Live Center

User Guide

2.0.0-8

escenic

A CCI COMPANY

Table of Contents

1 Introduction

Live Center is a liveblogging application based on the Escenic Content Engine. It consists of the the Live Center webapp plus supporting extensions to the Content Engine. The Live Center app provides bloggers, journalists and editors with a purpose-built, streamlined liveblogging platform. Since it 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.

Live Center 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 Live Center 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 Live Center webapp 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 Live Center is installed and configured, it is very easy to use. Events are created and published using Content Studio, in exactly the same way as other content items.

1.1 Browser Support

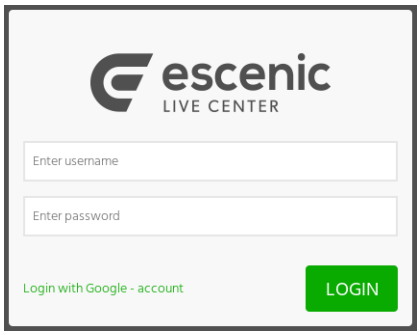
Some problems have been reported when using the current version of the Live Center editor in both Firefox and Safari. For the moment, therefore, we recommend the use of Chrome with the Live Center editor. Note that this limitation only applies to editing: published Live Center blogs can of course be viewed with any browser on any device.

2 Using Live Center

To launch Live Center open a browser window on whatever device you are using, and point it at:

```
http://your-server/live-center/
```

A login dialog is displayed:



Log in using your normal Content Studio user name and password.

Once you have logged in, a list of current events is displayed. Above the list is a filter control that you can use to filter the contents of the list if it's very long. Select **> Filter** (top left) to open the filter form and set your search criteria. Once the filter is set up, you switch it on and off by selecting the button (top right). See section 2.1 for information about how to use the event filter.

Find the event you want, click on to open it, and start blogging.

Using the Live Center webapp

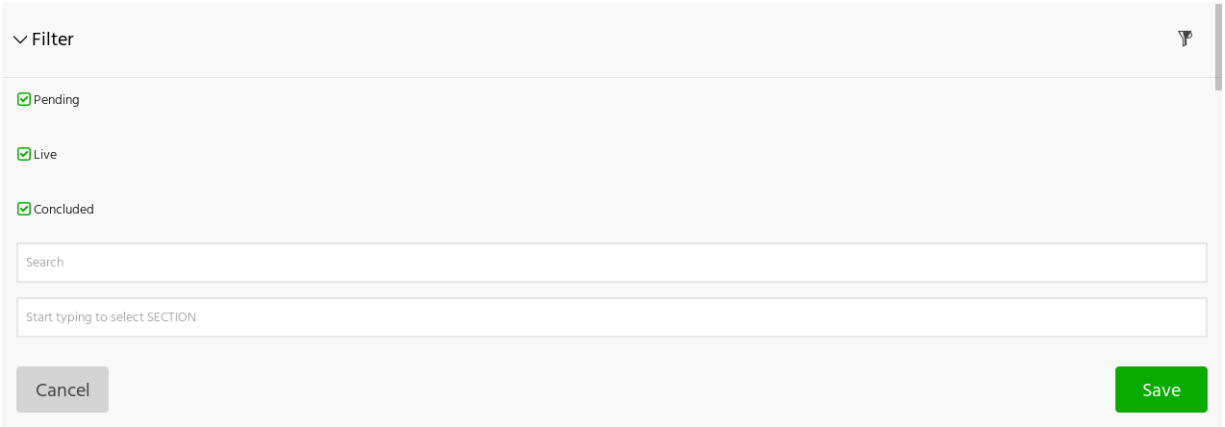
For information about writing and editing blogs in Live Center, see section 2.2.

To return to the list of current blogs, select the menu button ( ) in the top left corner and then select **Events**.

To log out of Live Center, select the menu button ( ) in the top left corner and then select **Logout** (right at the bottom of the screen).

2.1 Filtering the Event List

To set up a filter for the event list, click on the **> Filter** button at the top of the event list, on the left. The following form is then displayed:



The form contains the following filtering options:

Pending
Check this option if you want the list to include events that have not yet gone live (that is, the blog is not yet published or visible on the web).

Live
Check this option if you want the list to include events that are currently published on the web.

Concluded
Check this option if you want the list to include events that are over. Concluded events can no longer be edited, but are still visible on the web site.

Search
You can enter a search string in this field. The event list will then only contain events with titles that contain the string you entered.

Section
You can use this field to select a section from your live blog's publication. The event list will then only contain events that belong to the selected section. To select a section, start typing its name and then select it from the list of displayed options.

You can combine all of these fields to create a more complex filter. You could, for example, uncheck **Pending** and **Concluded**, Enter "Manchester" in the **Search** field and select the "Sports" section of your publication. The list would then only contain live events belonging to the "Sports" section with titles including the string "Manchester".

Set up the filter as you want and then select **Save** to hide the options. Once the filter is set up, you can switch it on and off by selecting the button (on the right).

2.2 Writing in Live Center

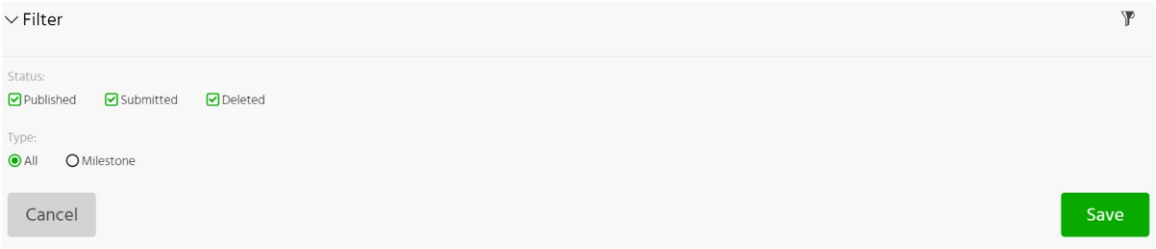
Writing and editing entries in Live Center is very simple. You simply fill out the fields displayed above the **Submit** button at the top of the window and when you are satisfied with your entry, select

Submit to submit it for approval to your editor. Exactly which fields are displayed above the **Submit** button will vary according to the type of event you are covering, but it will usually include at least one rich text field, often called **Body**. A rich text field has a palette of formatting buttons above it that you can use for simple text styling and the insertion of hyperlinks.

Some fields (the less frequently-used ones) are hidden from view in order to reduce clutter in the user interface. The hidden fields can be displayed by selecting the button displayed in the top right of the entry editor. The hidden fields always include an **Author** field. This field displays a drop-down list containing the names of all registered Content Studio / Live Center users, from which you can select a name. Use of this field is optional, and how you should use it will be determined by your publication designer or editor. If you don't select anything then your name will be displayed as the author of the entry by default, but you can choose a different name if required. Whether you use this field or not, and whichever name you select from it, the system will record you as the **creator** of the entry.

Some fields may be defined as having a maximum length (in characters). If this is the case, then the limit is displayed in the bottom right corner of the field. As you type in the field the number decreases, to show how many characters you have left. The limit is only advisory: if you exceed it, then the field is highlighted to indicate that is too long, but the entry can still be submitted and published. Exceeding the limit will, however, usually mean there is a risk of formatting issues in the published blog.

When you click on **Submit**, the entry you have been working on is moved to the top of the **event feed**, displayed immediately below the **Submit** button. It is highlighted to indicate that it is only submitted for review, and not yet published. Once it is published, the highlighting is removed. Entries can be in any of the following states: **submitted**, **published** or **deleted**. Above the event feed is a filter control that you can use to filter the entries displayed in the event feed. Select **> Filter** (on the left) to open the filter form and set your filtering criteria. **Status** can be set to any combination of **Submitted**, **Published** and/or **Deleted**, and you can in addition filter out everything except **Milestone** entries:



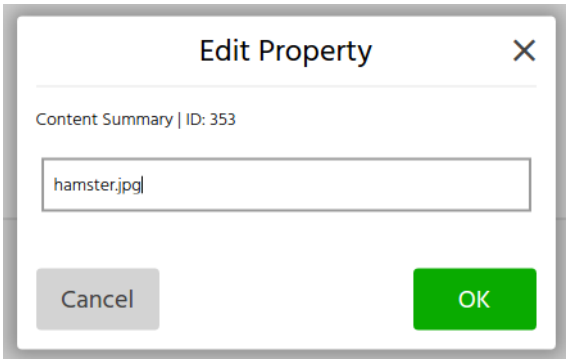
Once the filter is set up, you can switch it on and off by selecting the button (on the right).

2.2.1 Adding Images

You can post images by simply dragging them into an entry's rich text field. The source images can either be image content items dragged from Content Studio or images dragged from a web page. To upload an image from your computer or mobile device select the image button from the rich text editor toolbar.

When you drag an image in from Content Studio, one of its properties is copied along with it: this property can then be used as a title or caption for the image when it is displayed in the Live Center feed. You can see the content of this caption by holding the mouse pointer over the image (it's

displayed as a tooltip). You can also change it if you want. To do so, right-click on the image, select **Edit Properties** and then enter the required caption in the displayed dialog.



> The name of the property (**Alternative text** in the example shown above) will depend on how your Live Center installation has been set up. How the caption is used will also vary from installation to installation – it might be overlaid on the image, displayed above or below it, used as hover text or not used at all.

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 three different ways of doing this:

- Directly from feeds displayed inside the Live Center window
- By means of **embed codes**
- By means of the Live Center **bookmarklet**

2.2.2.1 Using Social Media Feeds

An event can include social feeds of various kinds. Currently Twitter, Instagram and YouTube feeds are supported, in addition to generic RSS feeds. An event's feeds (if it has any) are displayed as a column of icons down the right side of the screen. Clicking on a icon displays the content of the feed. If you see an item in the feed that you want to include in the event blog, then you can do so by clicking on it. The item is immediately copied into the current entry. You can then fill out the other fields in the entry as required and **Submit** it in the usual way.

The feeds included in an event are set up by the event designer in Content Studio – for details see section 2.5.

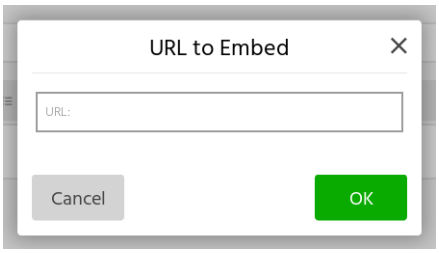
> You must have an Instagram account to be able to use Instagram feeds in Live Center. The first time you try to use an Instagram feed, you will be required to enter your Instagram user name and password. You will usually only need to do this once on each device you use – once you have authenticated yourself, Live Center stores the access token supplied by Instagram and re-uses it for all subsequent accesses.

Adding a tweet

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:

A dialog containing a single **URL** field is then displayed:



Enter the URL of the social media content you want to include in your entry. 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 Live Center'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.

Inserting a YouTube video

2.2.2.3 Using the Live Center Bookmarklet

The Live Center bookmarklet provides an alternative method of embedding content from sites that support Oembed. A bookmarklet is a special kind of browser bookmark that does more than just point

the browser to a new page. The Live Center bookmarklet is located in the Live Center main menu. To be able to use the bookmarklet you must first add it to your browser bookmarks bar. To do this:

1. Make sure your browser's bookmarks bar is displayed.
2. Select to display the main menu.
3. Drag the **Live Center** button from the menu into the bookmarks bar and drop it there.

The bookmarklet should now appear in the bookmarks bar as a button with the name Live Center.

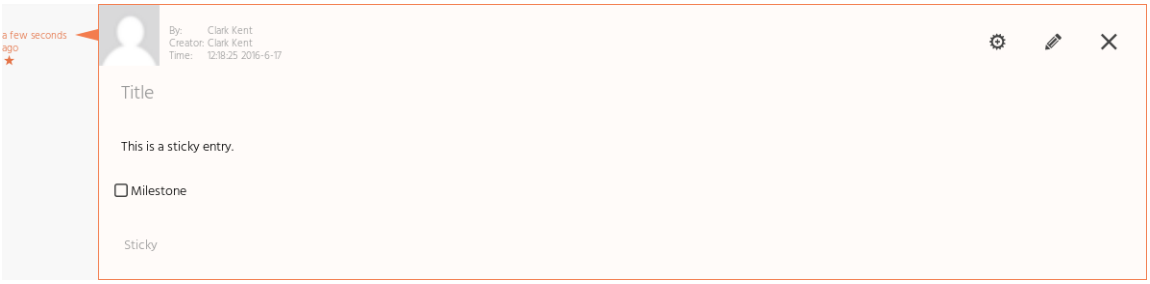
Once you have added the bookmarklet to the bookmarks bar, you can use it at any time to embed content from a site you visit into a Live Center entry. You can use it even if you are not currently logged in to Live Center: in this case, the bookmarklet will ask you to log in before embedding the selected content.

If you are currently viewing some embeddable content (a YouTube video, for example), then to embed it in an entry:

1. Select the Live Center bookmark in the bookmarks bar.
2. A new browser window opens to display Live Center.
3. If necessary, a login form is displayed: log in to Live Center.
4. A list of available events is displayed. There is a search field at the top that you can use to filter the list and find the event you are interested in.
5. Select the required event. The event is then opened and the page you started from is immediately embedded in the entry.
6. Fill out any other entry fields as you wish, and select **Submit** or **Publish** in the usual way.

2.2.3 Sticky Entries

The button displayed at the bottom of the entry editor is the "sticky" button. Clicking this button makes an entry stick to the top of the event feed. Unlike an ordinary entry, it will not be pushed down the feed as new entries are added. Sticky entries are also marked with a star and a special "sticky" tag in Live Center:



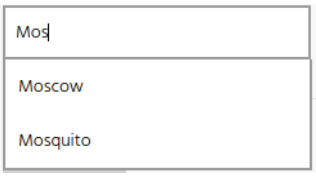
You can use sticky entries to keep an event summary or other important information visible at the top of your feed. You can create several sticky entries, in which case they will appear in reverse chronological order, just like ordinary entries.

2.2.4 Tagging Entries

Some events have **taggable** entries – that is, each entry has a tag field at the bottom that looks like this:

If the entries in an event do not have tag fields then you cannot tag them.

To add tags to an entry, click in its tag field and start typing. The tag field is a **type ahead** field: as you type it looks for tags with matching names and offers a list of suggestions:



Pick the tag you want and it is then displayed in the field like this:

You can add as many tags as you want 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 4.3.1.

2.2.4.1 Auto Tagging

Live Center 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 7.

2.2.5 Hidden Fields

Some event types may have entries that contain hidden fields. Hiding lesser-used fields can be a useful way of ensuring events do not become unwieldy and difficult to use when they contain many entries. The hidden fields in an entry are represented by a button. You can display the fields and hide them again by clicking on this button.

For information on how to hide fields in entry definitions, see section 4.2.

2.2.6 Editing Entries

You can edit existing entries as long as they have not yet been published. To edit an entry click on the edit button displayed in the top right corner of the entry. When you have finished making your

changes, tap or click on **Update** to save your changes. Note that you can edit any entries in the **submitted** state, not just ones that you created yourself.

Editing an entry

2.2.7 Posting to Twitter

The entries in some events may include a **Post to Twitter** option. If you check this option, then when the entry is published, it is also simultaneously posted to Twitter, using a predefined account.

2.3 Editorial Controls in Live Center

If you have editor access rights in Content Studio then you will also have some additional rights when using the Live Center webapp. 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 submitted entries
Click on the publish button (✓) displayed in the top right corner of any submitted entry to publish it.

Edit published entries
As an editor, you can change entries after they have been published. Click on the edit button (✎) displayed in the top right corner of an entry, make your changes and click on **Update** to save them.

Delete entries
To delete an entry, click on the delete button (×) displayed in the top right corner. Note that deleted entries are not physically deleted, simply marked as deleted. They are still displayed in the list of entries, but are grayed out.

2.4 Embedding Event Feeds

The event editor pages in Live Center include a <> button (at the top right, next to the preview button). If you click on this button you will see that it displays a text field containing a snippet of HTML code. This is an **embed code**, that a web developer can use to embed the current event on any web page, not just in Escenic publications. In order for the embed code to work, the event must be published in the usual way in an Escenic publication that is publicly accessible.

To use the embed code, simply copy it and paste it into the HTML source of the web page in which you want the event feed to appear.

2.5 Creating Events in Content Studio

To create a new event, start Content Studio and create a content item of the type **Event**. An Event content item usually has the following fields:

Title
The title of the event.

Description
A brief description of the event.

Entry type
Select the type of entry you want to use for this event. The options available in this field will depend upon the entry types defined in your publication.

Disable event
Check this option to disable or conclude an event. The event will then have the status **Concluded** in Live Center. Concluded events appear in the events list by default (although you can hide them, see section 2.1), but they are grayed out and 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.

Subscriptions
You can use this field to add social media feeds to your event. Currently you can add the following social media feeds.

- **Twitter:** Select **Twitter** as the **Source** and then enter a query string in the **Query** field. You can add multiple Twitter feeds, each with a different query string. Currently Live Center supports hash tag, user and free text search. You also need to add `twitterAPIKey` and `twitterAPISecret` as instructed in section 4.1.1
- **RSS:** Select **RSS** as the **Source** and then enter feed url in the **Query** field. You can add multiple RSS feeds, each with a different url.
- **YouTube:** Select **YouTube** as the **Source** and then enter the text that you want to search in the **Query** field. You can add multiple Youtube feeds, each with a different text. You also need to add `googleApiKey` as instructed in section 4.1.1
- **Instagram:** Select **Instagram** as the **Source** and then enter the text that you want to search in the **Query** field. You can add multiple Instagram feeds, each with a different text. You also need to add `instagramAccessToken` as instructed in section 4.1.1.

Twitter profile
The profile to use for posting entries to Twitter. The selected profile is only used if the event's entries include a **Post to Twitter** check box (see section 2.2.7).

Automatic Twitter Publishing
Use this field to add a list of **trusted Twitter accounts** to the event. A trusted Twitter account is one that you trust to provide relevant and appropriate content for the event. All tweets from a trusted account are automatically imported into the event and embedded in entries, so that nobody needs to actively monitor the account and manually drag tweets into entries. You can choose whether or not the imported tweets should be automatically published or just submitted ready for moderation by an editor.

To add a trusted account, click on + and then fill out the following fields:

Twitter Account
The name of the trusted Twitter account.

Contains Text
An optional filter string. If you enter anything in this field, then only tweets that contain the string you enter will be imported from the account.

Author
The person who will be credited as the author of the entry - a Live Center user, not the author of the tweet. The field displays a drop-down list containing the names of all registered Content Studio / Live Center users, from which you can select a name.

Auto-publish
Check this option if you want tweets from this account to be automatically published with no moderation. Otherwise the tweets will just be submitted and will need to approved by a Live Center editor.

> Live Center only imports tweets that are created while an event is live (that is, in the published state).

Video stream
You can use this field to add a live video stream to an event. The video stream is not displayed in the Live Center webapp, but can be displayed in the published live blog if your publication has been configured to support it. The field actually allows you to specify more than one video stream: this allows you to specify different versions of the video stream for display on different devices, for example. To add a video stream, click on the + button and then specify:

Mime type
The MIME type of the video stream

URL
The URL of the video stream

Once you have filled out all the fields, **Save** the content item to create the event.

> An event becomes available in the Live Center webapp as soon as it is saved – it does not need to be published. This means that you can prepare an event by adding and publishing "starter" entries in Live Center. Until you publish the event, it will have the status **Pending** in Live Center, and nothing will be visible on your web site.

3 Installation

The following preconditions must be met before you can install Live Center 2.0.0-8:

- The Content Engine and Escenic assembly tool have been installed as described in the Escenic Content Engine Installation Guide and are in working order.
- You have the required distribution file **`live-center-2.0.0-8.zip`**.

> In a multiple-server environment:
>
> - Live Center must be installed on **all** servers.
> - The Escenic event mechanism must be working correctly.

3.1 Conventions

The instructions in the following section assume that you have a standard Content Engine installation, as described in the Escenic Content Engine Installation Guide. *escenic-home* is used to refer to the **`/opt/escenic`** folder under which both the Content Engine itself and all plug-ins are installed.

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 Escenic Content Engine Installation Guide defines a set of special host names that are used throughout the manual.

Some of these names are also used here:

assembly-host
: The machine used to assemble the various Content Engine components into an enterprise archive or .EAR file.

engine-host
: The machine(s) used to host application servers and Content Engine 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 Live Center Installation

Installing Live Center involves the following steps:

1. Log in as **`escenic`** on your **assembly-host**.
2. Download the Live Center distribution. If you have a multi-host installation with shared folders as described in the Escenic Content Engine Installation Guide, then it is a good idea to download the distribution to your shared **`/mnt/download`** folder:

```
$ cd /mnt/download
$ https://user:password@maven.escenic.com/com/escenic/plugins/live-center/live-center/2.0.0-8/live-center-2.0.0-8.zip
```

Otherwise, download it to some temporary location of your choice.

3. If the folder **/opt/escenic/engine/plugins** does not already exist, create it:

```
$ mkdir /opt/escenic/engine/plugins
```

4. Unpack the Live Center distribution file:

```
$ cd /opt/escenic/engine/plugins
$ unzip /mnt/download/live-center-2.0.0-8.zip
```

This will result in the creation of an **/opt/escenic/engine/plugins/live-center** folder.

5. Log in as **escenic** on your **assembly-host**.
6. Run the **ece** script to re-assemble your Content Engine applications.

```
$ ece assemble
```

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

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

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

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

where *assembly-host-ip-address* is the host name or IP address of your **assembly-host**. Live Center should be run on all your **engine-host**s.

8. Deploy the EAR file and restart the Content Engine on each **engine-host** by entering:

```
$ ece stop
$ ece deploy
$ ece start
```

3.3 Verify The Installation

To verify the status of Live Center, open the Escenic 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.

3.4 Update The Database Schema

Live Center needs some additions to be made to the Content Engine 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-center/misc/database/mysql`**).
3. Run the scripts as follows
   - For MySql:

     ```
     $ cd /tmp/live-center/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.

4 Configuration

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

- Configure Live Center 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 Live Center) and upload it to your publication

> To support cross-publishing of live events you must add your event content type and **`entry-type`** publication resource to every publication in which you may want to publish a live event. Even publications that have no live events of their own but only cross-publish other publications' events must include a copy of the event content type and the **`entry-type`** publication resource.

4.1 Live Center Configuration

This set-up task consists of copying example configuration files from the Live Center installation into the Content Engine common configuration layer and then modifying the copied files to meet your requirements. In detail, you must:

1. Log in to your Content Engine host as the **`escenic`** user.
2. Copy all the supplied common configuration layer files as follows:

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

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 Live Center 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 Escenic Content Engine web service (**not** the Live Center web service). The default setting is **`/webservice`**, which will work if the web service is running on the same host as Live Center, 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.

editorialWebAppUrl
The URL of your Live Center webapp – **http://localhost:8080/live-center** by default. This property is used whenever Live Center users are required to authenticate themselves with an external service (currently only Instagram). It allows the browser to be automatically redirected back to Live Center from the external service's authentication interface.

editorialWebserviceUri
The URI of the Escenic Content Engine editorial web service. The URI must be specified as an absolute URI even if it is deployed on the same host as Live Center. 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 Live Center
- 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 Live Center 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 Live Center 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 Live Center 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 Live Center 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 Live Center 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 Live Center, 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 4.1.4.1.

enableTwitterStreaming
If you want to allow posts to a trusted Twitter account to be automatically republished as entries in a Live Center event, set this property to **true**. Otherwise set it to **false**. Note that this property merely enables the functionality. A number of other configuration properties must be set to specify the details of how the function is to be used. For detailed information, see section 4.1.4.3.

sseEnabled
Set this property to **true** to enable Server-sent Events (SSE) in the presentation layer. 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. If **sseEnabled** is set to **true**, then SSE can be used to keep published events up-to-date: if it is set to **false** then polling must be used. SSE offers better scalability than polling, so you are recommended to use it if possible. Note that enabling SSE here only enables it on the server side: in order for SSE to actually be used, the client code (i.e Javascript) responsible for updating published events must also support SSE. Currently SSE is supported by the demo publication delivered with Live Center and the Angular **livecenter-presentation-js** component described in section 9.1. It is not supported by the Live Entries view delivered with Widget Framework 3.6, although support is planned for a future version.

sseBaseUri
The URI of the SSE endpoint in Live Center's Presentation web service. The default setting is **/live-center-presentation-webservice/changelogSSE**, which will work if:

- The presentation web service has been deployed with the default name **live-center-presentation-webservice**. If this is not the case then you will need to rename the SSE endpoint here accordingly.
- The presentation web service is accessed via the same host as Live Center.This may not be the case if your live blogs are required to support large numbers of simultaneous visitors. SSE requires the Content Engine to hold large numbers of persistent connections open (one for each blog visitor), and Tomcat cannot handle more than 200 simultaneous client connections. Escenic therefore now ships an **SSE Proxy** with the Content Engine which can handle many thousands of simultaneous client connections on behalf of the Content Engine. If an SSE Proxy is used then the **sseBaseUri** must be an absolute URI that references the SSE Proxy host. For details of how to set up and run one or more SSE proxies for use with the Content Engine, see the SSE Proxy documentation.

4.1.2 Back-end User Configuration

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

- Images submitted by Live Center 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 Entry Configuration

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

`timeFormat`
: This property determines the content and format of the time stamp displayed with each Live Center entry. By default, the property has no value, and the time stamp shows a **relative** date in the format:

```
number [minutes|seconds|days] ago
```

If you don't want this kind of date stamp, you can switch to an **absolute** date stamp by entering a standard Java date formatting string such as:

```
timeformat=DD-MM-YYYY HH:mm:ss
```

whiteListedTags

Rich text fields are only allowed to contain a selected subset of HTML elements and attributes. Tags and attributes that do not belong to this whitelisted subset are either converted to **p** elements or removed, as appropriate. This property can be used to modify the set of whitelisted tags. The default setting is:

```
whiteListedTags={
 "em": true, \
 "br": true, \
 "u": true,  \
 "ol": true, \
 "ul": true, \
 "li": true, \
 "strong": true, \
 "b": true, \
 "i": true, \
 "p": true, \
 "h1": true, \
 "h2": true, \
 "h3": true, \
 "h4": true, \
 "h5": true, \
 "h6": true, \
 "strike": true, \
 "a": {href: true, target: true, rel: true, class: true, lang: true, title:
 true}, \
 "img": {src: true, alt: true, title: true, class: true, lang: true, translate:
 true}, \
 "blockquote": true
}
```

You can extend this set by adding more elements and/or attributes if you wish. You must **not**, however, remove any of the default elements or attributes, as they are required by the rich text editor.

latestEmbedCollapsed

When social media items such as tweets are added to an event, they are expanded by default. If you would prefer them to be collapsed when added, set this value to **true**:

```
latestEmbedCollapsed=true
```

The default value is **false**.

linkOpeningInNewTab

While adding a link using scribe editor link button, you can specify whether the link will open in new or current tab in browser. If you would prefer to open the link in new tab, set this value to **true**.

```
linkOpeningInNewTab=true
```

The default value is **false**.

`createEmbedOnPaste`
This property determines what happens when a URL is pasted into an entry. By default, the URL is inserted as a link in the usual way. If, however, you set this property to **`true`**:

```
createEmbedOnPaste=true
```

then Live Center will attempt to embed the URL target instead, if possible. This will only work for URLs belonging to domains for which an OEmbed request handler has been registered (see section 5.1). If **`createEmbedOnPaste`** is set to **`true`** but the URL target cannot be embedded, then a link is inserted instead.

`createEmbedOnCSDragDrop`
This property determines what happens when an Escenic content item is dragged from Content Studio and dropped into an entry (or copied from Content Studio and pasted into an entry). By default, dropped content items are inserted as links and copy/paste is not supported. If, however, you set this property to **`true`**:

```
createEmbedOnCSDragDrop=true
```

then Live Center will embed the content instead. You can control how different content types are embedded (what fields are displayed and how they are styled) by defining and registering your own request handlers. Live Center includes a default handler that provides a basic layout for embedded content, so you do not have to create your own request handlers. For further information, see chapter 6.

4.1.4 Twitter-related Configurations

Live Center 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 Live Center to support Twitter monitoring, you simply need to set the **`twitterAPIKey`** and **`twitterAPISecret`** properties in **`Configuration.properties`**, as described in section 4.1.1. Live Center 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 Live Center 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 4.1.4.1. You also need to select the Twitter profile to use for posting to Twitter when defining your events in Content Studio (see section 2.5) 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 4.3.4.2).

When Live Center 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 4.1.4.2.

Auto-publishing tweets
You can set up events to accept tweets from trusted Twitter accounts. Every tweet that is posted to the specified Twitter account is then automatically forwarded to Live Center, where an entry containing the tweet is added to the event. You need a Twitter profile (see section 4.1.4.1) to set up the connection that Twitter uses to send tweets from the trusted accounts to Live Center. For details of how to set up Twitter auto-publishing, see section 4.1.4.3.

4.1.4.1 Creating Twitter Profiles

A Twitter profile contains details of a Twitter account that can be used by Live Center. 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 2.2.7) option. When defining such an event in Content Studio, you can choose which profile is to be used for this purpose. (see section 2.5). If the Live Center user checks this option when submitting an entry, then the entry will be posted to the Twitter account defined in the profile you selected.

Auto-publishing stream set-up
Live Center's Twitter auto-publishing functionality requires the set-up of a Twitter API streaming connection for Twitter to be able to send tweets to Live Center. A Twitter account is required to set up this connection. You can either create a special profile for this purpose or use one of the profiles created for posting to Twitter. For further information, see section 4.1.4.3.

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 Content Studio.

   `twitterAPIKey`
   The API key of the account Live Center is to use for posting to Twitter when this profile is selected. See section 4.1.4.1.1 for details of how to obtain this value.

   `twitterAPISecret`
   The API secret of the same Twitter account. See section 4.1.4.1.1 for details of how to obtain this value.

   `twitterAccessToken`
   The access token of the same Twitter account. See section 4.1.4.1.1 for details of how to obtain this value.

   `twitterAccessTokenSecret`
   The access token secret of the same Twitter account. See section 4.1.4.1.1 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 4.2.1.5 and adding a "post to Twitter" option to the relevant entry definitions as described in section 4.3.3.1.

4.1.4.1.1 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.

4.1.4.2 Controlling the Pingback URL Format

When Live Center 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 Live Center 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. Copy **`PostToTwitterTransactionFilter.properties`** from the Live Center installation into your common configuration layer:

```
$ mkdir -p /etc/escenic/engine/webapp/live-center-editorial/com/escenic/
livecenter/social/filter
$ cp /opt/escenic/engine/plugins/live-center/misc/siteconfig/com/escenic/
livecenter/social/filter/PostToTwitterTransactionFilter.properties \
> /etc/escenic/engine/common/com/escenic/livecenter/social/filter/
PostToTwitterTransactionFilter.properties
```

2. Open the copied file 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 Live Center.

4.1.4.3 Auto-publishing Tweets

To enable auto-publishing of tweets from trusted Twitter accounts you need to:

1. Set the **`enableTwitterStreaming`** property in **`Configuration.properties`** (see section 4.1.1) to **`true`**.
2. Specify a **`twitterAPIKey`** and **`twitterAPISecret`** as described in section 4.1.1.
3. Configure a back-end user as described in section 4.1.2.
4. Create at least one Twitter profile, as described in section 4.1.4.1. This profile will be used to to set up the streaming connection Twitter needs to be able to forward tweets to Live Center. You don't need to set up a profile specifically for this purpose: if you have already created profiles for posting to Twitter, you can re-use one of them.
5. Copy **`TwitterStreamingClientService.properties`** from the Live Center installation into your common configuration layer:

   ```
   $ mkdir -p /etc/escenic/engine/common/com/escenic/livecenter/client
   $ cp /opt/escenic/engine/plugins/live-center/misc/siteconfig/com/escenic/
   livecenter/client/TwitterStreamingClientService.properties \
   > /etc/escenic/engine/common/com/escenic/livecenter/client/
   TwitterStreamingClientService.properties
   ```

6. Open the copied file for editing and set **`twitterStreamUserProfile`** to point to the Twitter profile you want to use. For example:

   ```
   twitterStreamUserProfile=/com/escenic/livecenter/client/DefaultTwitterProfile
   ```

7. Copy **`LiveEntryImportUser.properties`** from the Live Center installation into your common configuration layer:

   ```
   $ mkdir -p /etc/escenic/engine/common/com/escenic/livecenter
   $ cp /opt/escenic/engine/plugins/live-center/misc/siteconfig/com/escenic/
   livecenter/LiveEntryImportUser.properties \
   > /etc/escenic/engine/common/com/escenic/livecenter/LiveEntryImportUser.properties
   ```

   Open the copied file for editing and set the following properties:

   `userName`
   : The username of a Live Center user with the editor role. This user will be used to actually import tweets forwarded from Twitter as Live Center event entries.

   `password`
   : The specified user's password.

The basic auto-publishing functionality is now enabled. In order to enable it in specific events, however, you need to add some special fields to the event definitions in your **`content-type`** resource (see section 4.2.1.7), and configure the entry definitions in the event-type resource (see section 4.3.4.1).

4.1.5 Instagram Configuration

In order to be able to set up Instagram feeds in Live Center you must first register your Live Center 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 Live Center 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 Live Center 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/live-center`, then you must specify:

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

here, where *ip-address* is the IP address of your Live Center 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 Live Center installation, copy `InstagramAuthConfigProvider.properties` from the Live Center installation into your common configuration layer:

```
$ mkdir -p /etc/escenic/engine/common/com/escenic/livecenter/social/auth
$ cp /opt/escenic/engine/plugins/live-center/misc/siteconfig/com/escenic/livecenter/
social/auth/InstagramAuthConfigProvider.properties \
> /etc/escenic/engine/common/com/escenic/livecenter/social/auth/
InstagramAuthConfigProvider.properties
```

Open the copied file for editing and set `params.client_id` to the client ID you have obtained from Instagram.

4.1.6 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 Live Center editor, and one for controlling page length in publications.

4.1.6.1 Live Center Editor Pagination

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

1. Copy **`EntryResourceHelper.properties`** from the Live Center 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 /opt/escenic/engine/plugins/live-center/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
   ```

4.1.6.2 Presentation Layer Pagination

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

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

```
$ cp /opt/escenic/engine/plugins/live-center/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
   ```

4.1.7 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 Live Center presentation web service is deployed in a different domain from the Live Center web application then it will not be able to access any data unless you explicitly give it permission by configuring the Live Center 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 Live Center CORS filter allows you to specify what is returned in Live Center pre-flight responses.

To configure the CORS filter:

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

```
$ cp /opt/escenic/engine/plugins/live-center/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 Live Center 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 Live Center 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 Event Content Type Definition

In order to be able to use Live Center, 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 Escenic Content Engine 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 Live Center event.
- A **ui:decorator** element with the name **com.escenic.livecenter.LiveEventDecorator**

  ```
  <ui:decorator name="com.escenic.livecenter.LiveEventDecorator" />
  ```
- A **field** element with the name **livecenter**. You can control the visibility of the field with the **<ui:visibility/>** element (see section 4.3.2).
- Six parameters identifying fields in the content type that are used by Live Center for special purposes. These parameters and their associated fields are described in section 4.2.1.

4.2.1 Special Event Definition Fields

An event content type can contain a number of special fields for controlling Live Center 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.

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

An event content type must contain an entry type field, so that Content Studio 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 **livecenter**).
- Must be defined as a collection field, and configured to retrieve content from Live Center's **entry-types** web service.

```
<content-type name="event" xmlns:livecenter="http://xmlns.escenic.com/2015/live-
center">
  ...
  <panel name="main">
    ...
  <parameter name="com.escenic.live-center.content-type.field.entry-type"
 value="entryType"/>
  ...
    <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>
      <livecenter:entry-type />
      <ui:description>The entry type to use for this event</ui:description>
      <constraints>
        <required>true</required>
      </constraints>
    </field>
    ...
  </panel>
  ...
</content-type>
```

4.2.1.2 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 Live Center 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"/>
  ...
  <panel name="main">
    ...
    <field name="disabledEvent" type="boolean">
    </field>
    ...
  </panel>
  ...
</content-type>
```

An event can then be disabled by checking its **`disabledEvent`** option in Content Studio (see section 2.5).

4.2.1.3 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"/>
  ...
  <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 Live Center editor are persistent or the author field is cleared after an entry is submtted by checking/unchecking the **`clearAuthor`** option in Content Studio (see section 2.5).

4.2.1.4 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"/>
  ...
  <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"/>
          <enumeration value="instagram"/>
          <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 in Content Studio (see section 2.5).

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

In order to be able to post Live Center 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"/>
  ...
  <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 4.1.4.1. It is then possible to select the Twitter profile that will be used for posting entries from the **twitterprofile** field in Content Studio (see section 2.5).

An entry is only posted to the selected Twitter account if the Live Center 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 4.3.3.1.

4.2.1.6 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"/>
  ...
  <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 in Content Studio (see section 2.5).

4.2.1.7 The Twitter Auto-publishing Fields

You need to add a number of special fields to an entry definition in order to support the auto-publishing of tweets. The fields are all identified by parameters in the usual way. Note that for these fields to work, Twitter auto-publishing must have been correctly configured at the global level (see section 4.1.4.3 for details).

com.escenic.live-center.content-type.field.auto-post-config
This parameter must be set to point to an array of complex fields. Each member of the array defines a trusted Twitter account from which tweets will be accepted for publishing.

com.escenic.live-center.content-type.auto-post.field.account
This parameter must be set to point to a basic text field. This field is used to hold the name of a trusted Twitter account from which tweets are to be published.

com.escenic.live-center.content-type.auto-post.field.query
This parameter must be set to point to a basic text field. This field can be used to hold a search string used to filter the tweets sent from this trusted account. Only tweets that contain the specified string will be accepted for publishing. If this field is empty, then no filtering is performed.

com.escenic.live-center.content-type.auto-post.field.enabled
This parameter must be set to point to a boolean field. This field is used to determine how tweets from this account are handled. If **true**, then the entries created from accepted tweets are published immediately. If **false**, then the entries are only submitted, and must be published by an editor in Live Center.

com.escenic.live-center.content-type.auto-post.field.author
This parameter must be set to point to a collection field configured to retrieve content from Live Center's **person** web service. The field is used to select a person profile from the Content Engine's person and user archive. The selected person is then set as the author of the entries created from accepted tweets.

For example:

```
<content-type name="event">
  ...
  <parameter name="com.escenic.live-center.content-type.field.auto-post-config"
 value="twitterAutoPostConfig" />
  <parameter name="com.escenic.live-center.content-type.auto-post.field.account"
 value="twitterAccount" />
  <parameter name="com.escenic.live-center.content-type.auto-post.field.query"
 value="query" />
  <parameter name="com.escenic.live-center.content-type.auto-post.field.enabled"
 value="autoPublish" />
```

```
  <parameter name="com.escenic.live-center.content-type.auto-post.field.author"
 value="author" />
  ...
  <panel name="main">
    ...
    <field name="twitterAutoPostConfig" type="complex">
      <ui:label>Automatic Twitter Publishing</ui:label>
      <array default="0" />
      <complex>
        <field mime-type="text/plain" type="basic" name="twitterAccount">
          <ui:label>Twitter Account</ui:label>
          <constraints>
            <required>true</required>
          </constraints>
        </field>
        <field mime-type="text/plain" type="basic" name="query">
          <ui:label>Contains Text</ui:label>
        </field>
        <field linkrel="self" mime-type="text/plain" name="author" select="link"
 src="/webservice/escenic/livecenter/person" type="collection">
          <ui:label>Author</ui:label>
        </field>
        <field name="autoPublish" type="boolean">
          <ui:label>Auto-publish</ui:label>
        </field>
      </complex>
    </field>
    ...
  </panel>
  ...
</content-type>
```

4.2.2 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.live-center.content-type.field.auto-post-config"
 value="twitterAutoPostConfig" />
  <parameter name="com.escenic.live-center.content-type.auto-post.field.account"
 value="twitterAccount" />
  <parameter name="com.escenic.live-center.content-type.auto-post.field.query"
 value="query" />
```

```
 <parameter name="com.escenic.live-center.content-type.auto-post.field.enabled"
value="autoPublish" />
 <parameter name="com.escenic.live-center.content-type.auto-post.field.author"
value="author" />
 <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">
     <livecenter: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"/>
         <enumeration value="instagram"/>
         <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>
    <field name="twitterAutoPostConfig" type="complex">
      <array default="0" />
      <complex>
        <field mime-type="text/plain" type="basic" name="twitterAccount">
          <constraints>
            <required>true</required>
          </constraints>
        </field>
        <field mime-type="text/plain" type="basic" name="query"/>
        <field linkrel="self" mime-type="text/plain" name="author" select="link"
 src="/webservice/escenic/livecenter/person" type="collection"/>
        <field name="autoPublish" type="boolean"/>
      </complex>
    </field>
  </panel>
</content-type>
```

4.3 The entry-type Resource

Live Center requires an additional publication resource called **`entry-type`**. It is an XML resource that defines the field structure of different Live Center 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 Escenic Content Engine 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 Live Center entry type, not a content type.
- The **`panel`** element is required by **`content-type`** resource syntax, but has no meaning in Live Center. 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 Live Center and are ignored if present.
- The following field types are not supported by Live Center and will be ignored if present:

  `collection`
  `complex`
  `schedule`
- Some **`field`** elements may contain special Live Center elements belonging to the **`http://xmlns.escenic.com/2015/live-center`** namespace. For details see section 4.3.3.
- The **`entry-type`** resource may also contain **`parameter`** elements for special purposes. For details see section 4.3.4.

4.3.1 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 Escenic and tag scheme definition, see Manage Tag Structures.

4.3.2 Controlling Field Visibility

You can control the visibility of entry fields in the Live Center editor in the same way as you control the visibility of content item fields in Content Studio, by adding **ui:visibility** elements (see http://docservices.dev.escenic.com/ece-resource-ref/6.0/ih_visibility.html) to field definitions. The **ui:visibility** values are used by Live Center as described in the following table. Visibility may be different in the event feed than it is in an entry editor.

Visibility setting	Entry editor	Event feed
hidden	Hidden	Hidden
expert	Initially hidden, click ⚙ to display	Initially hidden if not set, click ⚙ to display. Visible if set
advanced	Initially hidden, click ⚙ to display	Initially hidden, click ⚙ to display
beginner	Always visible	Always visible

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

4.3.3 Special Live Center Elements

Some **field** elements may contain special Live Center elements belonging to the **http://xmlns.escenic.com/2015/live-center** namespace (usually assigned the prefix **livecenter**). The elements available in this namespace (and what they are used for) are described below:

4.3.3.1 The twitterPost Element

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

This element is used to identify a Boolean field as a "share on Twitter" field. If the Live Center 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 4.3.3.3) element.

Note that posting to Twitter will only work if Live Center has been configured with Twitter profiles (see section 4.1.4.1) and if the event containing the entry has been configured to use one of those profiles (see section 4.2.1.5).

You should always include an **entryShare** element with a **twitterPost** element to enhance the usability of the sharing option. For details see section 4.3.3.2.

4.3.3.2 The entryShare Element

```
<field type="boolean" name="twitter-post">
  <livecenter:twitterPost responseFieldName="twitter-response"/>
  <livecenter:entryShare />
</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, Live Center only supports the creation of sharing options for Twitter, via the twitterPost (section 4.3.3.1) element.)

4.3.3.3 The twitterPostResponse Element

```
<field mime-type="text/plain" type="basic" name="twitter-reponse">
  <livecenter: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 4.3.3.1) 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.

4.3.3.4 The milestone Element

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

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

4.3.3.5 The orientation Element

```
<field type="boolean" name="milestone">
  <ui:label>Milestone</ui:label>
  <ui:description>Milestone</ui:description>
  <ui:visibility>expert</ui:visibility>
  <livecenter:milestone/>
  <ui:orientation>right</ui:orientation>
</field>
```

This element (which is in the **http://xmlns.escenic.com/2008/interface-hints namespace**, not **http://xmlns.escenic.com/2015/live-center**) can be used to control the alignment of fields when displayed in the Live Center user interface. It can contain one of two values, **left** (the default alignment) or **right**.

4.3.4 entry-type Parameters

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

4.3.4.1 com.escenic.live-center.content-type.field.auto-publish-entry

Some events may be configured to support auto-publishing of tweets imported from specific trusted Twitter accounts. This parameter is used to identify the entry field in which imported tweets are to be embedded. 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). Imported tweets will be written to this field. The chosen field must be a rich text field (that is, it must have **type="basic"** and **mime-type="application/xhtml+xml"**). For example:

```
<content-type name="generic">
  ...
  <parameter name="com.escenic.live-center.content-type.field.auto-publish-entry"
 value="xhtml"/>
  ...
  <panel name="default">
    ...
    <field mime-type="application/xhtml+xml" type="basic" name="xhtml">
      <livecenter:body enabled="true"/>
      <ui:label>Body</ui:label>
      <ui:description>A sample xhtml field</ui:description>
      <ui:visibility>beginner</ui:visibility>
    </field>
    ...
  </panel>
  ...
</content-type>
```

4.3.4.2 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 Live Center 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>
        <constraints>
          <maxchars>200</maxchars>
        </constraints>
      </field>
    ...
  </panel>
  ...
</content-type>
```

4.3.5 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.auto-publish-entry"
 value="xhtml"/>
    <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: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:description>A sample xhtml field</ui:description>
      </field>

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

      <field type="enumeration" name="singleChoiceEnumeration">
        <ui:label>Happening</ui:label>
        <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>
```

4.4 Default Image Caption Configuration

Images included in event entries have captions. In Live Center, the caption is displayed as a tool tip if the user holds the mouse pointer over the image. The caption can be set by the user in Live Center (see section 2.2.1). If no caption is explicitly specified, then the image content item's title field (that is, the field pointed to by the **`ui:title-field`** element) is used as a default caption.

If you wish, you can configure the image content type so that Live Center uses the content of a different field as the default caption. You do this by adding a **`com.escenic.live-center.content-type.title`** parameter to the image's content type definition in the **`content-type`** resource. In the following case, for example:

```
<content-type name="image">
  <ui:title-field>name</ui:title-field>
  <panel name="main">
    <field mime-type="text/plain" type="basic" name="name">
      <ui:label>Name</ui:label>
      <ui:description>The name of the image</ui:description>
      <constraints>
        <required>true</required>
      </constraints>
    </field>
    ...
    <field mime-type="text/plain" type="basic" name="alttext">
      <ui:label>Alternative text</ui:label>
    </field>
    ...
</content-type>
```

the content of the **`name`** field is used as the default caption. You can force Live Center to use the **`alttext`** field as the default caption instead by adding a **`com.escenic.live-center.content-type.title`** parameter as follows:

```
<content-type name="image">
  <parameter name="com.escenic.live-center.content-type.title" value="alttext"/>
  <ui:title-field>name</ui:title-field>
  <panel name="main">
    <field mime-type="text/plain" type="basic" name="name">
      <ui:label>Name</ui:label>
      <ui:description>The name of the image</ui:description>
```

```
      <constraints>
        <required>true</required>
      </constraints>
    </field>
    ...
    <field mime-type="text/plain" type="basic" name="alttext">
      <ui:label>Alternative text</ui:label>
    </field>
    ...
</content-type>
```

4.5 Cache Configuration

Caching is one of the most important factors governing Live Center performance. You are strongly recommended to use an external cache such as Varnish in front of your Content Engine installation when using Live Center 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 Engine **`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 Engine object caches, see Tuning The Object Caches. For general information about installing and configuring **`memcached`**, see Distributed Memory Cache.

To configure Live Center 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 Live Center 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 Live Center 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 Live Center 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 Live Center installation into your **Live Center webapp** configuration layer (not the common configuration layer):

   ```
   $ cp /opt/escenic/engine/plugins/live-center/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 Live Center installation into your **Live Center webapp** configuration layer (not the common configuration layer):

   ```
   $ cp /opt/escenic/engine/plugins/live-center/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 Binary Upload Configuration

Live Center supports the upload of images from the desktop and/or mobile devices. In order for this functionality to work, the following configuration is required:

1. Create a **`com/escenic/livecenter/webapp/helpers/builder/`** folder in your **Live Center webapp** configuration layer (not the common configuration layer):

   ```
   $ mkdir -p /etc/escenic/engine/webapp/live-center-editorial/com/escenic/
   livecenter/webapp/helpers/builder/
   ```

2. Copy **`EscenicContentResourceEntityBuilder.properties`** into the new folder:

   ```
   $ cp /opt/escenic/engine/plugins/live-center/misc/siteconfig/webapp/com/escenic/
   livecenter/webapp/helpers/builder/EscenicContentResourceEntityBuilder.properties \
        > /etc/escenic/engine/webapp/live-center-editorial/com/escenic/livecenter/
   webapp/helpers/builder/
   ```

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

   `binary.contentType`
   The content type to be used for the binary object type you want to be able to upload (e.g. images). Currently images are the only binary object type supported. This property has global effect – it applies to all publications. If, for example, your image content type is called **`picture`**, then you would specify:

   ```
   binary.contentType=picture
   ```

   `binary.homeSectionId`
   The section id of the section to which uploaded binary content items will be added. If you do not specify this property, then images are uploaded to the publication's default section. This property has global effect – it applies to all publications.

   If you want binary upload to be handled differently for different publications, then you can add publication-specific properties to the file and use them instead of or alongside the global properties. These properties have names of the form:

   `publicationWiseBinary.`*publication-name*`.contentType`**

   where *publication-name* is the name of the required target publication.

   If you specify both global and publication-specific properties, then the publication-specific properties override the global settings where appropriate.

4.7 Third-Party Authentication

Escenic Content Engine 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 Engine is configured to use Google Apps for user authentication, then Google Apps will also be used by Live Center for authenticating users. This means that users in organizations that use Google Apps as their standard office suite can also log into Content Engine and Live Center using their Google log-ins. Live Center does **not**, however, currently support Microsoft Active Directory or Facebook-based authentication.

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

5 Embedding External Content

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

- YouTube
- Vimeo
- Instagram
- Vine
- Twitter

This functionality is based on oEmbed, an open standard for displaying embedded content. In addition to embedding support for the above services, Live Center 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 5.1). If you do this, then Content Engine 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 5.2). You can then provide your own HTML template and ensure that the embedded content looks exactly how you want it to.

Live Center 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 5.3 and section 5.4 for details).

5.1 Creating an oEmbed Request Handler

To create your own site-specific oEmbed request handler:

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

   ```
   $ cp /opt/escenic/engine/plugins/live-center/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
```

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

`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 5.5.

5.2 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 Live Center installation into your **webapp** configuration layer (not the common configuration layer), and rename it appropriately. For Flickr, for example, you might enter:

```
$ cp /opt/escenic/engine/plugins/live-center/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 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).

`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 5.5.

To compile your custom oEmbed request handler you need to ensure that the **`live-center-core-2.0.0-8.jar`** and **`live-center-api-2.0.0-8.jar`** libraries are in your classpath.

5.3 Creating an Open Graph Request Handler

To create a site-specific Open Graph request handler:

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

```
$ cp /opt/escenic/engine/plugins/live-center/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).

`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 5.5.

5.4 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 Live Center installation into your **webapp** configuration layer (not the common configuration layer), and rename it appropriately. For Flickr, for example, you might enter:

```
$ cp /opt/escenic/engine/plugins/live-center/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 5.3 for a description of the properties.
6. Register your request handler as described in section 5.5.

To compile your custom Open Graph request handler you need to ensure that the **`live-center-core-2.0.0-8.jar`** and **`live-center-api-2.0.0-8.jar`** libraries are in your classpath.

5.5 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 Embedding Escenic Content Items

Live Center supports the embedding of Escenic content in event entries. Content items can be dragged from Content Studio and dropped in an entry, or copied from Content Studio and pasted into an entry. The dropped content items do not have to belong to the same publication as the Live Center live blog – you can embed content from any publication in your installation. In order for embedding to work, the entry configuration property **`createEmbedOnCSDragDrop`** must be set to **`true`** (see section 4.1.3). If **`createEmbedOnCSDragDrop`** is set to **`false`** then items dragged from Content Studio are inserted as links rather than being embedded.

If **`createEmbedOnCSDragDrop`** is set to **`true`**, then a simple form of embedding is immediately enabled: any content item that is dragged into a Live Center entry appears as embedded content in the entry both in the Live Center editor and in the published blog once the entry is published.

6.1 Overriding the Default Embed Layout

> The embedding mechanism described in this section replaces an older, deprecated mechanism used in Live Center version 1.3. Since it is deprecated, documentation of this old mechanism has been removed. If for any reason you need access to the old documentation, see here.

You can override the default embed layout by modifying the following items as described below:

- Two request handler configurations – an editorial request handler configuration and a presentation request handler configuration.
- The JSP template to which embed requests are forwarded

Create your own editorial embed request handler as follows:

1. Copy **`DefaultECERequestHandler.properties`** from the Live Center installation to the correct location in your common configuration layer:

   ```
   $ mkdir -p /etc/escenic/engine/common/com/escenic/livecenter/service/oembed/ece
   $ cp /opt/escenic/engine/plugins/live-center/misc/siteconfig/com/escenic/
   livecenter/service/oembed/ece/DefaultECERequestHandler.properties \
   > /etc/escenic/engine/common/com/escenic/livecenter/service/oembed/ece/
   ```

2. Open **`DefaultECERequestHandler.properties`** file for editing and replace the default **`blockCodeTemplate`** setting:

   ```
   <iframe src="{{OEMBED_TEMPLATE_URL}}" width="480" height="300" frameBorder="0"
   scrolling="no" allowtransparency="true" allowfullscreen="true" /></iframe>
   ```

   with your required template code.

Then repeat the process to create your own presentation embed request handler:

1. Copy **`DefaultPresentationECERequestHandler.properties`** from the Live Center installation to the correct location in your **webapp** configuration layer (not the common configuration layer):

   ```
   $ mkdir -p /etc/escenic/engine/webapp/live-center-presentation-webservice/com/
   escenic/livecenter/presentation/service/oembed/
   ```

```
$ cp /opt/escenic/engine/plugins/live-center/misc/siteconfig/
webapp/com/escenic/livecenter/presentation/service/oembed/
DefaultPresentationECERequestHandler.properties \
> /etc/escenic/engine/webapp/live-center-presentation-webservice/com/escenic/
livecenter/presentation/service/oembed/
```

2. Open **DefaultPresentationECERequestHandler.properties** file for editing and replace the default **blockCodeTemplate** setting:

```
<iframe src="{{OEMBED_TEMPLATE_URL}}" width="480" height="300" frameBorder="0"
scrolling="no" allowtransparency="true" allowfullscreen="true" /></iframe>
```

   with your required template code.

The placeholder **{{OEMBED_TEMPLATE_URL}}** in both templates is replaced by the URL of the content item in its embedded form. This "embedded" URL is formed by appending the string **/oembed** to the end of the content item's ordinary URL. For example, a content item with the standard URL **http://myorg.com/dailynews/incoming/article123.ece** will have the embedded URL **http://myorg.com/dailynews/incoming/article123.ece/oembed**.

Any request with such an "embedded" URL is forwarded to the JSP template **publication/template/oembed/common.jsp**. You can therefore determine the layout of all the content items embedded in your live blog by editing this JSP template in the usual way. The template included in the Live Center demo application looks like this:

```
<%@ page language="java" pageEncoding="UTF-8" contentType="text/html; charset=UTF-8"
 %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://www.escenic.com/taglib/escenic-article" prefix="article"%>

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="no" lang="no">
  <head>
      <meta http-equiv="Content-type" content="text/html; charset=iso-8859-1"/>
      <title>${publication.name} :: ${article.fields.title}</title>

  </head>
  <body style="margin: 0; padding: 0;">
    <jsp:include page="${article.articleTypeName}.jsp"/>

  </body>
</html>
```

7 Auto-Tagging Entries

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

- The types of Escenic 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 Escenic mechanism for extending Content Engine functionality. A number of ready-made transaction filters to support the tagging criteria listed above are included with Live Center, 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 Live Center transaction filters, see chapter 8.

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 **`/opt/escenic/engine/plugins/live-center/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 Live Center to use the auto-tagging transaction filters you are interested in (see section 7.1).

7.1 Configuring the Auto-Tagging Transaction Filters

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

`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 7.1.1 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 7.1.2.

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.

7.1.1 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:livecenter="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 from Content Studio (see The Manage Tags Dialog).

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:livecenter="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>
```

7.1.2 Adding New Social Media Services to EmbedAutoLabelingTransactionFilter

If you have extended Live Center 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 from Content Studio (see The Manage Tags Dialog).

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>
```

8 Live Center 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 Engine mechanism for modifying and extending Content Engine behavior, and is described in detail in the Escenic Content Engine Advanced Developer Guide.

The Live Center 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 Live Center with an external system in some way: for example to copy all Live Center events to an external system as they are created.

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

8.1 Making A Transaction Filter

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

To make a Live Center 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 Engine 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 Engine's classpath

8.2 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 Engine 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
```

9 Publishing Events

Live Center events are different from ordinary Content Engine 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 Engine presentation layer. The basis of the Live Center 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 10.2.

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 Live Center 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 Escenic Maven repository. By using it in your event page JSP templates, you can again hide the details of using the Live Center presentation web service. For further information on how to use the library, see section 9.1.

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 10.2 for further information.

Once an event feed is successfully published in an Escenic publication, you can republish it on **any** web page (not just Escenic sites) by means of embedding. For details, see section 2.4.

9.1 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 Live Center 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 Live Center 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 Live Center 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-center</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.

9.1.1 Configuration

`livecenter-presentation-js` has a number of configuration options that you can set in your AngularJS app. The example app in section 9.1 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'}
```

10 Using The Live Center Web Services

Live Center 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 Live Center webapp, or for integrating Live Center with third-party systems. Access to this web service requires authentication as for the Content Engine 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 Engine and most other Content Engine 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 Engine. The presentation web service is effectively a subset of the editorial web service. No authentication is required to access this web service.

10.1 The Editorial Web Service

The Live Center editorial web service is very similar to the Content Engine 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 Engine web service, then you will find the Live Center web service easy to use. If you haven't used the Content Engine web service before, then you should probably start by reading at least the introduction of the Escenic Content Engine 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 Engine web service, the Live Center 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 Engine presentation layer or the Content Engine 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 Live Center 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
```

Live Center 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",
    "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",
```

```
    "payload": [
        {
            "name": "title",
            "value": "Liverpool - Manchester United"
        },
        {
            "name": "description",
            "value": "Match: 23. January 2015"
        }
    ],
    "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",
      "payload":[
        {
          "name": "basic",
          "value": "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 Live Center 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

10.1.1 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",
  "payload":[
    {
      "name": "basic",
      "value": "Test entry"
    }
  ],
  "publishDate": "2015-02-13T12:20:24.000+0000",
  "self": "http://host-ip-address/live-center-editorial/entry/283",
  "state": "published",
  "tags":[]
}
```

10.1.2 Changing an Entry

To change an Entry you retrieve the resource as described in section 10.1.1, 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 Live Center 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**. Live Center uses the **If-Match** header is used in exactly the same as the Content Engine. For a detailed explanation of what it is used for and how it works, see http://docs.escenic.com/ece-integration-guide/6.0/optimistic_concurrency.html.

10.1.3 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 **payload** field:

```
{
  "payload":[
    {
      "name": "basic",
      "value": "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:
>
> ```
> {
>   "payload":[
>     {
> ```

```
        "name": "xhtml",
        "value": "<p class=\"myclass\">Hello world</p>"
      }
    ]
  }
```

Embedding Foreign Content

Live Center 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 payload, 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**, **Vine** or **Instagram**. This identifies the source of the foreign content. No other sources of foreign content are currently supported.

The following example shows a payload containing an embedded YouTube video:

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

10.1.4 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.

10.2 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 Live Center presentation web service has no global entry point or "start" URL. You will usually access it by obtaining a URL from the Content Engine'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 Engine
- *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 10.2.1.

> Entries are returned 10 at a time by default. You can, however, change this page length — see section 4.1.6.2 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 10.2.2.2).

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 10.2.2.2 for details.

sseURI
The presentation web service URL of this event's server-sent events (SSE) connection. See section 10.2.2.1 for details.

10.2.1 Entry Fields

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

creator
The name and Escenic web service URL of the Content Engine user who created the entry.

author
The name and Escenic web service URL of the Content Engine 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 Live Center.

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 10.2, 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 10.2 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 Escenic web service URL.

`sticky`
Whether or not the entry is sticky.

10.2.2 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 10.2.2.1.

Polling the event change log
This is the original method used to keep Live Center 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 10.2.2.2.

10.2.2.1 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 Engine 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 Engine to hold large numbers of persistent connections open. This can quickly become a problem, since Tomcat cannot handle more than 200 simultaneous client connections. Escenic therefore now ships an **SSE Proxy** with the Content Engine. An SSE Proxy can handle up to 28,000 simultaneous client connections on behalf of the Content Engine. It is possible to run many such proxies in parallel, each of which only require one connection to the Content Engine, 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 Engine, see the SSE Proxy documentation.

10.2.2.2 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 Engine web service's change log. If you want a more detailed description of how it works, see Change Logs.

> Currently, the Live Center 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.

10.2.3 Retrieving Embedded Content

The Live Center 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.

10.2.4 Retrieving Selected Entries

The Live Center 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 Engine
- *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 10.2).

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.