Live Center

Plug-in Guide

1.2.0.175186

escenic

A CCI COMPANY

escenic
A CCI COMPANY

Table of Contents

1 Introduction

Live Center is a liveblogging plug-in for the Escenic Content Engine. It extends the Content Engine's with a special content type for holding live blogs plus the Live Center webapp. 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

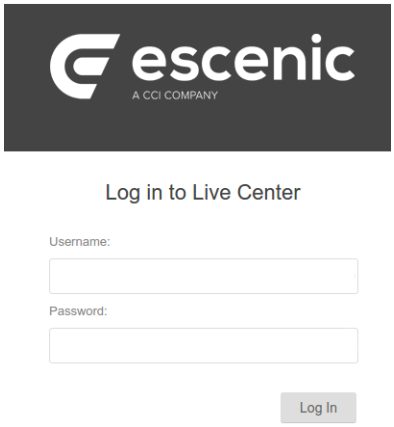
Some problems have been reported when using the current version of the Live Center editor in both Firefox and Safari. For the moment, therefore, we recommend the use of Chrome with the Live Center editor. Note that this limitation only applies to editing: published Live Center blogs can of course be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

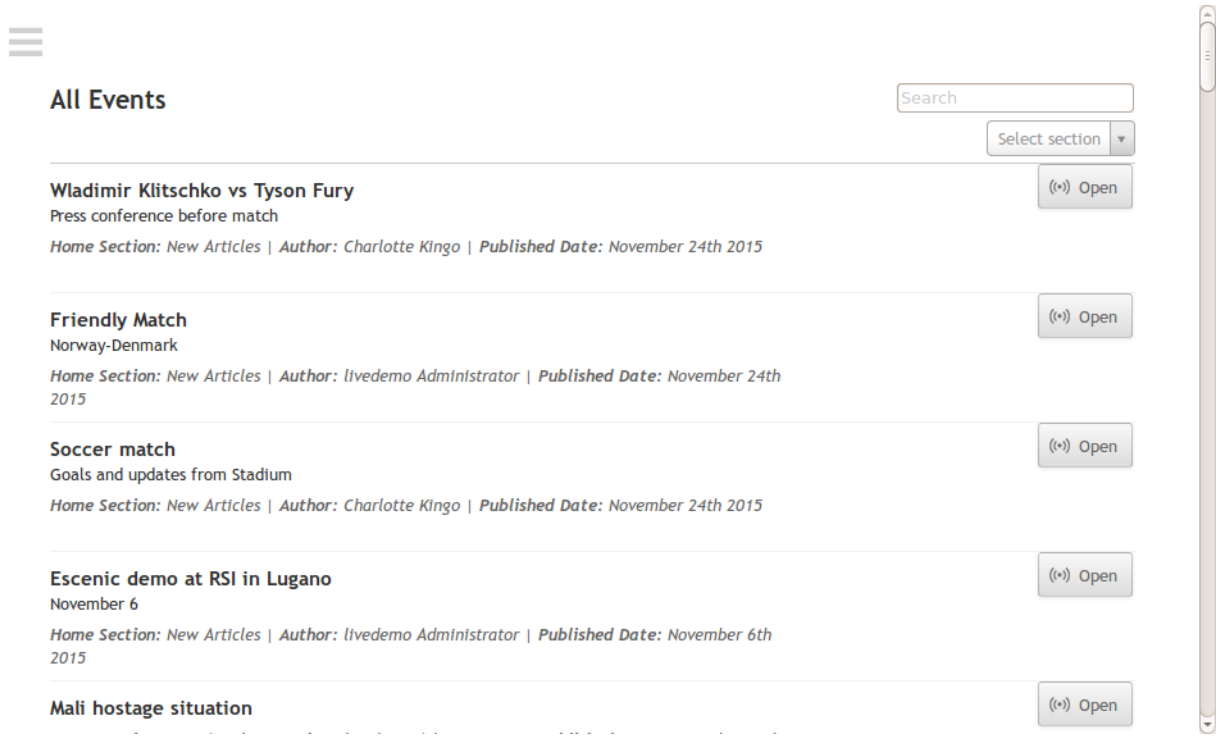
```
http://your-server/live-center/
```

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:



At the top of the list on the right hand side are a search field and a section selector that you can use to filter the list. The section selector displays only events that belong to a selected section.

Select the event you want, and start blogging.

Using the Live Center webapp

For information about writing and editing blogs in Live Center, see section 2.1.

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 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** as in the example above. 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.

All entries have an **Author** field at the top. It 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.

When you click on **Submit**, the entry you have been working on is moved to the top of the **event feed** displayed below the **Submit** button. It is highlighted in yellow 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**. At the top of the event feed, on the left, is a filter button that you can use to control which entries are displayed. If, for example, you want to see just the published entries, then click on this button and select **Only Published**.

2.1.1 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. You cannot, however, drag images from your desktop into an entry (that is, you cannot upload them from your computer or mobile device).

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.

2.1.2 Adding Social Content

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

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

2.1.2.1 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.

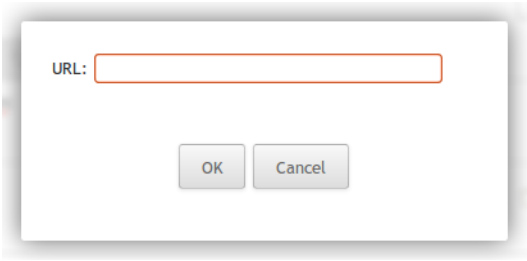
The feeds included in an event are set up by the event designer in Content Studio – for details see section 2.3.

Adding a tweet

2.1.2.2 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

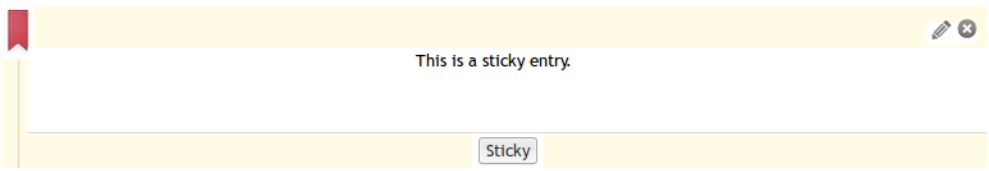
However, many other sites are also supported via an open standard for embedding called Oembed. There is a list of Oembed providers here. You should be able to embed content from any of the sites in this list.

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

Inserting a YouTube video

2.1.3 Sticky Entries

Checking an entry's **Sticky** option makes it 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 red icon in Live Center:



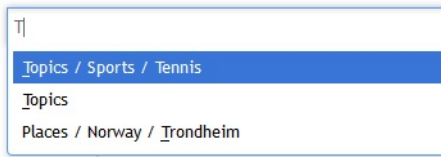
You can use sticky entries to keep an event summary or other important information visible at the top of your feed. You can create several sticky entries, in which case they will appear in reverse chronological order, just like ordinary entries.

2.1.4 Tagging Entries

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

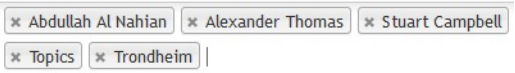
If the entries in an event do not have tag fields then you cannot tag them.

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.

Tags are displayed at the bottom of published event entries, and look like this:

For information about how you make entries taggable, see section 4.3.1.

2.1.4.1 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 6.

2.1.5 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 hidden fields by clicking on this button, and hide them again by clicking on the ▲ button.

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

2.1.6 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 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 2.1, 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.

2.3 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.

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.

Disable event
Check this option to disable an event. The event will then not appear in the Live Center **Live Events** list. It will still appear in the **All Events** list, but will be read-only.

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 alos 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

  Note that, Now in new api of Instagram, Every new app created on the Instagram Platform starts in Sandbox mode (see details here - https://www.instagram.com/developer/sandbox/) which has certain restrictions on api requests. Your app needs to be "Live" to get the proper search results (see more : https://www.instagram.com/developer/review).

Video stream
To be supplied.

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. Nothing will be visible on your web site until you publish the event in Content Studio.

3 Installation

The following preconditions must be met before you can install Live Center 1.2.0.175186:

- 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 plug-in distribution file **`live-center-dist-1.2.0.175186.zip`**.

> In a multiple-server environment:
>
> - The Live Center plug-in 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
$ wget http://user:password@technet.escenic.com/downloads/release/5.7.58.172045/
live-center-dist-1.2.0.175186.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-dist-1.2.0.175186.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 the Live Center plug-in, 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**. The Live Center plug-in 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 the Live Center plug-in, 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

The Live Center plug-in 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 the Live Center plug-in 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 the Live Center plug-in) and upload it to your publication

4.1 Live Center Configuration

This set-up task consists of copying example configuration files from the plug-in 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.

`twitterAPIKey`
: The API key Live Center is to use for connecting to Twitter. 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.

instagramAccessToken
The client id to use when connecting to Instagram. How to generate an access_token can be found in Instagram authentication page (https://www.instagram.com/developer/authentication/).

4.1.2 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`**.

4.1.3 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.

4.1.3.1 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 plug-in 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
   ```

4.1.3.2 Presentation Layer Pagination

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

1. Copy **`PublishedEntryResourceHelper.properties`** from the plug-in 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
   ```

4.1.4 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 plug-in 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 the Live Center plug-in, 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).
- Five parameters identifying fields in the content type that are used by Live Center for special purposes. These must have the names **com.escenic.live-center.content-type.field.entry-type**, **com.escenic.live-center.content-type.field.visibility-in-editor**, **com.escenic.live-center.content-type.field.clear-author-field-on-submit**, **com.escenic.live-center.content-type.field.subscription** and **com.escenic.live-center.content-type.field.video** and their **value** attributes must reference the appropriate content type fields. For example:

  ```
  <parameter name="com.escenic.live-center.content-type.field.entry-type"
   value="entryType"/>
  <parameter name="com.escenic.live-center.content-type.field.visibility-in-editor"
   value="disableEvent"/>
  <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.video" value="video"/>
  ```

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

```
<content-type name="event" >
  <ui:title-field>title</ui:title-field>
  <ui:decorator name="com.escenic.livecenter.LiveEventDecorator" />
  <parameter name="com.escenic.live-center.content-type" value="true"/>
  <parameter name="com.escenic.live-center.content-type.field.entry-type"
 value="entryType"/>
  <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.visibility-in-editor"
 value="disableEvent"/>
  <parameter name="com.escenic.live-center.content-type.field.subscription"
 value="subscriptions"/>
  <parameter name="com.escenic.live-center.content-type.field.video" value="video"/>
  <parameter name="com.escenic.index.summary.fields" value="description"/>
  <panel name="main">
```

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

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

    <field name="entryType" type="enumeration">
      <enumeration value="football"/>
      <enumeration value="simple"/>
      ...
     <constraints>
        <required>true</required>
      </constraints>
    </field>

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

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

    <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:label>Live Center</ui:label>
      <ui:hidden/>
    </field>
  </panel>
```

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

The content type's visibility in editor field (that is, the field pointed to by the **com.escenic.live-center.content-type.field.visibility-in-editor** parameter – **disableEvent** in the example shown above) should be defined as a boolean field. This field can be used to determine whether or not live events are visible in the Live Center editor.

The content type's clear author on submit (that is, the field pointed to by the **com.escenic.live-center.content-type.field.clear-author-field-on-submit** parameter – **clearAuthor** in the example shown above) should be defined as a boolean field. This field can be used to determine whether or not the author field will be cleared after submitting or publishing an entry in Live Center editor.

The content type's entry type field (that is, the field pointed to by the **com.escenic.live-center.content-type.field.entry-type** parameter – **entryType** in the example shown above) should defined as an enumeration field, and the enumeration values should be the names of entry types defined in an **entry-type** resource (see section 4.3).

4.3 The entry-type Resource

The Live Center plug-in 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

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/5.7/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 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"
    >
  <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>

    <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>
      </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 2.1.1). 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.

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**.

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 plug-in 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 plug-in 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 Upload and Create article via LiveCenter webservice

1. In order to upload and create binary article from external application(e.g. mobile/desktop) you need to do the following configuration:

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

```
$ 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/
```

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

binary.articleState
State of the uploaded article (e.g. **draft/published**)

`binary.contentType`
Content type of the uploaded article (e.g. **`image`**)

`binary.homeSectionId`
Section id to which the article will be uploaded.

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 plug-in 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 plug-in 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}
```

For Flickr, for example, you might specify a template like this:

```
blockCodeTemplate=<blockquote class="flickr-image" lang="en">TBS</
blockquote>
```

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.

5.3 Creating an Open Graph Request Handler

To create a site-specific Open Graph request handler:

1. Copy **OGPRequestHandler.properties** from the plug-in 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. To include Open Graph property values in your template, enclose the property name in double braces thus:

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

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 plug-in 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.

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 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 7.

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

- Create a tag structure as described here. The tag structure must have the scheme (i.e name or identifier) **`livelabels@escenic.com,2015`**. Add tags to the structure by importing the supplied tag syndication file **`tags.xml`** which you will find in **`/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 6.1).

6.1 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 6.1.1 for a description of how to specify the boolean fields to be watched.

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

To enable the transaction filters:

1. Open **/etc/escenic/engine/common/com/escenic/livecenter/LiveEntryDao.properties** for editing.
2. Add entries for the transaction filters you want to enable as follows:

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

3. Save your changes.

6.1.1 Enabling Use of BooleanAutoLabelingTransactionFilter

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

```
<auto-label labelName="tag-name" xmlns: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 "milestone" 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="milestone">
  <ui:label>Milestone</ui:label>
  <auto-label labelName="milestone" xmlns:livecenter="http://xmlns.escenic.com/2015/
live-center"/>
</field>
```

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

```
<tag term="milestone">
  <label>Milestone</label>
</tag>
```

6.1.2 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>
```

7 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

7.1 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

7.2 Using a Transaction Filter

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

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

The transaction filters executed by the Content 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
```

8 Using The Live Center Web Services

The Live Center plug-in 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.

8.1 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

8.1.1 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":[]
}
```

8.1.2 Changing an Entry

To change an Entry you retrieve the resource as described in section 8.1.1, 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/5.7/optimistic_concurrency.html.

8.1.3 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 2.1.2.2). 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>"
  }
]
```

8.1.4 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.

8.2 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"
}
```

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 8.2.1.

> Entries are returned 10 at a time by default. You can, however, change this page length — see section 4.1.3.2 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 8.2.2).

`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 8.2.2 for details).

8.2.1 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 8.2, 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 8.2 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.

8.2.2 Using The Event Change Log

The change log is what you use to keep your event page up to date once you have retrieved the initial entries to be display. Basically, once you have constructed and displayed the event page, you should 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 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.

8.2.3 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.