitter-tweet

This parameter is used to identify the entry field that will be posted to Twitter when **Post to Twitter** is selected in the CUE Live editor. The **parameter** element must be inserted as the child of a **content-type** element, and its **value** attribute must contain the name of a field belonging to the **content-type** element (that is, the entry type definition). The content of the specified field will then be posted to Twitter. The chosen field must be a plain text field (that is, it must have **type="basic"** and **mime-type="text/plain"**). For example:

```
<content-type name="generic">
  ...
  <parameter name="com.escenic.live-center.content-type.field.twitter-tweet"
 value="basic"/>
  ...
  <panel name="default">
    ...
      <field mime-type="text/plain" type="basic" name="basic">
        <ui:label>Title</ui:label>
        <ui:description>A sample plain text field</ui:description>
        <ui:counter/>
        <ui:visibility>beginner</ui:visibility>
```

```
        <ui:cue-live>title</ui:cue-live>
        <constraints>
          <maxchars>200</maxchars>
        </constraints>
      </field>
    ...
  </panel>
  ...
</content-type>
```

4.4.5 Example entry-type Resource

The following example shows a short extract from an **`entry-type`** resource, containing just one entry definition.

```
<?xml version="1.0"?>
<content-types version="4"
               xmlns="http://xmlns.escenic.com/2008/content-type"
               xmlns:ui="http://xmlns.escenic.com/2008/interface-hints"
               xmlns:livecenter="http://xmlns.escenic.com/2015/live-center"
    >
  <content-type name="football">
    <ui:label>Football</ui:label>
    <ui:description>A sample entry type containing fields of all supported types</
ui:description>
    <ui:title-field>basic</ui:title-field>

    <parameter name="com.escenic.live-center.content-type.field.twitter-tweet"
 value="basic"/>

    <panel name="default">
      <ui:label>Default</ui:label>
      <ui:description>The default set of fields</ui:description>

      <field mime-type="text/plain" type="basic" name="basic">
        <ui:label>Title</ui:label>
        <ui:cue-live>title</ui:cue-live>
        <ui:description>A sample plain text field</ui:description>
      </field>

      <field mime-type="application/xhtml+xml" type="basic" name="xhtml">
        <ui:label>Body</ui:label>
        <ui:cue-live>body</ui:cue-live>
        <ui:description>A sample xhtml field</ui:description>
      </field>

      <field type="boolean" name="boolean">
        <ui:label>Milestone</ui:label>
        <ui:cue-live>milestone</ui:cue-live>
        <ui:description>A sample boolean field</ui:description>
        <cue-live:milestone/>
      </field>

      <field type="enumeration" name="singleChoiceEnumeration">
        <ui:label>Happening</ui:label>
        <ui:cue-live>happening</ui:cue-live>
        <ui:description>A sample single choice enumeration field </ui:description>
        <enumeration value="first">
          <ui:label>Goal</ui:label>
```

```
          </enumeration>
          <enumeration value="second">
            <ui:label>Corner kick</ui:label>
          </enumeration>
          <enumeration value="third">
            <ui:label>Yellow card</ui:label>
          </enumeration>
          <enumeration value="four">
            <ui:label>Red card</ui:label>
          </enumeration>
        </field>
      </panel>
    <ui:tag-scheme>tag:livelabels@escenic.com,2015</ui:tag-scheme>
  </content-type>
  ...
</content-types>
```

4.5 Cache Configuration

Caching is one of the most important factors governing CUE Live performance. You are strongly recommended to use an external cache such as Varnish in front of your Content Store installation when using CUE Live in production, and to configure caching in accordance with the guidelines in this section.

4.5.1 Entry Cache

The entry cache is an object cache used to hold recently used entry objects. It is analogous to the Content Store **`PresentationArticleCache`**, which is used to hold recently used content items, and like the **`PresentationArticleCache`** it can be configured to use the distributed memory cache **`memcached`** (see below). If you have configured **`PresentationArticleCache`** to use **`memcached`**, then you are recommended to do the same for the entry cache.

All requests to both the editorial and presentation web services are served via the entry cache.

For general information about Content Store object caches, see Tuning The Object Caches. For general information about installing and configuring **`memcached`**, see Distributed Memory Cache.

To configure CUE Live to use **`memcached`**, you need to:

1. Open **`/etc/escenic/engine/common/com/escenic/livecenter/LocalLiveEntryCache.properties`** for editing and set the following properties as required:

   `maxSize`
   The maximum number of entries that will be held in the cache. Once this limit is reached, older entries are thrown out of the cache in order to stay below the limit. The default value is 5000.

   `$class`
   If **`memcached`** is installed and in use for the **`PresentationArticleCache`** at your installation, then you are recommended to use for CUE Live entries as well. To enable use of **`memcached`**, set this property to **`neo.util.cache.Memcached`**.

2. Open **/etc/escenic/engine/common/com/escenic/livecenter/CachingLiveEntryDao.properties** for editing (or create the file if necessary) and set the following property:

```
cache=./LocalLiveEntryCache
```

4.5.2 Change Log Caches

Both of the CUE Live web services' change logs (editorial and presentation) are cached in order to reduce load on the server and database. Clients poll the change logs frequently in order to ensure that the live blogs are indeed "live" – that they update as soon as any changes are published. With large numbers of clients, this can result in many thousands of requests per second. Without caching, every change log request would also result in a database request, and the system would very quickly be overloaded.

For both web services, caching is performed at two levels:

- Internal caching, with a default lifetime of 250 milliseconds
- External caching, with a default lifetime of 2 seconds for non-empty responses.
- External caching, with a default lifetime of 0 seconds for empty responses.

The external caching depends upon the use of Varnish or some other external cache system: all CUE Live does is to add a cache header to change log responses. If the response to a change log request is **empty** – that is, nothing has changed and there are no new entries to return – then **max-age** is set to zero in the cache header by default. This means requests will be handled by the back-end server rather than the cache server. If there has been a change, and the response therefore has some content, then **max-age** is set to 2 seconds by default. This means that for the next 2 seconds (or whatever time you specify), all requests will be handled by the external cache, which will return the cached response.

For empty/no change responses, the internal cache takes over, caching for a shorter period in order to maintain acceptably fast response times. If you want to add external caching for empty responses you can achieve that by overriding the default value.

You can modify the caching parameters for both the internal and external caches, and you can modify them separately for each web service (editorial and presentation). The specific effects of changes in cache settings is dependent on many different variables, but in general, increasing cache lifetimes reduces system load at the cost of increased latency and decreasing cache lifetimes does the opposite: decreases latency at the cost of increased system load.

To modify the **editorial change log** cache settings:

1. Copy **CacheConfig.properties** from the CUE Live installation into your **CUE Live webapp** configuration layer (not the common configuration layer):

```
$ cp engine-installation/plugins/live/misc/siteconfig/webapp/com/escenic/
livecenter/changelog/cache/CacheConfig.properties
        >/etc/escenic/engine/webapp/live-center-editorial/com/escenic/livecenter/
changelog/cache/CacheConfig.properties
```

2. Open the copied file for editing and set the properties described in section 4.5.2.1 as required:

To modify the **presentation change log** cache settings:

1. Copy **`CacheConfig.properties`** from the CUE Live installation into your **CUE Live webapp** configuration layer (not the common configuration layer):

```
$ cp engine-installation/plugins/live/misc/siteconfig/webapp/com/escenic/
livecenter/changelog/cache/CacheConfig.properties
        >/etc/escenic/engine/webapp/live-center-presentation-webservice/com/
escenic/livecenter/changelog/cache/CacheConfig.properties
```

2. Open the copied file for editing and set the properties described in section 4.5.2.1 as required:

4.5.2.1 CacheConfig.properties

`CacheConfig.properties` contains the following properties:

`cacheHeaderEnabled`
: This parameter controls external caching by determining whether or not an HTTP cache header is added to non-empty change log responses. Set to **`true`** (the default) to enable caching or **`false`** to disable it. You can usually set this to **`false`** for the editorial change log, since the number or requests from editorial clients is not likely to be very high and external caching is therefore not required.

`maxAge`
: If **`cacheHeaderEnabled`** is set to **`true`**, then this property determines the HTTP cache header's **`max-age`** parameter (which sets the external cache's lifetime) for non-empty responses. It is specified in seconds. The default setting is **`2`**. Increase this value to reduce load on the server and database, decrease it to reduce latency.

`emptyResponseMaxAge`
: If **`cacheHeaderEnabled`** is set to **`true`**, then this property determines the HTTP cache header's **`max-age`** parameter (which sets the external cache's lifetime) for empty responses. It is specified in seconds. The default setting is **`0`** – in other words, caching is disabled by default for empty responses. Increase this value to reduce load on the server and database, decrease it to reduce latency.

`accessModifier`
: If **`accessModifier`** is set to **`private`**, then this property indicates that all or part of the response message is intended for a single user and that it therefore **must not** be cached by a shared cache such as a proxy server. By default, this property is set to **`public`**.

`cachingEnabled`
: This parameter controls internal caching. Set to **`true`** (the default) to enable caching, or **`false`** to disable it.

`cacheLife`
: If **`cachingEnabled`** is set to **`true`**, then this property determines the internal cache's lifetime, specified in milliseconds. The default setting is **`250`**. Increase this value to reduce load on the server and database. Increase this value to reduce load on the server and database, decrease it to reduce latency.

4.6 Third-Party Authentication

CUE Content Store can be set up to use a third party for authentication of users, instead of doing the user authentication itself. Three third party authenticators are supported – Google Apps, Microsoft Active Directory and Facebook. If the Content Store is configured to use Google Apps for user authentication, then Google Apps will also be used by CUE Live for authenticating users. This means

that users in organizations that use Google Apps as their standard office suite can also log into Content Store and CUE Live using their Google log-ins. CUE Live does **not**, however, currently support Microsoft Active Directory or Facebook-based authentication.

For a full description of the Content Store's support for Google OAuth authentication and how to enable it, see the CUE Content Store Server Administration Guide. If Google OAuth authentication is set up as described in this section, then the CUE Live login form will include an alternative **Log in with Google** link alongside the standard **Login** button.

5 Embedding External Content

CUE Live provides out-of-the-box support for embedding content from various social media services in event entries:

- YouTube
- Vimeo
- Instagram (configuration required - see section 5.1)
- Vine
- Twitter

This functionality is based on oEmbed, an open standard for displaying embedded content. In addition to embedding support for the above services, CUE Live also includes a generic oEmbed handler. This means users can in fact embed content from any oEmbed **provider** (a provider is a site that implements the content provider end of the oEmbed protocol), which includes most social media sites. There is a list of oEmbed providers here.

The generic oEmbed handler provides somewhat simpler embedding support than the site-specific handlers. It uses a generic oEmbed HTML template and provides only basic formatting. You can, however, improve support for specific sites you are interested in by adding your own handlers. You can do this in two different ways:

- Create a request handler specifically for the site or service you are interested in (see section 5.2). If you do this, then Content Store will use an HTML template provided by the site itself. This is often more sophisticated than the generic oEmbed template.
- Create a custom request handler for the site or service you are interested in (see section 5.3). You can then provide your own HTML template and ensure that the embedded content looks exactly how you want it to.

CUE Live also supports Open Graph-based embedding, making it possible to embed content from sites that support the Open Graph protocol but not oEmbed. You can create Open Graph-based request handlers in more or less the same way as you create oEmbed request handlers. See section 5.4 and section 5.5 for details).

5.1 Enabling Instagram Embeds

Although a ready-made Instagram configuration is supplied with CUE Live, it will not work out of the box, because you need to include a Facebook/Instagram access token in one of the configuration files in order to be granted access to Instagram content. To enable Instagram embeds therefore, you need to:

1. Obtain a Facebook/Instagram access token. You will find instructions on how to do that here.
2. Open **`/etc/escenic/engine/common/com/escenic/livecenter/service/proxy/InstagramRequestHandler.properties`** for editing and add the following property settings:

```
OEmbedUrlEndPoint=https://graph.facebook.com/v8.0/instagram_oembed?
access_token=your-access-token&url=
```

```
serviceEnabled=true
```

where *your-access-token* is the Facebook/Instagram access token you obtained in step 1.

5.2 Creating an oEmbed Request Handler

To create your own site-specific oEmbed request handler:

1. Copy **`LiveCenterOEmbedRequestHandler.properties`** from the CUE Live installation into your **webapp** configuration layer (not the common configuration layer), and rename appropriately. For Flickr, for example, you might enter:

   ```
   $ cp engine-installation/plugins/live/misc/siteconfig/com/escenic/livecenter/
   service/proxy/LiveCenterOEmbedRequestHandler.properties \
   > /etc/escenic/engine/common/com/escenic/livecenter/service/proxy/
   FlickrOEmbedRequestHandler.properties
   ```

2. Open the copied file for editing and set the following properties as required:

   `urlPatterns`
   A comma-separated list of host names (**not** URL patterns as listed here) for which this handler is to be used. For Flickr, for example, you might want to set this property as follows:

   ```
   urlPatterns=flickr.com,flic.kr
   ```

   `blockCodeTemplate`
   An HTML template from which the embedding code for your web pages will be generated. To include oEmbed property values in your template, enclose the property name in braces thus:

   ```
   {property-name}
   ```

   Alternatively, you can specify the path of a file in which you have saved the template, for example:

   ```
   blockCodeTemplate=file:/path/to/block/code/template
   ```

   The path must be specified as a file system URL (that is, it must have a **`file:`** prefix). The URL must be absolute.

   `requestHeaders.`*`field-name`*
   These are optional properties that you can add if you want to include any special fields in the headers of your embed requests. *field-name* must be the name of an HTTP header field. To set the **`User-Agent`** field to some special value, for example, you would need to specify:

   ```
   requestHeaders.User-Agent=user-agent-string
   ```

3. Register your request handler as described in section 5.6.

5.3 Creating a Custom oEmbed Request Handler

Maybe neither the generic oEmbed HTML template nor the site-specific template provided by the oEmbed provider meet your requirements. In this case you will need to create a custom request handler. If you make a custom request handler then you can specify exactly how to request content

from the provider and embed the provider's content into your page. Making a custom request handler requires Java programming skills.

To make a custom request handler:

1. Create a Java class that extends the abstract class **`com.escenic.livecenter.service.proxy.api.AbstractOEmbedRequestHandler`**.
2. Override the following methods:

   `protected String doOEmbedRequest(final String pContentURL, final List<Header> pHeaders) throws Exception;`
   This method requests content from the provider. Writing your own method gives you the opportunity to deal with any special requirements the provider site might have (such as supplying credentials).

   `protected String generateOEmbedCodeFromTemplate(final String pContentURL, final List<Header> pHeaders)`
   This method merges the information returned by the provider with the HTML template. Writing makes it possible to deal with complex requirements that cannot be satisfied by a simple merge process.

3. Copy **`OEmbedCustomRequestHandler.properties`** from the CUE Live installation into your **webapp** configuration layer (not the common configuration layer), and rename it appropriately. For Flickr, for example, you might enter:

```
$ cp engine-installation/plugins/live/misc/siteconfig/com/escenic/livecenter/
service/proxy/OEmbedCustomRequestHandler.properties
> /etc/escenic/engine/common/com/escenic/livecenter/service/proxy/
FlickrCustomOEmbedHandler.properties
```

4. Open the copied file for editing and set the following properties as required:

   `urlPatterns`
   A comma-separated list of host names (**not** URL patterns as listed here) for which this handler is to be used. For Flickr, for example, you might want to set this property as follows:

   ```
   urlPatterns=flickr.com,flic.kr
   ```

   `OEmbedUrlEndPoint`
   The URL to which embed requests are to be sent. These URLs are included in the list of oEmbed providers

   `blockCodeTemplate`
   An HTML tem plate from which the embedding code for your web pages will be generated. To include oEmbed property values in your template, enclose the property name in braces thus:

   ```
   {property-name}
   ```

   Alternatively, you can specify the path of a file in which you have saved the template, for example:

   ```
   blockCodeTemplate=file:/path/to/block/code/template
   ```

   The path must be specified as a file system URL (that is, it must have a **`file:`** prefix). The URL must be absolute.

   `providerName`
   The name of the provider for which you are implementing this handler (**`Flickr`**, for example). This property is mandatory if you have specified a **`blockCodeTemplate`**, otherwise it is optional.

5. Register your request handler as described in section 5.6.

To compile your custom oEmbed request handler you need to ensure that the **`live-center-core-3.2.4-2.jar`** and **`live-center-api-3.2.4-2.jar`** libraries are in your classpath.

5.4 Creating an Open Graph Request Handler

To create a site-specific Open Graph request handler:

1. Copy **`OGPRequestHandler.properties`** from the CUE Live installation into your **webapp** configuration layer (not the common configuration layer), and rename it appropriately. For Flickr, for example, you might enter:

   ```
   $ cp engine-installation/plugins/live/misc/siteconfig/com/escenic/livecenter/
   service/proxy/OGPRequestHandler.properties \
   > /etc/escenic/engine/common/com/escenic/livecenter/service/proxy/
   FlickrOGPRequestHandler.properties
   ```

2. Open the copied file for editing and set the following properties as required:

   `urlPatterns`
   A comma-separated list of host names (**not** URL patterns as listed here) for which this handler is to be used. For Flickr, for example, you might want to set this property as follows:

   ```
   urlPatterns=flickr.com,flic.kr
   ```

   `blockCodeTemplate`
   An HTML template from which the embedding code for your web pages will be generated. The template is generated using Mustache, a popular templating language. At its simplest, this means you can include Open Graph property values in your template by enclosing the property name in double braces thus:

   ```
   {{property-name}}
   ```

   Mustache also offers more sophisticated functionality such as conditional processing, allowing you to construct more complex templates.

   Instead of specifying a template directly in the **`blockCodeTemplate`** property, you can specify the path of a file in which you have saved the template. For example:

   ```
   blockCodeTemplate=file:/path/to/block/code/template
   ```

   The path must be specified as a file system URL (that is, it must have a **`file:`** prefix). The URL must be absolute.

   `providerName`
   The name of the provider for which you are implementing this handler (**`Flickr`**, for example). This property is optional: if you don't specify it, then the value returned for **`providerNameKey`** is used instead.

   `providerNameKey`
   The name of the Open Graph property that the handler is to use as a provider name. This property is optional. If you don't specify it (and you haven't specified **`providerName`** either), then the provider name is read from the provider site's **`og:site_name`** property.

3. Register your request handler as described in section 5.6.

5.5 Creating a Custom Open Graph Request Handler

If creating a request handler based on **`OGPRequestHandler`** does not meet your requirements, you can create a custom request handler, which will allow you to control exactly how the provider's content is embedded into your pages. Making a custom request handler requires Java programming skills.

To make a custom request handler:

1. Create a Java class that extends the abstract class **`com.escenic.livecenter.service.proxy.api.AbstractOGPRequestHandler`**.
2. Override the method **`buildBlockCodeImpl(String pTemplate, Map <String, String> pOGPValues)`**. This gives you full control over how the Open Graph properties supplied by the provider site are merged with your HTML template.
3. Compile your class and add it to your web application's classpath.

4. Copy **OGPRequestHandler.properties** from the CUE Live installation into your **webapp** configuration layer (not the common configuration layer), and rename it appropriately. For Flickr, for example, you might enter:

```
$ cp engine-installation/plugins/live/misc/siteconfig/com/escenic/livecenter/
service/proxy/OGPRequestHandler.properties \
> /etc/escenic/engine/common/com/escenic/livecenter/service/proxy/
FlickrCustomOGPRequestHandler.properties
```

5. Open the copied file for editing and set the properties as required. See section 5.4 for a description of the properties.
6. Register your request handler as described in section 5.6.

To compile your custom Open Graph request handler you need to ensure that the **live-center-core-3.2.4-2.jar** and **live-center-api-3.2.4-2.jar** libraries are in your classpath.

5.6 Register Request Handlers

Any request handlers you create must be registered in the **RequestHandlerFactory** in order to work. The built-in request handlers are registered as follows:

```
requestHandler.0003=./TwitterRequestHandler
requestHandler.0004=./YoutubeRequestHandler
requestHandler.0005=./InstagramRequestHandler
requestHandler.0006=./VineRequestHandler
requestHandler.0007=./VimeoRequestHandler
```

The numbers in the property names determine the order in which provider host names are matched.

To add your own request handlers:

1. Create a **RequestHandlerFactory.properties** file in your webapp configuration layer as follows:

```
$touch /etc/escenic/engine/common/com/escenic/livecenter/service/proxy/
RequestHandlerFactory.properties
```

2. Open the file for editing and add properties to register your request handlers, using different number suffixes from the built-in request handlers. For example:

```
requestHandler.0008=./FlickrRequestHandler
requestHandler.0009=./CNNRequestHandler
```

3. If you actually **want** to override one of the supplied request handlers with your own, you can do so by simply redefining the appropriate property. You can, for example, replace the supplied **TwitterRequestHandler** with your own custom implementation by entering something like:

```
requestHandler.0003=./CustomTwitterRequestHandler
```

6 Auto-Tagging Entries

CUE Live can auto-tag entries based on a variety of criteria:

- The types of CUE content to which an entry contains links (image, video, story, gallery and so on)
- The types of social media items embedded in the entry (Twitter, YouTube, Vine and so on
- Whether or not the entry is sticky
- Whether or not various boolean fields in the entry are checked

Auto-tagging is implemented by means of **transaction filters**, a standard CUE mechanism for extending Content Store functionality. A number of ready-made transaction filters to support the tagging criteria listed above are included with CUE Live, but you can also create your own transaction filters in order to support other tagging criteria. For general information about transaction filters, see Transaction Filters. For instructions on how to create your own CUE Live transaction filters, see chapter 7.

To enable auto-tagging using the supplied transaction filters you need to:

- Create a tag structure as described here. The tag structure must have the scheme (i.e name or identifier) **`livelabels@escenic.com,2015`**. Add tags to the structure by importing the supplied tag syndication file **`tags.xml`** which you will find in *engine-installation*`/plugins/live/misc/tags`. You can edit the tag labels in this file (to translate them, for example) before importing it, and you can if you wish add tag definitions of your own, but you must not delete any of the supplied tags or change their **`term`** attributes (IDs). Import the file as described here.
- Configure CUE Live to use the auto-tagging transaction filters you are interested in (see section 6.1).

6.1 Configuring the Auto-Tagging Transaction Filters

Three ready-to-use auto-tagging transaction filters are supplied with CUE Live:

`StickyAutoLabelingTransactionFilter`
This transaction filter automatically tags sticky entries.

`BooleanAutoLabelingTransactionFilter`
This transaction filter enables automatic tagging of entries in which specified boolean fields are set. See section 6.1.1 for a description of how to specify the boolean fields to be watched.

`EmbedAutoLabelingTransactionFilter`
This transaction filter automatically tags entries that contain embedded social media content. It is pre-configured to recognise and tag embedded content from Twitter, YouTube, Vimeo, Instagram and Vine. If you have added support for embedding content from other services (see chapter 5), then you can also configure **`EmbedAutoLabelingTransactionFilter`** to tag content from these services as well. For details, see section 6.1.2.

To enable the transaction filters:

1. Open **`/etc/escenic/engine/common/com/escenic/livecenter/LiveEntryDao.properties`** for editing.

2. Add entries for the transaction filters you want to enable as follows:

```
filter.stickyAutoLabeling=/com/escenic/livecenter/label/filter/
StickyAutoLabelingTransactionFilter
filter.booleanAutoLabeling=/com/escenic/livecenter/label/filter/
BooleanAutoLabelingTransactionFilter
filter.embedAutoLabeling=/com/escenic/livecenter/label/filter/
EmbedAutoLabelingTransactionFilter
```

3. Save your changes.

6.1.1 Enabling Use of BooleanAutoLabelingTransactionFilter

In order for this filter to work, you need to add **`auto-label`** elements to boolean field definitions in your **`entry-type`** resource. An **`auto-label`** element looks like this:

```
<auto-label labelName="tag-name" xmlns:cue-live="http://xmlns.escenic.com/2015/live-
center" />
```

The **`auto-label`** element must be inserted as the child of a boolean **`field`** element, and *tag-name* must be the **term** (ID) of the tag you want to use for this field. The element must belong to the **`http://xmlns.escenic.com/2015/live-center`** namespace.

For each field you mark up in this way, you must also add a tag definition to the **`livelabels@escenic.com,2015`** tag structure. You can do this by either adding a tag definition to the **`tags.xml`** file and re-importing or by adding the term manually.

To enable tagging for a "sticky" field in one of your **`entry-type`** definitions, for example, you would need to add the following to your **`entry-type`** resource:

```
<field type="boolean" name="sticky">
  <ui:label>Sticky</ui:label>
  <auto-label labelName="sticky" xmlns:cue-live="http://xmlns.escenic.com/2015/live-
center"/>
</field>
```

and then add the following tag to **`livelabels@escenic.com,2015`**:

```
<tag term="sticky">
  <label>Sticky</label>
</tag>
```

6.1.2 Adding New Social Media Services to EmbedAutoLabelingTransactionFilter

If you have extended CUE Live to support embedding content from additional social media services (see chapter 5), then you can also configure **`EmbedAutoLabelingTransactionFilter`** to tag embedded content from these services. To do so:

1. Add a new properties file to your common configuration layer, **`/etc/escenic/engine/common/com/escenic/livecenter/label/filter/EmbedAutoLabelingTransactionFilter.properties`**, and open it for editing.

2. For each service you want to support, add a property like this:

   ```
   socialLabels.provider-name=tag-name
   ```

   where:

   - *provider-name* is the provider name specified in your embedding request handler configuration (see chapter 5)
   - *tag-name* is the **term** (ID) of the tag you want to use for this service.

3. Add a tag definition for the term to the **livelabels@escenic.com,2015** tag structure. You can do this by either adding a tag definition to the **tags.xml** file and re-importing or by adding the term manually.

You could add support for a Flickr embedding extension like one of those described in chapter 5, for example, by add the following entry to **EmbedAutoLabelingTransactionFilter.properties**:

```
socialLabels.flickr=flickr
```

and then adding the following tag to **livelabels@escenic.com,2015**:

```
<tag term="flickr">
  <label>Flickr</label>
</tag>
```

7 CUE Live Transaction Filters

A **transaction filter** is a user-defined function that gets executed whenever certain operations (or transactions) are performed, and can thereby modify the outcome of those operations. It is a standard Content Store mechanism for modifying and extending Content Store behavior, and is described in detail in the CUE Content Store Advanced Developer Guide.

The CUE Live can also be extended/modified using transaction filters. The main reason you might want to make use of them would be in order to integrate CUE Live with an external system in some way: for example to copy all CUE Live events to an external system as they are created.

The CUE Live transactions that can be modified using transaction filters are:

- Object creation
- Object update
- Object deletion

A transaction filter is implemented as a Java class. A transaction filter can, for example:

- Modify a CUE Live entry as it is being saved
- Prevent a CUE Live entry from being deleted unless certain criteria are met
- Perform additional actions when a live entry has been created

7.1 Making A Transaction Filter

For general background information on making transaction filters, see the CUE Content Store Advanced Developer Guide.

To make a CUE Live transaction filter, create a Java class that extends the abstract class **`com.escenic.livecenter.TransactionFilterService`**. This is a convenience class containing empty "do nothing" implementations of the methods defined in the **`com.escenic.livecenter.TransactionFilter`** interface:

`beforeCreate(`*`pLiveEntry`*`)`
Called immediately before a new entry is saved.

`beforeUpdate(`*`pLiveEntry`*`)`
Called immediately before changes to an existing entry are saved.

`beforeDelete(`*`pLiveEntry`*`)`
Called immediately before an existing entry is deleted.

`afterCreate(`*`pLiveEntry`*`)`
Called immediately after a new entry is saved.

`afterUpdate(`*`pLiveEntry`*`)`
Called immediately before changes to an existing entry are saved.

`afterDelete(`*`pLiveEntry`*`)`
Called immediately before an existing entry is deleted.

isEnabled
Called by the Content Store to determine whether or not the filter is currently enabled.

All you need to do in your class is re-implement the method(s) that you are interested in. In the **before** methods you can query the entry and modify it. In the case of **beforeCreate** and **beforeUpdate**, any changes you make are reflected in the saved object.

Before a transaction filter can be used it must be:

- Compiled
- Added to the Content Store's classpath

7.2 Using a Transaction Filter

Create a property file for your transaction filter, for instance *configuration-root*/**com/mycompany/MyFilter.properties**. The file must at least contain a **$class** entry to identify the class that implements the filter, for example:

```
$class=com.mycompany.transactionFilters.MyTransactionFilter
```

The transaction filters executed by the Content Store are defined in a file called *configuration-root*/**com/escenic/livecenter/LiveEntryDao.properties**. To enable this filter, you would need to add the following entry to the file:

```
filter.myfilter=/com/mycompany/transactionfilters/MyTransactionFilter
```

8 Publishing Events

CUE Live events are different from ordinary Content Store content items in two ways:

- Events have a specialized internal structure, as defined in the **`entry-type`** resource
- Events are "live" content items that update themselves without any user intervention

These differences are reflected in the presentation layer used to generate the event feeds on published event pages, which is different from the standard Content Store presentation layer. The basis of the CUE Live presentation layer is its presentation web service – a REST web service that serves event feed content as JSON data. For a description of this web service, see section 9.2.

You can include event feeds in your publications in the following ways:

Using the Widget Framework
The Widget Framework (version 3.6 or higher) includes a Live Entries view that you can use to include event feeds in event page templates. It is very simple to use and completely hides all details of using the CUE Live presentation web service. If you already use the Widget Framework, then it is the obvious choice. For details see the Widget Framework documentation.

Using the `livecenter-presentation-js` component
This Javascript library is available in the CCI Europe Maven repository. By using it in your event page JSP templates, you can again hide the details of using the CUE Live presentation web service. For further information on how to use the library, see section 8.1.

Using the presentation web service directly
If neither of the above methods meet your requirements, then you can write your own client-side code for accessing and displaying the event feed data exposed via the presentation web service. See section 9.2 for further information.

8.1 livecenter-presentation-js

`livecenter-presentation-js` is a Javascript library that you can use to manage the display of event feeds in Escenic publications. It provides all the functionality needed to retrieve event entries from the CUE Live presentation web service and render the entries as HTML content. It supports the use of SSE if it is enabled (see section 4.1.1) and otherwise polls the web service to keep feeds up-to-date. **`livecenter-presentation-js`** is used in the demo publication included in the CUE Live distribution.

`livecenter-presentation-js` is an AngularJS component and is very straightforward to use. It includes a custom **`livecenter-feed`** element that encapsulates all of an event feed's HTML layout. Assuming you already have a working CUE Live installation with a correctly defined event content type and corresponding event type definitions, then all you need to do to use the **`livecenter-feed`** element in your event pages is:

1. Add the following dependencies to your publication's **`pom.xml`** file:

```
<dependency>
  <groupId>com.escenic.plugins.live</groupId>
  <artifactId>live-center-presentation-js</artifactId>
  <version>${project.version}</version>
```

```
    <type>war</type>
  </dependency>
  <dependency>
    <groupId>org.webjars</groupId>
    <artifactId>jquery</artifactId>
    <version>1.10.2</version>
  </dependency>
  <dependency>
    <groupId>org.webjars</groupId>
    <artifactId>angularjs</artifactId>
    <version>1.2.7</version>
  </dependency>
  <dependency>
    <groupId>org.webjars</groupId>
    <artifactId>URI.js</artifactId>
    <version>1.12.0</version>
  </dependency>
```

2. Add the following filters to your publication's **`web.xml`** file:

```
<filter>
  <filter-name>/com/escenic/servlet/filter/JarResourceFilter</filter-name>
  <filter-class>com.escenic.servlet.filter.JarResourceFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>/com/escenic/servlet/filter/JarResourceFilter</filter-name>
  <url-pattern>/webjars/*</url-pattern>
</filter-mapping>
```

3. Download the Moment.js library and add it to your publication project in the following location:

   `src/main/webapp/js/lib/moment.min.js`

4. Create an AngularJS app that uses the **`livecenter-presentation-js`** library to render your event feeds. A simple example of how to do this would be to add a Javascript file called **`src/main/webapp/js/all.js`** to your project, containing the following code, which creates an AngularJS app called **`LiveDemo`**:

```
var livedemo;
(function (livedemo) {
  livedemo.LiveDemo = angular.module('LiveDemo', [
    'livecenter.presentation'
  ]);
})(livedemo || (livedemo = {}));

var livedemo;
(function (livedemo) {
  var EntryList = (function () {
    function EntryList($scope) {
      this.$scope = $scope;
      var config = {
        liveLabelTagSchemeName: "tag:livelabels@escenic.com,2015",
        loadMoreStyle: 'button'
      };
      $scope.config = config;
    }
    EntryList.$inject = ['$scope'];
    return EntryList;
  })();
  livedemo.EntryList = EntryList;
  livedemo.LiveDemo.controller('EntryList', EntryList);
```

```
})(livedemo || (livedemo = {}));
```

See chapter 4 for a complete list of configuration options.

If you prefer to use Typescript, then the same app would look like this:

```
module livedemo {
 export var LiveDemo = angular.module('LiveDemo', [
   'livecenter.presentation'
 ]);
}

module livedemo {
 import FeedConfig = livecenter.presentation.FeedConfig;
 export class EntryList {
   public static $inject = ['$scope'];

   constructor(private $scope:any) {
     var config:FeedConfig = {
       liveLabelTagSchemeName: "tag:livelabels@escenic.com,2015",
       loadMoreStyle: 'button'
     };
     $scope.config = config;
   }
 }
 LiveDemo.controller('EntryList', EntryList);
}
```

5. Create a JSP file for the event feed template (**`/src/main/webapp/template/article-template/livefeed.jsp`**, for example) and add the following code:

```
<%@ page language="java" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<c:set var="liveEventStatus"
 value="${requestScope.article.fields.liveEventStatus.value}"/>
<c:if test="${empty requestScope.article.fields.liveEventStatus.value}">
 <c:set var="liveEventStatus" value="false"/>
</c:if>

<div data-ng-controller="EntryList">
 <!-- The component to render live-center feeds -->
 <livecenter-feed
     eventUrl="${requestScope.article.fields.livecenter.value}"
     eventStatus="${liveEventStatus}"
     contextPath="/demo-temp-dev"
     config="config">
 </livecenter-feed>
</div>

<!-- javascripts supplied from webjars. We need to add a filter in web.xml to
 support /webjars urls -->
<c:url var="angular" value="/webjars/angularjs/1.2.7/angular.min.js"
 scope="request"/>
<c:url var="angular_sanitize" value="/webjars/angularjs/1.2.7/angular-
sanitize.min.js" scope="request"/>
<c:url var="uri_js" value="/webjars/URI.js/1.12.0/URI.min.js" scope="request"/>
<c:url var="jquery" value="/webjars/jquery/1.10.2/jquery.min.js" scope="request"/>
<!-- javascripts we have packaged inside our project -->
```

```
<c:url var="moment" value="/js/lib/moment.min.js" scope="request"/>
<c:url var="all_js" value="/js/all.js" scope="request"/>
<!-- javascript and style we use from live-center-presentation-js -->
<c:url var="feed_js" value="/live-center-presentation-js/js/all.js"
 scope="request"/>
<c:url var="presentation_js_css" value="/live-center-presentation-js/css/
style.css" scope="request"/>
<link href="${presentation_js_css}" rel="stylesheet">

<script src="${angular}"></script>
<script src="${angular_sanitize}"></script>
<script src="${uri_js}"></script>
<script src="${moment}"></script>
<script src="${feed_js}"></script>
<script src="${all_js}"></script>
<script src="${jquery}"></script>
```

6. Declare your AngularJS app (**`LiveDemo`** in this case) by adding a **`data-ng-app`** attribute to one of your live feed template's ancestor elements. You can do this either by wrapping the template in a **`div`** element:

   ```
   <div data-ng-app="LiveDemo">
     ...
   </div>
   ```

   or by simply adding **`data-ng-app="LiveDemo"`** to an existing ancestor element (the **`html`** element in **`/src/main/webapp/template/article-template/header.jsp`**, for example.

7. Build and deploy your publication.

8.1.1 Configuration

`livecenter-presentation-js` has a number of configuration options that you can set in your AngularJS app. The example app in section 8.1 shows just one option being set:

```
var config = {
  liveLabelTagSchemeName: "tag:livelabels@escenic.com,2015",
  loadMoreStyle: 'button'
};
```

but a number of other options are available:

```
timeStyle?: string; // {'timestamp', 'relative'}
timestampFormat?: string;
showTags?: boolean;
liveLabelTagSchemeName?: string;
showLiveLabels?: boolean;
imageRepresentationName?: string;
//pollingInterval in millisecond
pollingInterval?: number;
//Stop polling after X minutes inactivity
stopPollingAfterInactive?: number;
changelogSize?: number;
showAuthor?: boolean;
showCreator?: boolean;
showAvatar?: boolean;
loadMoreStyle?: string; // {'scroll', 'button'}
```

9 Using The CUE Live Web Services

CUE Live provides two REST API web services:

- An editorial web service with full read-write access to entries. This web service can be used to implement custom user interfaces that meet requirements not satisfied by the CUE Live webapp, or for integrating CUE Live with third-party systems. Access to this web service requires authentication as for the Content Store web service.
- A presentation web service with more limited access to entries. This web service is intended to fulfil the same purpose as the Java bean-based JSP presentation layer used by the Content Store and most other Content Store plug-ins, while allowing you to use whatever languages and UI technologies you choose to build your web pages. A similar presentation web service is planned for future versions of the Content Store. The presentation web service is effectively a subset of the editorial web service. No authentication is required to access this web service.

9.1 The Editorial Web Service

The CUE Live editorial web service is very similar to the Content Store web service, with the main difference being that instead of returning content in the form of XML Atom feeds, it returns JSON data. If you are familiar with the Content Store web service, then you will find the CUE Live web service easy to use. If you haven't used the Content Store web service before, then you should probably start by reading at least the introduction of the CUE Content Store Integration Guide.

The editorial web service provides a standard REST HTTP API in which all operations are performed by sending **`GET`**, **`POST`**, **`PUT`** or **`DELETE`** requests to various URLs. The default name of the web service is **`live-center-editorial`**.

Unlike the Content Store web service, the CUE Live editorial web service has no global entry point or "start" URL. Before you do anything with the web service, therefore, you usually need to obtain the ID of an Event content item (either from the Content Store presentation layer or the Content Store web service). Once you have an Event id you can request information about it by submitting a **`GET`** request like this to the web service:

```
http://host-ip-address/live-center-editorial/event/event-id
```

where *host-ip-address* is your CUE Live host name or IP address and *event-id* is the ID of the event you are interested in. Authentication is required, so if you submitted the request using **`curl`**, then the whole command for an event with the id **`24`** would look like this:

```
curl -u user:password -X GET http://host-ip-address/live-center-editorial/event/24
```

CUE Live then returns a JSON structure that looks something like this:

```
{
    "id": 24,
    "title": "Liverpool - Manchester United",
    "entries": "http://host-ip-address/live-center-editorial/event/24/entries",
    "publishedEntries": "http://host-ip-address/live-center-editorial/event/24/
entries?status=published"
```

```
    "unPublishedEntries": "http://host-ip-address/live-center-editorial/event/24/
entries?status=unpublished",
    "self": "http://host-ip-address/live-center-editorial/event/24",
    "changelog": "http://host-ip-address/live-center-editorial/changelog/event/24",
    "entryModel": "http://host-ip-address/live-center-editorial/model/24/football",
    "values": {
        "title": "Liverpool - Manchester United",
        "body": "<p>Match: 23. January 2015</p>"
    },
    "subscriptions": [
        {
            "link": "http://host-ip-address/live-center-editorial/event/24/
subscription/0",
            "source": "twitter"
        },
        {
            "link": "http://host-ip-address/live-center-editorial/event/24/
subscription/1",
            "source": "twitter"
        }
    ]
}
```

The actual **content** of this document, as in all the JSON structures returned by the web service, is in the **`payload`** field. In this case **`payload`** contains the field values of the Event content item - a title and a description. The rest of the document mostly consists of links that you can follow to obtain further information. Submitting a new **`GET`** request to the **`entries`** URL, for example:

```
curl -u user:password -X GET http://host-ip-address/live-center-editorial/event/24/
entries
```

This returns a JSON document containing all the event's entries plus related information:

```
{
  "entries":[
    {
      "author":{
        "value": "A.N. Other",
        "origin": "http://host-ip-address/webservice/escenic/person/7"
      },
      "creator":{
        "value": "livedemo Administrator",
        "origin": "http://host-ip-address/webservice/escenic/person/1"
      },
      "creationDate": "2015-02-13T12:20:24.000+0000",
      "eTag": "0d1f1e1f-6bf2-46fd-8de4-c1de0c015f0a",
      "event": "http://host-ip-address/live-center-editorial/event/24",
      "lastModifiedDate": "2015-02-13T12:36:38.000+0000",
      "model": "http://host-ip-address/live-center-editorial/model/24/football",
      "parent": "http://host-ip-address/live-center-editorial/entry/251",
      "values":{
          "basic": "Test entry"
        },
      "publishDate": "2015-02-13T12:20:24.000+0000",
      "self": "http://host-ip-address/live-center-editorial/entry/283",
      "state": "published",
      "tags":[]
    },
    ...(more entries)...
```

```
    ]
}
```

Like many REST APIs, the CUE Live API is more or less self-documenting. It consists entirely of URLs, and the fields in the returned JSON documents have reasonably self-explanatory names. A good way of learning the API is to install a REST API browser extension such as DHC for Chrome, and simply explore it.

Usage of the HTTP commands follows standard rest conventions:

- Send **`GET`** to a URL to retrieve a resource (as in the examples above)
- Send **`PUT`** to a URL to modify a resource
- Send **`POST`** to a URL to create a new resource
- Send **`DELETE`** to a URL to delete a resource

9.1.1 Retrieving an Entry

To retrieve a single Entry, rather than a list of all the Entries in an Event, you send **`GET`** to the Entry's **`self`** URL:

```
curl -u user:password -X GET http://host-ip-address/live-center-editorial/entry/283
```

This returns a single Entry, rather than an array of entries:

```
{
  "author":{
    "value": "A.N. Other",
    "origin": "http://host-ip-address/webservice/escenic/person/7"
  },
  "creator":{
    "value": "livedemo Administrator",
    "origin": "http://host-ip-address/webservice/escenic/person/1"
  },
  "creationDate": "2015-02-13T12:20:24.000+0000",
  "eTag": "0d1f1e1f-6bf2-46fd-8de4-c1de0c015f0a",
  "event": "http://host-ip-address/live-center-editorial/event/24",
  "lastModifiedDate": "2015-02-13T12:36:38.000+0000",
  "model": "http://host-ip-address/live-center-editorial/model/24/football",
  "parent": "http://host-ip-address/live-center-editorial/entry/251",
  "values":{
      "basic": "Test entry"
    },
  "publishDate": "2015-02-13T12:20:24.000+0000",
  "self": "http://host-ip-address/live-center-editorial/entry/283",
  "state": "published",
  "tags":[]
}
```

9.1.2 Changing an Entry

To change an Entry you retrieve the resource as described in section 9.1.1, make whatever change you require and send the modified document back to the same URL as a **`PUT`** request. Using **`curl`**, for example, you would save the modified JSON data in a file (say **`edited-entry.json`**) and then resubmit it like this:

```
curl --include -u user:password -X PUT -H 'If-Match: "0d1f1e1f-6bf2-46fd-8de4-
c1de0c015f0a"' \
> -H 'Content-Type: application/json' http://host-ip-address/live-center-editorial/
entry/283 \
> --upload-file edited-entry.json
```

Note the two headers that are included with the request:

`Content-Type: application/json`
: You must always specify the MIME type of the data you are submitting.

`If-Match: "0d1f1e1f-6bf2-46fd-8de4-c1de0c015f0a"`
: The **`If-Match`** header is used by CUE Live to manage concurrency. You must always return the exact value supplied in the **`eTag`** field of the Entry you are modifying, **including the quotes surrounding the value**. CUE Live uses the **`If-Match`** header is used in exactly the same as the Content Store. For a detailed explanation of what it is used for and how it works, see http://docs.escenic.com/ece-integration-guide/7.10/optimistic_concurrency.html.

9.1.3 Creating an Entry

To create a new Entry you create a correctly structured JSON data set and send it to the entries list URL of the Event you want to add it to. The entry must be sent as a **`POST`** request. Many fields can be omitted from the data structure you submit, since they contain system generated values. All you really need to include to create an entry, is the **`values`** field:

```
{
  "values":{
      "basic": "This is a new entry."
    }
}
```

Using **`curl`** you would save the modified JSON data in a file (say **`new-entry.json`**) and submit it like this:

```
curl --include -u user:password -X POST -H "Content-Type: application/json" \
> http://host-ip-address/live-center-editorial/event/24/entries --upload-file new.json
```

You need to include a **`Content-Type`** header with the request, specifying the data MIME type (**`application/json`**).

> The JSON format requires you to escape any quote marks in field values using a preceding backslash (**`\`**). This is commonly required when inserting HTML into rich text fields, for example:
>
> ```
> {
>   "values":{
>       "xhtml": "<p class=\"myclass\">Hello world</p>"
>     }
> }
> ```

Embedding Foreign Content

CUE Live allows foreign content from social media sites such as Twitter and YouTube to be embedded in entries' rich text fields (see section 2.2.2.2). To do this when editing a JSON values property, you must insert the foreign content's URL as an XHTML anchor (**`a`**) element, and include two special CSS classes:

`esc-social-embed`
This class indicates that the link is to be rendered as embedded content rather than an ordinary link.

`esc-tag-`*`providerName`*
Where *providerName* is one of **`Youtube`**, **`Twitter`**, **`Vimeo`**, **`Instagram`** or **`Vine`**. This identifies the source of the foreign content. No other sources of foreign content are currently supported.

The following example shows a values property containing an embedded YouTube video:

```
"values":{
    "xhtml": "<p><a class=\"esc-social-embed esc-tag-Youtube\" href=\"https://
www.youtube.com/watch?v=yVwAodrjZMY"></a></p>"
  }
```

9.1.4 Deleting an Entry

To delete an Entry, you send a **`DELETE`** request to the Entry's **`self`** URL:

```
curl -u user:password -X DELETE http://host-ip-address/live-center-editorial/entry/283
```

No additional headers are required to delete an Entry.

9.2 The Presentation Web Service

The presentation web service provides a standard REST HTTP API in which all operations are performed by sending **`GET`** requests to various URLs. Since the presentation web service is read-only, no other request types are allowed. The default name of the web service is **`live-center-presentation-webservice`**.

Like the editorial web service, the CUE Live presentation web service has no global entry point or "start" URL. You will usually access it by obtaining a URL from the Content Store's (JSP) presentation layer as follows:

```
${article.fields.livecenter.value}
```

Assuming **`${article}`** is an Event content item, then this will return a URL like this:

```
http://host-ip-address/live-center-presentation-webservice/event/event-id/entries
```

where:

- *host-ip-address* is the IP address and port number of the Content Store
- *event-id* is the content item ID of the event

Submitting a **`GET`** request to such a URL returns a JSON structure containing the event's top 10 entries, something like this:

```
{
  "entries": [
    {
      "author":
        {
```

```
          "value": "livedemo Administrator",
          "origin": "http://host-ip-address/webservice/escenic/person/1"
        },
      "creator":
        {
          "value": "livedemo Administrator",
          "origin": "http://host-ip-address/webservice/escenic/person/1"
        },
      "creationDate": "2015-05-14T12:23:18.000+0000",
      "eTag": "ccd54879-ac58-4e86-bdf2-3f00901e5e17",
      "lastModifiedDate": "2015-05-14T12:23:18.000+0000",
      "publishDate": "2015-05-14T12:23:18.000+0000",
      "payload": [
        {
          "name": "basic",
          "value": "This is the title"
        }
      ],
      "state": "published",
      "tags": [ ],
      "sticky": false
    }
  ],
  ...(up to 9 more entries)...
  "self": "http://host-ip-address/live-center-presentation-webservice/event/3/entries?
count=10",
  "after": "http://host-ip-address/live-center-presentation-webservice/event/3/
entries/after/2015-05-14T12:23:18.000+0000?count=10",
  "before": "http://host-ip-address/live-center-presentation-webservice/event/3/
entries/before/2015-05-14T12:23:18.000+0000?count=10",
  "changelog": "http://host-ip-address/live-center-presentation-webservice/changelog/
event/3"
  "sseURI": "http://host-ip-address/live-center-presentation-webservice/
changelogSSE/3"
}
```


The returned structure contains the following fields:

`entries`
An array of **`entry`** structures. For a event with 10 or fewer entries, all entries are returned. If there are more than 10 entries, then only the top 10 entries are returned. The entries are sorted in the order they should appear on the event page - with sticky entries at the top and then ordinary entries sorted by **`publishDate`** (or **`creationDate`** for unpublished entries), most recent first. For a description of the fields in the entries, see section 9.2.1.

> Entries are returned 10 at a time by default. You can, however, change this page length — see section 4.1.5.2 for details.

`self`
This structure's presentation web service URL.

`after`
The presentation web service URL for the 10 entries **above** the current set of entries: that is, entries that should appear above the current set on the event page. If you want to specify a different maximum number of entries to retrieve, just change the URL's **`count`** parameter.

> You can use this URL for implementing paging functionality (displaying more entries when the user reaches the bottom of the page, for example). To poll for new entries or changes use the **`changelog`** URL (see section 9.2.2.2).

`before`
The presentation web service URL for the 10 entries **below** the current set of entries: that is, entries that should appear below the current set on the event page. If you want to specify a different maximum number of entries to retrieve, just change the URL's **`count`** parameter.

`changelog`
The presentation web service URL of this event's change log. See section 9.2.2.2 for details.

`sseURI`
The presentation web service URL of this event's server-sent events (SSE) connection. See section 9.2.2.1 for details.

9.2.1 Entry Fields

Each entry in the **`entries`** array contains the following fields.

`creator`
The name and CUE web service URL of the Content Store user who created the entry.

`author`
The name and CUE web service URL of the Content Store user to whom authorship of the entry is attributed. By default this is the same user as **`creator`** but can be different if a different author has been explicitly selected in CUE Live.

`creationDate`
The date the entry was created.

`eTag`
A system generated value. Used for client-side caching.

`lastModifiedDate`
The date the entry was last modified.

`publishDate`
The date the entry was published.

`payload`
The actual content of the entry, an array of fields. In the example shown in section 9.2, the entry consists of only one basic (plain text) field. An entry more often contains several fields, at least one of which is a rich text field containing XHTML markup like this:

```
{
  "name": "body",
  "value": "<p>Here is some XHTML text.</p>"
}
```

`state`
The state of the entry. Currently this may either be **`"published"`** or **`"deleted"`**.

`tags`
An array of tags. The example shown in section 9.2 has no tags, so the array is empty. An array with content looks like this:

```
"tags": [
  {
    "value": "Twitter",
    "origin": "http://host-ip-address/webservice/escenic/classification/tag/
tag:livelabels@escenic.org,2015:twitter"
  }
]
```

Each tag consists of the tag's label and the tag's CUE web service URL.

`sticky`
Whether or not the entry is sticky.

9.2.2 Keeping the Event Page Up-to-Date

Once you have retrieved the initial entries to be displayed, you need to keep the event page up-to-date. This can be done in one of two ways:

Using Server-sent Events (SSE)
Server-sent events is a standard technology introduced with HTML 5 that makes it possible to keep a web page updated with dynamic content without resorting to polling. Instead, a client can open a persistent link to the server and receive notifications of changes via that link whenever they occur. This is the recommended method of keeping event pages updated. For details, see section 9.2.2.1.

Polling the event change log
This is the original method used to keep CUE Live events up-to-date. It has scalability issues, however, and is no longer the recommended approach - certainly not for blogs that can expect large numbers of visitors. For details see section 9.2.2.2.

9.2.2.1 Using Server-sent Events

To access the change log via SSE, first send a GET request to the **`changelog`** URL. From this you can obtain a new URL, the change log's **`previous`** link. Without SSE, you would need to poll this **`previous`** link at intervals in order to get details of new entries. Using SSE, polling is not required. Instead, you now send a **`GET`** request to the **`sseURI`** URL. This sets up a persistent link over which the Content Store can send notifications.

In the browser, an **`EventSource`** object can be used to receive server-sent event notifications. On creation, the object is connected to the source of notifications (in our case, a particular live blog's SSE link, **`http://`***host-ip-address*`/live-center-presentation-webservice/changelogSSE/`***`event-id`**). It has an **`onmessage`** event that is fired every time a notification is received. So all you need to do is write an event function that responds appropriately to the notifications received. For example:

```
var source = new EventSource("http://host-ip-address/live-center-presentation-
webservice/changelogSSE/event-id");
source.onmessage = function(event) {
    // handle the SSE notification here:
};
```

The event data does not directly contain any details of what change has occurred - the notification simply indicates that a change has been added to the change log. Your event code can then retrieve details of the change by sending a **`GET`** request to the change log's **`previous`** URL. The **`GET`** request sent to the change log **`previous`** URL is identical to an ordinary polling request, except that in this case, the response is guaranteed to contain some changes.

> Server-sent events reduce the load of responding to large numbers of polling requests, but at the price of requiring the Content Store to hold large numbers of persistent connections open. This can quickly become a problem, since Tomcat cannot handle more than 200 simultaneous client connections. CCI Europe therefore now ships an **SSE Proxy** with the Content Store. An SSE Proxy can handle up to 28,000 simultaneous client connections on behalf of the Content Store. It is

possible to run many such proxies in parallel, each of which only require one connection to the Content Store, effectively removing any limitation on the total number of SSE connections that it is possible to handle.

For details of how to set up and run one or more SSE proxies for use with the Content Store, see the SSE Proxy documentation.

9.2.2.2 Polling The Event Change Log

To access the event change log by polling, send a request to the **changelog** URL. This returns a JSON structure like this:

```
{
  "next": "http://host-ip-address/live-center-presentation-webservice/changelog/
event/18/before/458?count=10",
  "previous": "http://host-ip-address/live-center-presentation-webservice/changelog/
event/18/after/472?count=10",
  "self": "http://host-ip-address/live-center-presentation-webservice/changelog/
event/18/before/473?count=10",
  "entries": [...]
}
```

The **entries** will contain **entry** structures for any entries that have changed plus any new entries created since your last request. You can use this information to update your page. You should then send a request to the change log structure's **previous** URL, and it will return a new change log structure containing any newer changes.

By polling the **previous** URL at regular intervals you can keep your page up-to-date with all changes made to the event.

The change log sorts entries by **lastModifiedDate**, not by **sticky** and **publishedDate**/**creationDate**. That is why you need to use the **changelog** URL to keep your page up-to-date, and not the **after** URL.

The event change log functions in exactly the same way as the Content Store web service's change log. If you want a more detailed description of how it works, see Change Logs.

Currently, the CUE Live change log does not take account of sticky entries correctly. If an event contains 2 sticky entries and you request 5 entries, then a total of 7 entries are returned.

9.2.3 Retrieving Embedded Content

The CUE Live web service includes a proxy service for formatting embedded content. The proxy URL is:

```
http://host-ip-address/live-center-presentation-webservice/proxy/
```

The proxy service, merges the embedded content URL with the appropriate HTML template and returns the resulting HTML snippet so all you have to do is insert it at the correct location.

A rich text field containing an embedded tweet looks something like this:

```
{
  "name": "xhtml",
  "value": "<p>Look, a tweet!</p>
```

```
            <p>
              <a xmlns=\"http://www.w3.org/1999/xhtml\"
                 href=\"https://twitter.com/threadless/status/606078523548266496\"
                 class=\"esc-social-embed esc-tag-Twitter\">
              </a>
            </p>"
}
```

To display the tweet, all you need to do is to pass its URL on to the proxy service like this:

```
http://host-ip-address/live-center-presentation-webservice/proxy/?url=https://
twitter.com/threadless/status/606078523548266496
```

The proxy service will then respond with a JSON structure containing the merged HTML:

```
{
  "html":"<blockquote class="twitter-tweet" lang="en">
            <a href="https://twitter.com/threadless/status/596560045782990848"></a>
          </blockquote>
          <script type="text/javascript" src="http://platform.twitter.com/
widgets.js"></script>",
  "provider_name":"Twitter"
}
```

You can then just replace the original **<a/>** element with the content of the **xhtml** field.

9.2.4 Retrieving Selected Entries

The CUE Live provides a web service for retrieving selected entries from an event. Currently, it allows you to select entries by tag. The web service is located at:

```
http://host-ip-address/live-center-presentation-webservice/event/eventId/search
```

where:

- *host-ip-address* is the IP address and port number of the Content Store
- *event-id* is the content item ID of the event from which you want to retrieve events

To retrieve all entries tagged with a particular tag, append a **tagId** parameter to the URL, specifying the tag **term**. For example:

```
http://host-ip-address/live-center-presentation-webservice/event/265/search/?
tagId=tag:livelabels@escenic.com,2015:sticky
```

The web service will then return a JSON structure containing any entries that are tagged with the specified tag. The entries are returned 10 at a time in exactly the same way as ordinary entry requests (see section 9.2).

Note that you must specify a tag **term** in the request URL, not a tag label. For information on how to find tag terms, see Search For a Tag.

You can include more than one tag term in the request URL by repeating the **tagId** parameter. For example:

```
http://host-ip-address/live-center-presentation-webservice/event/265/search/?
tagId=tag:livelabels@escenic.com,2015:sticky&tagId=tag:livelabels@escenic.com,2015:twitter
```

The returned JSON structure will then contain any entries tagged with one or more of the specified tags.

