# Table of Contents

Introduction to App Studio Markup ................................................................. 1
  Connect to Fusion .................................................................................. 2
    The search:platform tag .................................................................. 2
    The search:query tag ..................................................................... 2
    The search:response tag .................................................................. 3
  Control the page layout ........................................................................ 4
Anatomy of a View .................................................................................... 5
  Include a partial file ............................................................................ 6
  Sidebar with facet list ........................................................................ 7
  Search statistics and results list ............................................................. 8
  Individual results ................................................................................ 9
  Null search results ............................................................................ 10
Create a New Web App ............................................................................... 11
  Prerequisites ...................................................................................... 12
  How to create a new Web app ............................................................... 13
  What’s next ....................................................................................... 19
Code Editor ............................................................................................... 20
  Opening the code editor ................................................................... 21
  The toolbar ....................................................................................... 22
  Saving your changes ........................................................................ 23
Download a Project .................................................................................. 24
  Downloading a project from Fusion .................................................... 25
  Contents of the project directory ......................................................... 27
  The app-studio/app-studio.bat script .................................................. 28
  Deploying a project .......................................................................... 29
Files and Directories ................................................................................ 30
  Directory structure in the code editor ................................................ 31
  Directory structure in a downloaded project ....................................... 32
  The app-studio/app-studio.bat script .................................................. 33
  Configuration files ............................................................................. 34
    config/twigkit.conf ...................................................................... 34
    config/cors.conf ......................................................................... 35
    Hierarchical configuration .............................................................. 36
    Configuration locations .................................................................. 37
    Adding special characters to key names in configuration files ......... 37
Search Queries ........................................................................................ 38
  Usage example .................................................................................. 39
Query Parameters ..................................................................................... 40
Filters ...................................................................................................... 43
  Filter formats .................................................................................... 44
  Short format ..................................................................................... 44
  Long format ...................................................................................... 44
Parameters .................................................................................................................. 81
Example request ......................................................................................................... 81
As XML .......................................................................................................................... 82
As JSON ......................................................................................................................... 82
Data operations for bookmarks ..................................................................................... 83
  Resource URI ................................................................................................................. 83
  GET bookmark/{id} ....................................................................................................... 83
  POST bookmark ............................................................................................................ 84
  DELETE bookmark/{id} ............................................................................................... 85
Search for Annotated Documents .................................................................................... 85
  Method and resource URI ............................................................................................ 86
  Parameters .................................................................................................................... 86
  Example request .......................................................................................................... 86
Protect Against CSRF Attacks ........................................................................................ 87
Localise an App ............................................................................................................... 88
  Localisation service .................................................................................................. 89
  Loading translations ................................................................................................. 89
  Enabling translations ............................................................................................... 89
  Localisation Keys .................................................................................................... 91
Demo Deployment ........................................................................................................... 94
  Startup in demo mode ............................................................................................... 95
  Shutdown in demo mode ........................................................................................... 96
Introduction to App Studio Markup

The App Studio markup language defines a set of HTML-like elements, or tags, for defining search queries to the backend and visualizing the search results and facets, handling all the complexities of connecting to Fusion.

You can get detailed information about every App Studio tag in the Reference Guides. You can also see visual examples with reusable code snippets in the App Studio Gallery.
Connect to Fusion

The most important tags in every page template are the first three, which configure how App Studio connects to Fusion:

- Define which provider to use (the search:platform).
- Create a query from the URL, or any parameters you explicitly declare (the search:query).
- Send the query to the platform and obtain a response (the search:response).

Below is an example from the default template:

```
<search:platform var="platform" conf="platforms.fusion.data"></search:platform>
<search:query var="query" parameters="*" results-per-page="12"></search:query>
<search:response var="response" platform="platform" query="query">
  <widget:spinner></widget:spinner>
</search:response>
```

The `search:platform` tag

The `search:platform` tag specifies the endpoint where App Studio obtains its data, pointing to a specific platform configuration endpoint.

See `search:query` for additional available parameters.

In this case we declare a data provider (platform) for a particular Fusion collection and set of parameters to search through an index.

```
<search:platform var="platform" conf="platforms.fusion.data"></search:platform>
```

See `search:platform` for additional available parameters.

The `search:query` tag

The `search:query` tag constructs an App Studio query object. You can choose to build a query from scratch by passing in the parameters you want or you can have it automatically created from the page URL.

Here's how you build a query using all query string parameters, and default to 12 results per page:

```
<search:query var="query" parameters="*" results-per-page="12"></search:query>
```

If you didn't want the user to be able to specify results per page you would simply omit that parameter when building a query from the URL:

```
<search:query var="query" parameters="*,-rpp" results-per-page="12"></search:query>
```

See `search:query` for additional available parameters.
The `search:response` tag

The `search:response` tag brings the platform and the query together.

In the example below we refer to the previously-defined platform and query by variable name:

```xml
<search:response var="response" platform="platform" query="query">
  <widget:spinner/>
</search:response>
```

See `search:response` for additional available parameters.
Control the page layout

The layout tags determine the layout and structure of the page. Using these ensures that the application will work equally well on desktops, mobiles and other devices (like high-resolution screens).

App Studio uses a grid layout to control how large sections appear on the screen, and how they behave when the visible area or resolution changes.

A simple grid:

```xml
<layout:grid>
  <layout:block small="1-2" large="1-4">
    Narrow section on large screens.
  </layout:block>
  <layout:block small="1-2" large="3-4">
    Wide on large screens.
  </layout:block>
</layout:grid>
```

See the following reference topics for complete information about layout tags:

- layout:animate
- layout:block
- layout:box
- layout:center-content
- layout:content
- layout:controls
- layout:grid
- layout:include
- layout:logo
- layout:sidebar
Anatomy of a View

In this topic, we'll explore the elements that compose the default App Studio view.

Open the code editor to see its complete contents; excerpts are included below to show how the page is constructed.

You can view many more examples, with downloadable code, in the App Studio Gallery.

The default view looks like this:

As we explore the components of the page, we’ll highlight each area in turn.
Include a partial file

The header of the default view is a partial file that is included with the `layout:include` tag:

```html
<layout:include file="views/partials/header.tpl.html" action="search"></layout:include>
```

The `header.tpl.html` file, in turn, contains a number of `layout` tags, plus the important `search:box` tag.
Sidebar with facet list

The sidebar is rendered using the simple `layout:sidebar` tag. Within the sidebar, the `search:facet-list` and `search:facet` tags contain the parameters that configure the facet fields and their display.

```html
<lucidworks>
</lucidworks>

<layout:block md="1-3" lg="1-4" drawer="left" id="sidebar" styling="blocksidebar-light">
  <layout:sidebar>
    <layout:box>
      <!-- List facets and give them intuitive labels -->
      <search:facet-list facet-names="genres_ss" response="response" platform="platform" query="query"
        styling="facet-list facet-list-wrappedheader">
        <search:facet when-field="genres_ss" show="10" search-enabled="true" platform="platform"
          query="query" max-characters="40" show="12" show-more="24" collapsible="true"></search:facet>
        <search:facet show="10" max-characters="40" show="12" show-more="24"
          collapsible="true"></search:facet>
      </search:facet-list>
    </layout:box>
  </layout:sidebar>
</layout:block>
Search statistics and results list

In the content area, *layout* tags are used again to construct the grid layout. For brevity, we'll omit those from the code samples below.

```html
<p class="response-statistics">(Fusion took {{ response.time }}ms)</p>

<search:result-list response="response" styling="cards-sm-1 cards-lg-2 cards-xl-2" platform="platform" query="query" instant-results="true">
  ...
</search:result-list>
```

The `search:response-statistics` tag displays the number of search results, and a `{{ response.time }}` variable displays the number of milliseconds Fusion took to respond:

The `search:result-list` tag lays out the grid of search results:
Individual results

The `search:result` and `search:field` tags configure which fields are displayed with each individual search result. A field can be styled as the title or description, and any additional fields can have a variety of styles.

**From Dusk Till Dawn**

1996

genres_ss  Action
          Comedy
          Horror
          Thriller

```html
<search:result>
  <search:field name="title_txt" styling="title" urlfield"></search:field>
  <search:field name="year_i" styling="description" max-characters="150"></search:field>
  <search:field name="genres_ss" styling="label-left"></search:field>
</search:result>
```
Null search results

In the case of no search results, the `search:no-results` tag displays a default message:

```html
<search:no-results response="response"></search:no-results>
```
Create a New Web App

The current version of App Studio creates a single-page search application that you can further customize through the built-in code editor.
Prerequisites

The instructions in this topic assume that you have a running instance of Fusion Server, with one or more collections of indexed data and at least one query profile.
How to create a new Web app

1. In the Fusion workspace, navigate to App Studio > Build New UI.

2. On the Set Query Source page, select the query profile to use for your search UI:
3. Click **Next**.

App Studio connects to the specified collection and query profile to analyze your searchable data:

4. Click **Next**.

5. On the **Set Result Title** page, select the field to use as the title of each search result:
6. Click Next.

7. On the Set Result Description page, select the field to use as the description of each search result:

8. Click Next.

9. On the Set Facets and Additional Fields page, select one or more multi-valued fields to use as facets:
Note

In some cases, facet fields aren’t selectable in the wizard. In this case, you can define facets later using markup.

Optionally:

- Select **Enable search within facet** to add a search box for searching the facet field names to the Facets pane.

- Enter a user-friendly label for each facet field, such as "Genre" instead of "genres_ss".

- Select **Include in results as Additional Field** to display the values of the facet fields within each search result.

10. Click **Next**.
11. On the **Customize UI** page, enter a title and select a color theme:

A preview of your search interface and data appear:

12. Click **Save and Launch UI**.

A preview of your search interface and data appear:

**Note**

It can take about a minute for the preview to appear. If the preview doesn’t appear, refresh the browser tab.

After you’ve created a search UI, you can open it from the Fusion workspace by navigating to the **App Studio** menu and selecting the search UI:
What’s next

- The code editor provides a handy interface for customizing and publishing your search UI.
- To customize your search UI, you use App Studio’s markup syntax.
Code Editor

App Studio's interface includes a built-in code editor that opens in your browser.

The code editor modifies the file that configures your search interface, using App Studio's markup tags. Your search interface reloads automatically when you save your changes in the code editor.

The code editor is also where you can publish your search interface to Fusion Server.
Opening the code editor

How to open the code editor

• Press the ESC key

or

• Click the code editor button:
The toolbar

Code editor toolbar

- Save and reload the search UI
- Maximize the editor
- Save as...
- Close the editor
Saving your changes

The **Save** button (or -S/CTRL-S) saves the file and reloads the page to display your updated interface:
Download a Project

If you prefer to develop your search interface using your own tools, you can download it as a .zip file.

The project directory includes tools for running a demo deployment or compiling an application for a production deployment.
Downloading a project from Fusion

You can download your App Studio project as a .zip file from its configuration panel in App Studio.

| Note | This option is unavailable after you have uploaded a WAR file for the project. |

1. From the App Studio menu, select your project:

   ![Project Configuration Panel]

   The project configuration panel appears.

2. Click **Download project**:
Your browser downloads the project as a .zip file.
Contents of the project directory

When you download your project and unpack the .zip file, you'll find these files and directories:

- **app-studio** (Unix) and **app-studio.bat** (Windows) are the scripts that build your project or launch a demo deployment.
- **bin** contains files used by the **app-studio** script, plus the **twigcrypt** utility.
- **src** contains the files that you modify to configure and customize your search interface.
- At the root level are these additional files that support the build process, plus a handy **README**:
  - karma.conf.js
  - package.json
  - pom.xml
  - README_STANDALONE.md
- Additional directories are created the first time you run the **app-studio** script:
  - dist
  - node_modules
  - target
The app-studio/app-studio.bat script

This script can do the following:

• Launch a demo deployment for previewing or demoing your UI development work
• Create an app with an embedded Tomcat web server
• Create an executable Java JAR file
• Compile a WAR file for deployment in a servlet container or in Fusion
Deploying a project

There are several ways to deploy a project. For information, see Deployment Overview.
Files and Directories

Most App Studio customization is performed by editing Web resources or configuration files. This topic shows you where to find the files you need.

The directory structure that you see depends on whether you are viewing files in the code editor or in a downloaded project.
Directory structure in the code editor

When you open the code editor, the files that configure the current search UI are available on the left:

- By default, views/search.html is selected for editing. This is the main file that controls the layout and styling of your search UI. The files in views/partials control the layout and styling of the header and footer of your UI.
- conf contains App Studio configuration files.
- assets contains image files, stylesheets, and other Web assets.
- styles contains the .less files that configure the theme of the UI. When your UI requires custom stylesheet information, add it to /app/styles/includes/custom.less.
Directory structure in a downloaded project

See Downloading A Project to learn how to download a project. When you download your project and unpack the .zip file, you’ll find these files and directories:

- **app-studio** (Unix) and **app-studio.bat** (Windows) are the scripts that build your project or launch a demo deployment.
- **bin** contains files used by the **app-studio** script, plus the **twigcrypt** utility.
- **src** contains the files that you modify to configure and customize your search interface.
- At the root level are these additional files that support the build process, plus a handy **README**:
  - **karma.conf.js**
  - **package.json**
  - **pom.xml**
  - **README_STANDALONE.md**
- Additional directories are created the first time you run the **app-studio** script:
  - **dist**
  - **node_modules**
  - **target**
The app-studio/app-studio.bat script

This script can do the following:

- Launch a demo deployment for previewing or demoing your UI development work
- Create an app with an embedded Tomcat web server
- Create an executable Java JAR file
- Compile a WAR file for deployment in a servlet container or in Fusion
Configuration files

You can find the configuration files in `conf/` in the code editor, or in `src/main/resources/conf/` in the project directory. The configuration files are organized by function or by module, as follows:

```
/conf
  /activity
  /message
  /platforms
  /processors
  /security
  /services
```

These configuration files support hierarchical cascading of the files via the open-source Fig project.

This section describes the most important configuration files in the `/config` folder:

- **config/twigkit.conf**
  
  Global configuration settings.

- **config/cors.conf**
  
  Cross-Origin Resource Sharing (CORS) configuration.

- **config/platforms/fusion/fusion.conf**
  
  Specify Fusion platform settings.

- **config/platforms/fusion/data.conf**
  
  Specify which Fusion query profile to use.

- **config/security/fusion.conf**
  
  Configure the Fusion security realm.

**config/twigkit.conf**

The App Studio application looks up global settings from a `twigkit.conf` configuration file that the application tries to locate relative to `config` on the runtime classpath.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>license-file</td>
<td>string</td>
<td>The path to the license file relative to the <code>app-studio/</code> directory, such as <code>file:///twigkit.lic</code>.</td>
</tr>
<tr>
<td>ui.trim-white-space</td>
<td>string</td>
<td>&quot;True&quot; to trim white space in the application UI.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>bundleTimeToLive</td>
<td>long</td>
<td>The number of milliseconds a resource bundle (for example, a file like <code>languages_en.properties</code>), can remain in the Java resource bundle cache without being validated against the source file from which it was constructed. The value 0 indicates that a bundle must be validated each time it is retrieved from the cache. If <code>bundleTimeToLive</code> is negative, or missing, then the bundle cache will have no expiration control (i.e. entries are only evicted from cache due to runtime or memory constraints).</td>
</tr>
</tbody>
</table>

**config/cors.conf**

CORS or Cross-Origin Resource Sharing is a recent W3C effort to introduce a standard mechanism for enabling cross-domain requests from web browsers to servers that wish to handle them. App Studio supports CORS filtering by default. You can control the options for the particular header attributes using the `cors.conf` file at the root of the `/conf` folder.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cors.allowOrigin</td>
<td>string</td>
<td>Whitespace-separated list of origins that the CORS filter must allow. Requests from origins not included here will be refused with an HTTP 403 &quot;Forbidden&quot; response. If set to * (asterisk), any origin is allowed.</td>
</tr>
<tr>
<td>cors.supportedMethods</td>
<td>string</td>
<td>List of the supported HTTP methods. These are advertised through the <code>Access-Control-Allow-Methods</code> header and must also be implemented by the actual CORS web service. Requests for methods not included here will be refused by the CORS filter with an HTTP 405 &quot;Method not allowed&quot; response.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: * GET, POST, DELETE, HEAD</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cors.supportsCredentials</td>
<td>boolean</td>
<td>Indicates whether user credentials, such as cookies, HTTP authentication or client-side certificates, are supported. The CORS filter uses this value in constructing the Access-Control-Allow-Credentials header.</td>
</tr>
<tr>
<td>cors.maxAge</td>
<td>integer</td>
<td>Indicates how long the results of a preflight request can be cached by the web browser, in seconds. If -1 unspecified. This information is passed to the browser via the Access-Control-Max-Age header.</td>
</tr>
</tbody>
</table>

**Hierarchical configuration**

To make version control and specialization of configuration easier, the configuration is loaded hierarchically. For example, in the `/platforms` directory are subdirectories for `/fusion` and `/solr`:

```
/config
    /platforms
        /fusion
        /solr
```

Within the `/fusion` folder is `/fusion/fusion.conf`, which may contain the following general attributes:

```
native: twigkit.search.fusion.Fusion
backwardsCompatible: true
timeOut: 30000
resultIDField: id
highlight: true
defaultQuery: *:*  
```

Within the same folder, you might find one or more configuration files which inherit or extend this one, such as `data.conf` or `people.conf`. To access a given configuration, use dot notation, such as `platforms.fusion.data` or `platforms.fusion.people`. The configuration system will traverse the hierarchy (no matter how deep) and aggregate the configuration files, overwriting attributes from higher-level files when those attributes also appear in lower-level files.

For example, if `fusion.conf` contains a `defaultQuery` attribute and you create `internal.conf` which also contains a `defaultQuery`, then the value from `internal.conf` is used. This allows you to create variations on the same platform configuration. You can then refer to any platform instance in other configuration files or in the platform tag:
In this case App Studio will use the platform as configured centrally, irrespective of the search engine behind it, effectively abstracting the data provider from the view.

**Configuration locations**

The configuration files can be placed outside of the application and centrally accessed by multiple instances. To reference these in a different location, set the `twigkit.conf` system property to the absolute file path of the `/conf` folder, like this:

```
-Dtwigkit.conf=file://${path-to-your-conf-folder}
```

**Adding special characters to key names in configuration files**

To add special characters to key names in configuration files you need to escape the special characters. For example, if a key name contained a whitespace, such as `My key: value` then you would need to escape the whitespace for the configuration to be correctly loaded. In this case, the correct syntax would be:

```
My\ key: value
```

This rule does not just apply to whitespace but any special character that you may wish to use within a key name.
Search Queries

The `search:query` tag instantiates a search query that can be submitted to Fusion. The `search:query` tag can use all applicable parameters from the query string (HTTP request) to create a query object. Queries can also include filters. You can use the `query:filter` tag and the `query:custom` tag in the body of a `search:query` tag to add invariant query filters or custom engine parameters, respectively.
Usage example

As an example, imagine the URL of the page looks like this:

http://intranet/search?q=guidelines&rpp=10

In this example, guidelines is the query term, and the resultsPerPage attribute (rpp) is limited to 10.

You can override the rpp to fix how many results should be displayed per page, or override the value specified in the URL query string. To do this, append the search:query tag with the long-form name of the attribute you want to override:

<search:query var="query" parameters="*" results-per-page="5" />

You can also clone a query instance using the from attribute:

<search:query var="query" from="${otherQuery}" resultsPerPage="5" />
Query Parameters

In the table below, we list all query parameters that can be specified as attributes on the `search:query` tag. For each parameter, we also list the corresponding short form that is used to encode the parameter in a URL request.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>var (String)</td>
<td>This is the name of the variable into which the query object is stored.</td>
</tr>
<tr>
<td>query parameters</td>
<td>The list of URL parameters to consider when constructing a new query instance. Setting parameters to <code>*</code> means that all parameters from the HTTP request are used to create a new query. You can specify a comma-separated list of parameters. You can also use negation (-) in conjunction with a wildcard to exclude one or more parameters; for example, <code>,-rpp</code> means that we look at all URL parameters except for <code>rpp</code>.</td>
</tr>
<tr>
<td>from (String)</td>
<td>Create a new query from existing query object.</td>
</tr>
</tbody>
</table>

For the parameters below, parameters in the URL override parameters in the `search:query` tag. To ignore URL parameters and always use the parameter in the tag, exclude the URL parameters by negating them with a hyphen (-) in the `parameters` attribute. For example, to ignore any occurrences of `rpp` in URLs, use `-rpp`, like this: `<search:query parameters="*,-rpp" results-per-page="5"`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query (&amp;q=) (String)</td>
<td>Use the query (or <code>q</code>) parameter to specify the query term to use.</td>
</tr>
<tr>
<td>modified-query (&amp;mq=) (String)</td>
<td>Specify an alternative query which is submitted to the search engine in place of the query parameter supplied by the user (but can include user-submitted parameters). To the user, it appears as if the query parameter had been used, but the modified-query is submitted to the search engine.</td>
</tr>
<tr>
<td>type (&amp;t=) (String)</td>
<td>Specifies how to interpret multiple query terms. Takes one of three values:</td>
</tr>
<tr>
<td></td>
<td>• any (boolean OR the terms)</td>
</tr>
</tbody>
</table>

For example, take the query "apple ipod". With `type="any"`, documents containing either term (apple OR ipod) are returned.
• **all** (boolean AND the terms)
  Documents containing both terms (apple AND ipod) are returned.

• **adv** (interpret using search platform-specific advanced query language)

`any`

results-per-page (`&rpp=`) (Integer)
The number of results to display per page.

10

max-results (`&max=`) (Integer)
The maximum number of results to evaluate. Defaults to -1, which means no upper limit.

-1

page (`&p=`) (Integer)
Set the page of results to display. The default value is 1 (1-based index), which would display the first page of results.

1

facets (`&fa=`) (String)
Specifies which facets to evaluate and return. Specified as a comma-separated list without spaces (such as facets=“cat,inStock”). This overrides the defaultFacets parameter of the `search:platform` tag.

fields (`&fi=`) (String)
A comma-separated list of fields that should be returned in the response.

sorts (`&s=`) (String)
Specifies fields and order for sorting. Sort order is specified by one of two characters:

• + for ascending
• - for descending

To sort ascending by the field price, the sort is specified as sorts=“+price”. Multilevel sorting is supported. To sort by multiple levels using the `search:query` tag, use & to separate the values. For example, to sort by price ascending and then SKU descending, use sorts=“+price-&sku”. To do the same using the URL query string, use multiple instances of the s parameter, such as ?&s=+price&s=-sku.

view (`&v=`) (String)
Set which search engine view to use. Behaviour is engine-dependent.

collection (`&co=`) (String)
Set which search engine collection to search against.

filters (`&f=`) (String)
Specifies filters to apply, specified as an ampersand (&) separated string. In the query string, however, there are multiple
&f= arguments. Filters are used to restrict field values that appear in results. We generally recommend using the query:filter tag for adding query filters instead of specifying them as an encoded string with this option.

custom (&c=) (String)

The custom attribute lets you send custom parameters to the search platform. Parameters are specified using the format parameter[=value] for a single parameter and corresponding value. Multiple parameters can be submitted with the search:query tag by separating them with & characters, as in custom="parameter1[=value1]&parameter2[=value2]". To submit multiple custom parameters using the URL query string, use multiple instances of the c parameter, such as ?c=parameter1[=value1]&c=parameter2[=value2]. We generally recommend using the query:custom tag for adding custom parameters instead of specifying them as an encoded string with this option.
Filters

You can apply filters to search queries.
Filter formats

Filters can have the following formats.

Short format

Filters are specified using the format `fieldName[value]['display name']`. The `['display name']` part is optional in all cases.

For example, adding `filters="locations['san francisco']"` to the query tag restricts the results to those with "san francisco" appearing in the `locations` field.

Multiple filters can be specified on the query tag by using the ampersand (`&`) as a separator character. For example, `filter="locations['san francisco']&people['obama']"` would restrict the results to those with the value "san francisco" appearing in the `locations` field and the value "obama" appearing in the `people` field.

Multiple filters can be specified with the URL query string by adding multiple `f` parameters: `?f=locations['san francisco']&f=people['obama']`

<table>
<thead>
<tr>
<th>Important</th>
<th>If the value is a string, it should be surrounded by quotes; all other primitives and dates should have the quotes omitted. For example:</th>
</tr>
</thead>
</table>
| • Filtering on string: `&f=stringField['My String']['String example']`  
• Filtering on integer: `&f=integerField[123]['Integer example']`  
• Filtering on double: `&f=doubleOrFloatField[123.45]['Double example']` |

Long format

When the query value of the filter differs from the display value, the format is slightly different to accommodate this extra information: `filters="fieldName[value]['display name']"`.

Range format

App Studio supports range queries for numbers, dates and strings. For example, `filters=salary[1000,100000]` searches the salary field for values between 1000 and 100000.

Date format

To filter by date (on date fields allowing range queries) use the DateFilter format (in all cases the display name is optional):

- To Filter on dates before the provided Date use: `filters=dateFieldName[,]yyyy-MM-dd'T'HH:mm:ss['Before the given date']`
- To Filter on dates after the provided Date use: `filters=dateFieldName[yyyy-MM-dd'T'HH:mm:ss,]"After the given date"
- To Filter on dates within a given range use: `filters=dateFieldName[yyyy-MM-dd'T'HH:mm:ss,yyyy-MM-dd'T'HH:mm:ss]["Date range"]`
To exclude the upper or lower limit of a date range it is possible to replace the [ and ] with a ( and ). For example:

```
filters=dateFieldName[yyyy-MM-dd'T'HH:mm:ss,yyyy-MM-dd'T'HH:mm:ss]["Date range excluding upper limit"]
```
Invariant filters

To configure a query with invariant constraints, you can add filter tags to the body of the query tag like so:

```xml
<search:query var="query" ... >
  <query:filter field="myField" value="myValue" />
</search:query>
```

Which constrains the query to all documents with the value “myValue” in `myField`. It is not possible to use ranges with the `query:filter` tag. The value is always interpreted as a string.

### Table 1. Tag attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>field (java.lang.String)</td>
<td>Name of the field to constrain.</td>
<td>none</td>
</tr>
<tr>
<td>value (java.lang.Object)</td>
<td>The value that the filter should match.</td>
<td>none</td>
</tr>
<tr>
<td>optional (java.lang.String)</td>
<td>In the case where multiple filters are being applied, this attribute may be used to determine how the filters may be combined to produce the desired query term. Values for this attribute are:</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>• field</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• value</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See Optional logical operators below for more details.</td>
<td></td>
</tr>
<tr>
<td>hidden (java.lang.Boolean)</td>
<td>Whether the filter is hidden or not.</td>
<td>false</td>
</tr>
<tr>
<td>exclude (java.lang.Boolean)</td>
<td>Whether filter is to exclude (negate) or not.</td>
<td>false</td>
</tr>
</tbody>
</table>
Optional logical operators

Use the optional tag attribute for a filter to specify a connective logical operator for the filter. This tells App Studio how to combine a filter with other filters.

Values for the optional attribute are:

- field
- value
- none

For example, if the commonality of the query as prescribed by the filters is found through field, then in this case you set the optional attribute to field.

See the corresponding Java enumerations below.

field – Fully optional condition

To make a given filter’s condition fully optional, use the field flag. This means that these filters are combined with the rest using an OR.

This is encoded in the query URL using a tilde (~) before the field name:

```
&f=foo['bar']&f=~marx['groucho']&f=~marx['harpo']*
```

This would result in a query like:

```
((foo:bar) OR (marx:groucho) OR (marx:harpo))
```

value – Values within a given field

Filters for the same field that are value optional are combined using an OR operator. This group of filters is then combined with the rest of the filters using a logical AND.

This is the recommended approach when using checkboxes to select values within a facet. In that case, the default behavior is to allow the user to select multiple options which are combined using an OR. This is encoded in the query URL using an asterisk (*) after the filter value:

```
&f=foo['bar']&f=marx['groucho']&f=marx['harpo']*
```

This results in a query like: *(foo:bar AND (marx:groucho OR marx:harpo))*

none – Logical conjunction (AND)

By default, individual filters (&f=) are applied to the query using logical conjunction (AND). This means that the documents returned must satisfy both filter conditions. This is the default connective.
Filter syntax quick reference

A quick reference to App Studio query filters is shown below.

- `<group-number>` is an integer value denoting a grouping of filter constraints.
- `<field-name>` is the name of the field to apply the constraint to. A valid field name is one that matches the regular expression `[A-Za-z_][A-Za-z_0-9\-\:\. ].`
- `<display-value>` is an optional label to attach to the filter.
- `<filter-value>` is the constraint to be applied to the specified field.
<filter-expression> ::= <field-expression> <group-expression> <value-expression> <display-expression>
<field-expression> ::= <exclusion-expression> <field-optional-expression> <field-name>
<exclusion-expression> ::= "-" | ""
<field-optional-expression> ::= "." | ""
<group-expression> ::= "(" <group-number> ")"
<group-number> ::= <digit> | <digit> <group-number>

<field-name> ::= <field-name-prefix> <field-name-suffix>
<field-name-prefix> ::= <letter> | ""
<field-name-suffix> ::= <field-symbol> | <field-symbol> <field-name-suffix>
<field-symbol> ::= <letter> | <digit> | "." | ":" | " " | ""

<value-expression> ::= <value-bracket-left> <filter-value> <value-bracket-right>
<value-bracket-left> ::= "[" | "(" 
<value-bracket-right> ::= "]" | ")"
<filter-value> ::= <value> | <range-value>
<value> ::= <string-expression> | <numeric-value> | <date-value>
<string-expression> ::= <partial-match> <string-value> <partial-match>
<string-value> ::= '"' <text> '"' | "'" <text> "'
.partial-match ::= '"' | ' '*
<numeric-value> ::= <floating-point> | <integer>
<number-sequence> ::= <digit> | <number-sequence> <digit>
<integer> ::= "0" | <positive-digit> <number-sequence>
-floating-point ::= <integer> | <integer> "." <number-sequence>
<date-value> ::= <four-digits> "-" <two-digits> "-" <two-digits> "'T'" <two-digits> ":" <two-digits> ":" <two-digits>
	<two-digits> ::= <digit> <digit>
<four-digits> ::= <two-digits> <two-digits>
<range-value> ::= <value> "," <value> | "," <value> | <value> ""
<display-expression-opt> ::= "" | "[" <display-expression> "]"
<display-expression> ::= "[" "" <display-value> "" "]" | "[" "" <display-value> """]"

where

<letter> ::= "A" | "B" | "C" | "D" | "E" | "F" | "G"
 | "H" | "I" | "J" | "K" | "L" | "M" | "N"
 | "O" | "P" | "Q" | "R" | "S" | "T" | "U"
 | "V" | "W" | "X" | "Y" | "Z"
 | "a" | "b" | "c" | "d" | "e" | "f" | "g"
 | "h" | "i" | "j" | "k" | "l" | "m" | "n"
 | "o" | "p" | "q" | "r" | "s" | "t" | "u"
 | "v" | "w" | "x" | "y" | "z";

<positive-digit> ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<digit> ::= "0" | <positive-digit>
Configure Autocomplete

App Studio supports the ability to generate suggestions on the fly, also called autocomplete or type-ahead, using the Terms component or Suggest component embedded in Fusion's Solr core.

The Solr Terms component provides access to the indexed terms in a field and the number of documents that match each term. This can be used, for example, to find all terms in the Solr index that match an expansion of the given query term. For example, "car" might return CARRY, CARRIED, CARE, CARL, and so on.

| Note | Since the Terms component only allows suggestions one term at a time, in cases when multiple terms are being typed, suggestions are given for the last term typed. |

How to configure suggestions with the Terms component

1. Configure Fusion:
   a. In Fusion, navigate to Querying > Query Pipelines.
   b. Select the query pipeline you are using.
   c. Select the Solr Query pipeline stage.
   d. Under configure Request Handlers Allowed for Queries, click Add.
   e. Enter "Terms".
   f. Click Save.

| Note | Unlike the Suggest component, you do not need to modify solrconfig.xml to use the Terms component; it is preconfigured. |

2. Configure App Studio:
   a. Open config/services/suggestions/terms.conf.
Its default content is as follows:

```yaml
name: twigkit.search.solr.suggestions.SolrQueryTermsService
source: platforms.fusion

# List out all fields to base your terms suggestions on
term-fields: title
title:
```

b. Change the default `source: platforms.fusion` to `source: platforms.fusion.data`.

c. Edit the value of `term-fields` as a comma-separated list of fields in your data from which to retrieve terms.

<table>
<thead>
<tr>
<th>Tip</th>
<th>This corresponds to Solr's <code>terms.fl</code> parameter.</th>
</tr>
</thead>
</table>

d. Edit the value of `title` to describe the group of suggestions returned by Fusion.

   This title can be used to group suggestions in the UI. If you are implementing only one group of suggestions, you can leave this blank or set it to something generic, like "Suggested phrases".

e. Save and close the file.

f. Open `app/views/partials/header.tpl.html`.

g. Uncomment the following line:

```xml
<query-suggestions completion-service="services.suggestions.terms" action="/search"></query-suggestions>
```

h. Save the file.

3. Refresh the search UI in your browser.

4. Test the feature by typing a search phrase.

   App Studio now displays suggested terms on the fly:

   [suggested terms]

The remaining attributes shown above are optional and are the same as described above for when backwards compatibility is disabled.
Social and Collaboration Tools

The social and collaboration tools provided by the Social Module extend the search application with the ability to store user-generated information on individual documents and document collections, and to share this information among groups of authenticated users. For example, documents can be tagged or bookmarked, and users may save search queries and add comments to individual documents.

User-generated content can be stored in a relational database or any other kind of data store, using our pluggable architecture. All social data, such as tags or bookmarks, is also available for indexing via web services, which make it possible to enhance search results based on input from users of the application.

Authentication and authorisation of users is managed via the Security module.
Prerequisite

Prior to using social and collaboration tools, you must enable them in Fusion and in App Studio.
Terminology

While much of the language used to describe collaborative tools is carefully chosen to be self explanatory, there are several general concepts which are used throughout and are introduced in this section.

Annotations

Key to the collaboration enabled by inclusion of this module, there are several types of annotation which can be applied to documents. These include bookmarks, comments, and likes, which are commonly found in many applications and provide a versatile way for users to contribute and save information during their search.

Topics

A topic is a general term for a subject that user(s) are intending to collect information about. This module provides facilities for users to create a named group with which they can associate any number of annotated documents. Topics themselves can be created and selected using the topic form tag. Many of the collaborative tags included in this module have a topic attribute which is used to link the saved information to a specific topic (which can be picked up from the URL - more on this in the Topic pages module. Topics can be defined by the developer as being 'namespaced' so they can be global (available for all users to collaborate on) or local (only viewable by the individual user that created them).

Collections

Some annotations can be categorised by collection. This is a mechanism for separating annotations of different types of document. There is a subtle difference between topics and collections, in that collections are usually determined by the categories of information in the data and generally pre-defined by the application developer offering a small number of collections with which to associate annotations. Collections are intended for a finite number of categorical groups whereas any number of topics can be created.
Combining social and collaboration tools

Topics and collections can even be combined such that a user can create a named ‘bucket’ to group information together, but have the saved information partitioned. One example is enabling the user to construct a wishlist of items that are automatically separated by category.
**Like Documents**

The Like Action Tag displays controls for the user to "like" a document. It also displays statistics on the number of times a specified target document has been liked.

**Usage**

The Like Action Tag is typically used within a Result List Tag, to enable the user to "like" a particular document in the result list:

```xml
<search:resultList response="${response}" >
  <search:result>
    <social:likeAction target="${result.id}" />
  </search:result>
</search:resultList>
```

**Required Tag Attribute**

target (java.lang.String):
The ID of the document that should be liked.

**Optional Tag Attributes**

collection (java.lang.String)
The collection to which the target document belongs.

topic (twigkit.model.Topic)
A topic to which the new "like" annotation should belong.

label (java.lang.String)
Label for the "like" action link.
Default: "Like"

tooltip (java.lang.String)
Tool tip for the "like" action link.
Default: "Do you like this?"

activeLabel (java.lang.String)
Label to show when document has already been liked by the current user.
Default: "Unlike"

activeTooltip (java.lang.String)
Tool tip to show when document has already been liked by the current user.
Default: "You have liked this."

singularLabel (java.lang.String)
Label for statistics (following the number of times the document in question has been liked) when the document has been liked by a single person.
Default: ""
**singularTooltip (java.lang.String)**
Tool tip for statistics when the document has been liked by a single person.
Default: "person likes this."

**pluralLabel (java.lang.String)**
Label for statistics (following the number of times the document in question has been liked) when the document has been liked by more than one person.
Default: ""

**pluralTooltip (java.lang.String)**
Tool tip for statistics when the document has been liked by more than one person." Default: "people like this."
Save and Retrieve Search Queries

The Appkit Social Module lets users save and retrieve their search queries. Use the Saved Query Form tag to let users save search queries. Use the Saved Query List tag to let users retrieve their saved search queries.

Saved Query Form tag

The Saved Query Form Tag displays controls for storing the current search query.

Usage

The Saved Query Form Tag is placed on a page to allow the authenticated user to store a reference to the current search query, under a given name:

```<social:savedQueryForm name="${search:hasDisplayValue(query.value) ? query.value.display : 'My saved search'}" query="${query}" /></social:savedQueryForm>
```

Required Tag Attributes

- **query** (twigkit.model.Query)
  The search query that should be stored.

Optional Tag Attributes

- **name** (java.lang.String)
  Default name for a query.
  Default:"

- **topic** (twigkit.model.Topic)
  A topic to which the saved query should belong.

- **legend** (java.lang.String)
  Legend for the form.
  Default: "Saved Searches"

- **buttonLabel** (java.lang.String)
  Label of form submit button.
  Default: "Save"

- **inputLabel** (java.lang.String)
  Label of form input box for the query name.
  Default: "Name"

- **placeholder** (java.lang.String)
  Placeholder value to put in the query name text box.
  Default: "Type a name for the current search query..."
Saved Query List tag

The Saved Query List tag retrieves and displays all the search queries that have been stored by a particular user.

Usage

The Saved Query List tag is typically used as follows:

```xml
<social:savedQueryList user="${social:getCurrentUserProfile()}" emptyText="You don't currently have any saved searches." />
```

Tag Attributes

- **user (twigkit.model.UserProfile)**
  Specifies the user whose queries should be retrieved.

- **topic (twigkit.model.Topic)**
  Optionally, saved queries can be scoped to a particular topic. If this attribute is set, then the tag will list only those queries that are linked to that topic.

- **format (java.lang.String)**
  Which date format to use.
  Default: "relative"

- **deleteLabel (java.lang.String)**
  Label to display for the delete action.
  Default: "x"

- **show (java.lang.Integer)**
  How many queries should be displayed initially.
  Default: 5

- **showMore (java.lang.Integer)**
  How many more queries should be added to the list each time the user clicks 'Show more'.
  Default: 10

- **showMoreLabel (java.lang.String)**
  Label to display for 'show more' action.
  Default: "Show more"

- **showLessLabel (java.lang.String)**
  Label to display for 'show fewer' action.
  Default: "Show fewer"

- **emptyText (java.lang.String)**
  Text to display if the user has not saved any queries.
  Default: ""

- **action (java.lang.String)**
  Path to prefix all query URLs with, e.g. "/mysearch".
  Default: ""
**JSP expressions**

These Social Module methods are useful for working with saved queries. They are accessible through JSP Expression Language (EL) expressions. Sample expressions are shown.

```java
twigkit.social.model.Topic getPublicTopicByPath(java.lang.String topicPath)
Finds and returns the public topic with the given path.
${social:getPublicTopicByPath('myTopic')}
```

```java
twigkit.social.model.Topic getPublicTopicByPathAutoCreate(java.lang.String topicPath)
Finds and returns the public topic with the given path, creating a new topic if one does not already exist.
${social:getPublicTopicByPathAutoCreate('myTopic')}
```

```java
twigkit.social.model.Topic getPrivateTopicByPath(java.lang.String topicPath, twigkit.social.model.Profile profile)
Finds and returns the topic that belongs to a specific user.
${social:getPrivateTopicByPath('myTopic', social:getCurrentUserProfile())}
```

For example, a saved query can be associated with a particular topic:

```xml
<social:savedQueryForm query="${myquery}" topic="${social:getPublicTopicByPathAutoCreate('myTopic')}" />
```

When the list of saved queries is displayed, the list is restricted to that particular topic:

```xml
<social:savedQueryList topic="${social:getPublicTopicByPath('myTopic')}" />
```
**Bookmark Search Results**

The Appkit Social module lets users bookmark search results:

- Use the `bookmarkAction` element to display bookmarking controls.
- Use the `bookmarkList` tag to retrieve and display bookmarks.

JSP Expression Language functions are available for getting a user's profile, the number of bookmarks the user has made, and whether a particular result has been bookmarked by a user.

**Bookmark Action tag**

The Bookmark Action tag displays controls for adding a new bookmark, in addition to statistics on the number of times a specified target document has been bookmarked.

**Usage**

The Bookmark Action tag is typically used within a Result List tag, to enable the user to bookmark a particular document in the result list:

```xml
<search:resultList response="${response}" >
  <search:result>
    <social:bookmarkAction target="${result.id}" url="${result.fields.url}" title="${result.fields.title" />
  </search:result>
</search:resultList>
```

Note that new bookmarks are always associated with the currently-authenticated user.

**Required tag attributes**

- `target` *(java.lang.String)*
  The ID of the document that should be bookmarked.

- `title` *(java.lang.String)*
  The title of the new bookmark.

**Optional tag attributes**

- `collection` *(java.lang.String)*
  The collection to which the target document belongs.

- `url` *(java.lang.String)*
  A URL which the new bookmark should link to.
  Default: the bookmark target, if it is a URL.

- `topic` *(twigkit.model.Topic)*
  A topic to which the new bookmark should belong.

- `label` *(java.lang.String)*
Label for the bookmark link.
Default: "Bookmark"

tooltip (java.lang.String)
Tool tip for the bookmark link.
Default: "Bookmark this?"

activeLabel (java.lang.String)
Label to show when document has already been bookmarked by the current user.
Default: "Remove bookmark"

activeTooltip (java.lang.String)
Tool tip to show when document has already been bookmarked by the current user. Default: "You have bookmarked this."

singularLabel (java.lang.String)
Label for bookmark statistics (following the number of bookmarks for the document in question) when there is a single bookmark.
Default: ""

singularTooltip (java.lang.String)
Tool tip for bookmark statistics when there is a single bookmark.
Default: "person has bookmarked this."

pluralLabel (java.lang.String)
Label for bookmark statistics (following the number of bookmarks for the document in question) when there are multiple bookmarks.
Default: ""

pluralTooltip (java.lang.String)
Tool tip for bookmark statistics when there are multiple bookmarks.
Default: "people have bookmarked this."

**Bookmark List tag**

The Bookmark List tag retrieves and displays all bookmarks that have been created by a particular user, are within a given topic, or are linked to a particular document (or any combination of the three constraints).

**Usage**

The Bookmark List tag is typically used as follows:

```twig
<social:bookmarkList user="${social:getCurrentUserProfile()}">emptyText="You don't currently have any bookmarks." /></social:bookmarkList>
```

**Tag attributes**

**user (twigkit.model.UserProfile)**
Specifies the user whose bookmarks should be retrieved.

target (java.lang.String)
If set, only display bookmarks related to this target document.

**topic (twigkit.model.Topic)**
Optionally, bookmarks can be scoped to a particular topic. If this attribute is set, then the tag will list only those bookmarks that are linked to that topic.

**collection (java.lang.String)**
Optionally, bookmarks can be scoped to a particular collection, identified by a string ID. If this attribute is set, then the tag will list only those bookmarks that belong to the given collection.

**format (java.lang.String)**
Which date format to use.
Default: "relative"

**deleteLabel (java.lang.String)**
Label to display for the delete action.
Default: "x"

**show (java.lang.Integer)**
How many bookmarks should be displayed initially.
Default: 10

**showMore (java.lang.Integer)**
How many more bookmarks should be added to the list each time the user clicks 'Show more'.
Default: 10

**showMoreLabel (java.lang.String)**
Label to display for 'show more' action.
Default: "Show more bookmarks"

**showLessLabel (java.lang.String)**
Label to display for 'show fewer' action.
Default: "Show fewer"

**emptyText (java.lang.String)**
Text to display if the user has not created any bookmarks.
Default: ""

**urlPattern (java.lang.String)**
A pattern which can be used to construct a custom URL action in the bookmark list. The pattern should be an expression that may include occurrences of the variable holder {{ target }} which will be replaced with the bookmark target at runtime.

**JSP expressions**

These Social Module methods are useful when working with bookmarks. They are accessible via JSP Expression Language functions. Sample expressions are shown.

```jsp
twigkit.model.UserProfile getCurrentUserProfile()
Get the profile of the currently active user.
${social:getCurrentUserProfile()}
```
long isBookmarkedByUser(twigkit.model.UserProfile user, java.lang.String id)
Indicates (by a positive value) whether or not the item with the given ID has been bookmarked by the specified user.
${social:isBookmarkedByUser(social:getCurrentUserProfile(),result.id) > -1 ? 'bookmarked-by-you' : ''}$

int getBookmarkCount(twigkit.model.UserProfile user, twigkit.model.Topic topic)
Returns the number of items that have been bookmarked by the user, within the scope of the topic specified (if the topic parameter is not null).
${social:getBookmarkCount(social:getCurrentUserProfile(), null)}$
Comment on Search Results

The Appkit Social module lets users comment on search results and display other users' comments. This gives users a flexible way to interact and contribute in an application, and can be used to implement features such as feedback, discussion, and reviewing systems.

- `<social:commentForm>` – Let users comment on search results.
- `<social:commentList>` – Display the comments of other users.

Usage

A typical use of the comment form is to embed it in a search result list, as shown here:

```xml
<search:resultList response="${symbol_dollar}{response}">
  <search:result>
    <!-- Title -->
    <search:field fieldName="title" url="${result.fields.url}" style="title" />
    <!-- Description and URL, Comments -->
    <search:field fieldName="summary" style="description" maxCharacters="250" />
    <search:field fieldName="url" style="url" maxCharacters="92" url="${result.fields.url}" />
    <social:commentForm target="${result.id}" buttonLabel="Post" placeholder="Leave a comment..." />
    <social:commentList target="${result.id}" format="dd MMM yyyy" />
  </search:result>
</search:resultList>
```

This is an example of how the tags can be rendered when used in this manner:
SYSTEMS ENGINEER/ SYSTEMS ADMINISTRATOR...

1
LONDON / £38000.00 ANNUAL SALARY
... communication then please apply now with your CV.
SYSTEMS ENGINEER/ ADMINISTRATOR, LONDON,
PERMANENT, ASAP.

Interview on Monday!

Twigkit@Twigkit.Com 29 Nov 2013
This one could be a good fit for me.

Linux Systems Administrator - London / Slough - £44-48k...

... all aspects of Linux systems administration including error... configuration of Linux systems using CentOS. ... and undertake system enhancements and...

System administrator LINUX, APACHE, CMS, HTML, SQL
Required Tag Attributes

target (java.lang.String)
The ID of the document that should be commented on.

Optional Tag Attributes

collection (java.lang.String)
The collection to which the target document belongs.

topic (twigkit.model.Topic)
A topic to which the new "comment" annotation should belong.

buttonLabel (java.lang.String)
Label of form submit button (default: "Post Comment").

placeholder (java.lang.String)
Placeholder value to put in the comment text box (default: "Write a comment...").

legend (java.lang.String)
Legend for the comment form (default: "Comment").

Make the comment form visible

By default the commentForm is hidden and cannot be displayed without an icon/button that toggles it. See the screenshot above for an example of this icon (in this case a blue speech bubble).

To ensure that this icon/button is visible on your page, so that a user can toggle the comment form, insert this code snippet inside a <search:result> tag:

```xml
<search:result>
  ...
  <div class="actions">
    <div class="comment"><a class="action"></a></div>
  </div>
  ...
</search:result>
```
Create Topics

The *Topic Form tag* lets users create named topics with which to associate saved information. Topics can be restricted to the user who creates a topic or global to all users.

An example of how to include the tag is shown below:

```xml
<%-- Topics --%>
<widget:popover id="topics" title="Cases" linkText="Cases" align="right">
    <social:topicForm name="My saved case" user="${social:getCurrentUserProfile()}" />
    <social:topicList user="${social:getCurrentUserProfile()}" format="dd MMM yyyy" emptyText="You don't currently have any cases." action="/case/" />
</widget:popover>
```

Typically this form would be placed inside the *search controls* of a search header.

**Required tag attributes**

There are no required tag attributes.

**Optional tag attributes**

- **name** (*java.lang.String*)
  Default title for a topic (default: ").

- **user** (*java.lang.String*)
  A user to which the new topic should belong (default: null).

- **legend** (*java.lang.String*)
  Legend for the form (default: "Topics").

- **buttonLabel** (*java.lang.String*)
  Label of form submit button (default: "Save").

- **inputLabel** (*java.lang.String*)
  Label of form input box for the topic (default: "Name").

- **placeholder** (*java.lang.String*)
  Placeholder value to put in the topic name text box (default: "Type a name for the new topic...").

- **namespace** (*java.lang.String*)
  Determines whether the topics created using the form will be "local" (restricted to the user who creates a topic) or "global" (available to all users). The default is "local".
Enable Social Features

Prior to using social and collaboration tools, you must enable them in Fusion and in App Studio.

Configure Fusion

How to configure Fusion

1. Update the managed-schema to include the new social fields:

```xml
<field indexed="true" multiValued="false" name="type" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="user_id" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="full_name" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="anonymous" required="false" stored="true" type="boolean"/>
<field indexed="true" multiValued="false" name="created" required="false" stored="true" type="pdate"/>
<field indexed="true" multiValued="false" name="created_epoch" required="false" stored="true" type="plong"/>
<field indexed="true" multiValued="false" name="name" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="path" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="private" required="false" stored="true" type="boolean"/>
<field indexed="true" multiValued="false" name="query_url" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="collection" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="target" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="title" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="description" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="url" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="accessibility" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="namespace" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="creator__anonymous" required="false" stored="true" type="boolean"/>
<field indexed="true" multiValued="false" name="creator__full_name" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="creator__id" required="false" stored="true" type="plong"/>
<field indexed="true" multiValued="false" name="creator__user_id" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="topic__id" required="false" stored="true" type="plong"/>
<field indexed="true" multiValued="false" name="topic__title" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="topic__private" required="false" stored="true" type="boolean"/>
<field indexed="true" multiValued="false" name="topic__path" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="topic__namespace" required="false" stored="true" type="string"/>
<field indexed="true" multiValued="false" name="topic__description" required="false" stored="true" type="string"/>
```
Configure App Studio

Configure App Studio to support social features.

Upgrade to the latest version of Appkit

Upgrade to the latest version of Appkit.

Enable social features

1. Create the file social.conf in the conf/social directory and add the following parameter to it:

```
platform: platforms.fusion.social
```

2. Create the file social.conf in the conf/platforms/Fusion directory and add the following parameters to it:

```
# Required for query/index pipeline
collection: fusion-app-name_user_data

# Required for query/index profiles
index-profile: fusion-app-name_user_data
query-profile: fusion-app-name_user_data

# Allow Fusion platform to store Social entities
readOnly: false
webservice-enabled: false
```
Persist Social Data

When using the Appkit Social module, an Appkit application can save social data (comments, bookmarks, and saved search queries) for logged in users. By default, the application saves the data in an in-memory database, the contents of which are lost when the application restarts.

You can configure Appkit to persist social data to a database that uses disk storage.

Requirements

- Any JDBC-compliant relational database (such as Oracle, MySQL, or SQL Server) or a Fusion deployment
- Network access from the Appkit application to the relational database or to Fusion

The database doesn't have to be empty because Appkit prefixes all of its tables with a twigkit identifier. But for simplicity, we recommend using a separate schema, because you probably won't need to join Appkit tables with tables in other schemas.

Persist data in a relational database

Perform the tasks in this section to persist Appkit social data in a relational database.

Overview of procedure

To configure a database for persistence on disk, perform the following steps, which are explained in more detail below:

1. Declare a Maven POM dependency in pom.xml to specify the database for storing the metadata that the Social module collects.
2. Configure a persistence.xml file with the appropriate elements, including those for database connections.
3. If necessary, perform other steps as required for specific databases.

The sections that follow describe these steps in more detail for the default Derby database and for other databases.

Use Derby

Declare a POM dependency

To persist social data in a disk-based database, declare a POM dependency in the file pom.xml, which is located at the root level of a project.

By default, the Appkit Social module uses the lightweight, in-memory Derby database to persist social data, as described in the following POM dependency in pom.xml:

```xml
<dependency>
  <groupId>twigkit</groupId>
  <artifactId>twigkit.social.provider.db.derby.memory</artifactId>
  <version>${project.parent.version}</version>
</dependency>
```

To persist social metadata in a disk-based Derby database, declare this POM dependency:
Create and configure persistence.xml

You’ll need to create and configure a persistence.xml file with the appropriate elements, including those for database connections.

Create the persistence.xml file here:
src/main/resources/META-INF/persistence.xml

See the Derby-memory and Derby examples.

Specify the type of Hibernate ID generator mappings

Hibernate ID generator mappings changed in Hibernate 5. Ensure use of the correct mappings as follows:

• Apps created with versions of Appkit that precede version 4.2:

   Ensure that both pre- and post-Hibernate 5 IDs are generated using the old mapping. Add this property to the persistence.xml file:

   <property name="hibernate.id.new_generator_mappings" value="false" />

• Apps created with Appkit 4.2 or later:

   Apps can use new ID generator mappings. Add this property to the persistence.xml file:

   <property name="hibernate.id.new_generator_mappings" value="true" />

Specify the Derby system directory

You can specify the Derby system directory, which is the directory that will contain the social database, by adding this flag to your Java options:

-Dderby.system.home=/my/derby/directory

If the system directory that you specify with derby.system.home doesn’t exist at startup, Derby creates the directory automatically.

Use a different relational database

You can configure Appkit to persist social data in an on-disk JDBC-compliant relational database other than Derby (such as Oracle, MySQL, or SQL Server).

Other pre-configured versions are also available, including:
1.1.1. Declare the POM dependency

To persist social metadata in a disk-based Derby database, change the POM dependency to:

**MySQL:**

```xml
<dependency>
  <groupId>twigkit</groupId>
  <artifactId>twigkit.social.db.provider.mysql</artifactId>
  <version>${project.parent.version}</version>
</dependency>
```

**Oracle:**

```xml
<dependency>
  <groupId>twigkit</groupId>
  <artifactId>twigkit.social.db.provider.oracle</artifactId>
  <version>${project.parent.version}</version>
</dependency>
```

**SQL Server:**

```xml
<dependency>
  <groupId>twigkit</groupId>
  <artifactId>twigkit.social.db.provider.sqlserver</artifactId>
  <version>${project.parent.version}</version>
</dependency>
```

1.1.2. Create and configure persistence.xml

You'll need to create and configure a persistence.xml file with the appropriate elements, including those for database connections.

Create the persistence.xml file here:

```
src/main/resources/META-INF/persistence.xml
```

View examples of persistence.xml files for MySQL, Oracle, and SQL Server.

1.1.3. Specify the type of Hibernate ID generator mappings

Hibernate ID generator mappings changed in Hibernate 5. Ensure use of the correct mappings as follows:

- **Apps created with versions of Appkit that precede version 4.2:**

  Ensure that both pre- and post-Hibernate 5 IDs are generated using the old mapping. Add this property to the persistence.xml file:
• Apps created with Appkit 4.2 or later:

Apps can use new ID generator mappings. Add this property to the persistence.xml file:

```xml
<property name="hibernate.id.new_generator_mappings" value="true" />
```

## Persist Social module data in Fusion

Perform the tasks in this section to persist Appkit Social module data in Fusion.

1. Configure Fusion.
2. (Only for apps created with prior versions of App Studio or Appkit) Upgrade to the latest version of Appkit.
3. Enable social features.

## Troubleshooting database configuration

Use these tips to troubleshoot problems with database configuration.

### Connection issues

Many databases require the application to manage the connections, refreshing them and maintaining enough open connections to service the load on the database. In some cases the default connection management provided is inadequate for production setups.

**If you notice bad performance, connections going stale, and associated errors (usually intermittently) the default connection pooling is probably inadequate for your environment.**

To remedy this situation you can use a third-party connection pooling technology. We recommend 'C3P0', which can be configured with the following simple steps:

1. Add the dependency for Hibernate c3p0 to the pom.xml:

```xml
<!-- Hibernate c3p0 connection pooling -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-c3p0</artifactId>
  <version>X.X.X.Final</version>
</dependency>
```

The version of Hibernate c3p0 you should use depends on the version of Appkit you are using. To begin with try version 4.1.7.Final (legacy) or 5.2.2.Final.

2. Add the configuration for C3P0 to the persistence.xml:
The settings above should be adequate for a standard Appkit application using the Social module, but they can be adjusted as desired.
Web Services

The web services REST API provides a set of simple interfaces for Social Module functionality.

All REST resources support both XML and JSON response formats, with XML being the default. To request a JSON response, the "Accept" header of the HTTP request for a particular REST resource should be set to "application/json". For example:

```
GET /twigkit/api/social/myrequest HTTP/1.1
Accept: application/json
...
```

Comment on documents

Users provide additional information about documents by commenting on them. For example, users might comment on the quality or relevance of a document, expand upon information in a document, or refer to related documents.

Methods and resource URIs are:

- GET /twigkit/api/social/comments: Retrieve comments that match one or more predicates.
- GET /twigkit/api/social/comment/{id}: Retrieve a single comment, identified by the id parameter.
- POST /twigkit/api/social/comment: Post a comment.
- DELETE /twigkit/api/social/comment/{id}: Delete a single comment, identified by the id parameter.

Bookmark documents

Bookmarks are used to identify documents for later retrieval or reference. Each bookmark is given a title, which serves to describe the intended reference. Like all other annotations, a bookmark can be optionally associated with a topic.

Methods and resource URIs are:

- GET /twigkit/api/social/bookmark/{id}: Retrieves a single bookmark, identified by the id parameter.
- POST /twigkit/api/social/bookmark: Creates a new bookmark for the authenticated user.
- DELETE /twigkit/api/social/bookmark/{id}: Deletes permanently the bookmark identified by the id parameter.
- GET /twigkit/api/social/bookmarks: Returns a collection of bookmarks matching some number of filter predicates.

Search for annotated documents

All annotations (bookmarks, comments, likes, and tags) are linked to external documents.

The method and resource URI are:

- GET /twigkit/api/social/annotated: Retrieves a list of documents that have been annotated, according to some search criteria.
Protect against CSRF attacks

To help protect against Cross-Site Request Forgery (CSRF) attacks on social web services, App Studio can ensure that all requests are tokenized. That is, each request is provided with a randomized Appkit request token.

Web Services for Comments

This article describes Appkit web services for comments.

Searching for comments

Users provide additional information about documents by commenting on them. For example, users might comment on the quality or relevance of a document, expand upon information in a document, or refer to related documents.

Searching for comments returns a collection of comments that match one or more filter predicates. Searches support filtering by target, by user, by topic, or by any combination of the three.

When a large number of comments match the specified predicates, only the 10 most recent entries are returned.

Methods and resource URIs

Methods and resource URIs are:

- GET /twigkit/api/social/comments: Retrieve comments that match one or more predicates.
- GET /twigkit/api/social/comment/{id}: Retrieve a single comment, identified by the id parameter.
- POST /twigkit/api/social/comment: Post a comment.
- DELETE /twigkit/api/social/comment/{id}: Delete a single comment, identified by the id parameter.

GET comments

Get comments that satisfy one or more predicates.

Parameters

<table>
<thead>
<tr>
<th>Note</th>
<th>At least one predicate parameter (target, userId, or topicId) must be specified.</th>
</tr>
</thead>
</table>

**target**
optional

Returns all comments that apply to the specified target document.

Example values: document123 or /story.news.yahoo.com/news?tmpl=story2

**userId**
optional

Returns comments that have been created by the user with the specified numerical ID.

Example values: 123

**topicId**
optional

Returns comments that are in the context of the topic with the specified numerical ID.
Example values: 456

Example request

Find all comments that were created by user 1:

As XML

GET /twigkit/api/social/comments?userId=1 HTTP/1.1
Accept: application/xml

<comments>
  <comment id="6">
    <created>2013-02-06T15:48:27.047Z</created>
    <creator id="1">
      <full-name>admin</full-name>
      <user-name>admin</user-name>
    </creator>
    <target>/news?u=/nm/20040814/bs_nm/column_mergers_dc</target>
    <text>Very good</text>
  </comment>
</comments>

As JSON

The same request, now calling for a response in JSON format:

GET /twigkit/api/social/comments?userId=1 HTTP/1.1
Accept: application/json

[  {
    "id":6,
    "created":1360165707047,
    "creator":{"id":1,"full-name":"admin","user-name":"admin"},
    "target":/news?u=/nm/20040814/bs_nm/column_mergers_dc",
    "text":"Very good"
  }
]

Data operations for comments

Basic operations for handling single comments are: create, retrieve, and delete.

Resource URI

/twigkit/api/social/comment
GET comment/{id}

Retrieve a single comment, specified by the id path parameter.

Parameters

id
required
A numerical value which uniquely identifies a comment. Given as a path parameter, not a query parameter.
Example value: 123

Example request

Retrieve comment with ID 6:

As XML

GET /twigkit/api/social/comment/6 HTTP/1.1
Accept: application/xml

```
<comment id="6">
  <created>2013-02-06T15:48:27.047Z</created>
  <creator id="1">
    <full-name>admin</full-name>
    <user-name>admin</user-name>
  </creator>
  <target>/news?u=/nm/20040814/bs_nm/column_mergers_dc</target>
  <text>Need to follow up on this</text>
</comment>
```

As JSON

The same request, now calling for a response in JSON format:

GET /twigkit/api/social/comment/6 HTTP/1.1
Accept: application/json

```
{
  "id":6,
  "created":1360165707047,
  "creator":{"id":1,"full-name":"admin","user-name":"admin"},
  "target":"/news?u=/nm/20040814/bs_nm/column_mergers_dc",
  "text":"Need to follow up on this"
}
```

POST comment

Creates a new comment for the authenticated user.

Parameters

target
required
A unique identifier for a document that is being commented on.
Example value: document123
Maximum length: 2083 characters

text
required
The text of the new comment.
Example value: Interesting document
Maximum length: Effectively none (that is, constrained by the maximum LOB length of the underlying database)

topic
optional
Numerical ID of a topic which the new comment should relate to.
Example value: 123

Example request
Create a new comment:

```
POST /twigkit/api/social/comment/6 HTTP/1.1
Content-Type: application/x-www-form-urlencoded
...
text=Follow%20up%20on%20this
```

```
<comment id="7">
  <created>2013-02-06T15:48:27.047Z</created>
  <creator id="1">
    <full-name>admin</full-name>
    <user-name>admin</user-name>
  </creator>
  <target>document99</target>
  <text>Follow up on this</description>
</bookmark>
```

DELETE comment/{id}
Permanently delete the comment specified by the id path parameter. The operation fails if the authenticated user is not
the user that created the comment in the first place.

Parameters

id
required
A numerical value which uniquely identifies a comment. Given as a path parameter, not a query parameter.
Example value: 123

Example request
Delete the comment with ID 6:
Web Services for Bookmarks

This article describes the Appkit web service for bookmarks.

Searching for bookmarks

Bookmarks are used to identify documents for later retrieval or reference. Each bookmark is given a title, which serves to describe the intended reference. Like all other annotations, a bookmark can be optionally associated with a topic.

Searching for bookmarks returns a collection of bookmarks that match one or more filter predicates. Supports filtering by target, by user, by topic, or by any combination of the three.

When a large number of bookmarks match the specified predicates, only the 10 most recent entries are returned.

Methods and resource URIs

Methods and resource URIs are:

- GET /twigkit/api/social/bookmark/{id}: Retrieves a single bookmark, identified by the id parameter.
- POST /twigkit/api/social/bookmark: Creates a new bookmark for the authenticated user.
- DELETE /twigkit/api/social/bookmark/{id}: Deletes permanently the bookmark identified by the id parameter.
- GET /twigkit/api/social/bookmarks: Returns a collection of bookmarks matching some number of filter predicates.

Parameters

<table>
<thead>
<tr>
<th>Note</th>
<th>At least one predicate parameter (target, userId, or topicId) must be specified.</th>
</tr>
</thead>
</table>

**target**

optional

Returns all bookmarks that apply to the specified target document.

Example values: `document123` or `/story.news.yahoo.com/news?tmpl=story2`

**userId**

optional

Returns bookmarks that have been created by the user with the specified numerical ID.

Example value: `123`

**topicId**

optional

Returns bookmarks that are in the context of the topic with the specified numerical ID.

Example value: `456`

**Example request**

Find all bookmarks that were created by user 1:
As XML

GET /twigkit/api/social/bookmarks?userId=1 HTTP/1.1
Accept: application/xml

<bookmarks>
  <bookmark id="6">
    <created>2013-02-06T15:48:27.047Z</created>
    <creator id="1">
      <full-name>admin</full-name>
      <user-name>admin</user-name>
    </creator>
    <target>/news?u=/nm/20040814/bs_nm/column_mergers_dc</target>
    <description>
      Reuters - Private investment firm Carlyle Group has quietly placed its bets on another part of the market.
    </description>
    <title>Carlyle Looks Toward Commercial Aerospace (Reuters)</title>
  </bookmark>
  <bookmark id="5">
    <created>2013-02-06T15:48:01.221Z</created>
    <creator id="1">
      <full-name>admin</full-name>
      <user-name>admin</user-name>
    </creator>
    <target>/news?u=/nm/20040814/bs_nm/markets_bears_dc</target>
    <description>
      Reuters - Short-sellers, Wall Street's dwindling band of ultra-cynics, are seeing green again.
    </description>
    <title>My bookmark</title>
  </bookmark>
</bookmarks>

As JSON

The same request, now calling for a response in JSON format:

GET /twigkit/api/social/bookmarks?userId=1 HTTP/1.1
Accept: application/json
Data operations for bookmarks

Basic operations for handling single bookmarks: create, retrieve, and delete.

Resource URI

/twigkit/api/social/bookmark

GET bookmark/{id}

Returns a single bookmark, specified by the id path parameter

Parameters

id
required
A numerical value which uniquely identifies a bookmark. Given as a path parameter, not a query parameter.
Example value: 123

Example request

Retrieve bookmark with ID 6:

As XML

GET /twigkit/api/social/bookmark/6 HTTP/1.1
Accept: application/xml
As JSON

The same request, now calling for a response in JSON format:

```
GET /twigkit/api/social/bookmark/6 HTTP/1.1
Accept: application/json
```

```
{
  "id":6,
  "created":1360165707047,
  "creator":{"id":1,"full-name":"admin","user-name":"admin"},
  "target":"/news?url=/nm/20040814/bs_nm/column_mergers_dc",
  "title":"Carlyle Looks Toward Commercial Aerospace (Reuters)",
  "description":"Reuters - Private investment firm Carlyle Group has quietly placed its bets on another part of the market."
}
```

POST bookmark

Creates a new bookmark for the authenticated user.

**Parameters**

**target**
required
A unique identifier for a document that is being bookmarked.
Example value: document123
Maximum length: 2083 characters

**title**
required
A title for the new bookmark.
Example values: Interesting document
Maximum length: 4000 characters

**description**
optional
A description of the bookmark, or the document it relates to.
Example values: This document could be useful later.
Maximum length: 4000 characters

**topic**
optional
Numerical ID of a topic which the new bookmark should relate to.
Example values: 123

**Example request**

**POST** details of a new bookmark:

```
POST /twigkit/api/social/bookmark/6 HTTP/1.1
Content-Type: application/x-www-form-urlencoded
...
title=Follow-up&description=Look%20further%20into%20this&amp;target=document99
```

```
<bookmark id="7">
  <created>2013-02-06T15:48:27.047Z</created>
  <creator id="1">
    <full-name>admin</full-name>
    <user-name>admin</user-name>
  </creator>
  <target>document99</target>
  <description>Look further into this</description>
  <title>Follow-up</title>
</bookmark>
```

**DELETE bookmark/{id}**

Permanently delete the bookmark specified by the id path parameter. The operation fails if the authenticated user is not the user that created the bookmark in the first place.

**Parameters**

**id**
required
A numerical value which uniquely identifies a bookmark. Given as a path parameter, not a query parameter.
Example value: 123

**Example request**

Delete bookmark with ID 6:

```
DELETE /twigkit/api/social/bookmark/6 HTTP/1.1
```

**Search for Annotated Documents**

All annotations (bookmarks, comments, likes, and tags) are linked to external documents.

Retrieve a list of documents that have been annotated, according to some search criteria. Without any parameters, the
10 most recently annotated documents are retrieved.

A document has been annotated if it has been tagged, bookmarked, "liked", or commented on.

**Method and resource URI**

The method and resource URI are:

- **GET /twigkit/api/social/annotated**: Retrieves a list of documents that have been annotated, according to some search criteria.

**Parameters**

*from*
optional
Returns documents that have been annotated at or later than the given date. The date needs to be formatted according to the pattern "yyyy-MM-dd HH:mm:ss".
Example value: 1999-07-23 22:10:04

*to*
optional
Returns documents that have been annotated at or earlier than the given date. The date needs to be formatted according to the pattern "yyyy-MM-dd HH:mm:ss".
Example value: 1999-07-23 22:10:04

*userId*
optional
Returns documents that have been annotated by the user with the specified numerical ID.
Example value: 123

*topicId*
optional
Returns documents that have been annotated in the context of the topic with the specified numerical ID. Example value: 456

**Example request**

Find all documents that have been annotated by user 1 since 14:10 on November 3, 2001:

**XML**

```
GET /twigkit/api/social/annotated?userId=1&from=2001-11-03%2014:10:00 HTTP/1.1
Accept: application/xml
```

```
<annotated from="2001-11-03 14:10:00" user-id="1">
  <target>document2</target>
  <target>document2</target>
</annotated>
```
The same request, now calling for a response in JSON format:

```
GET /twigkit/api/social/annotated?userId=1&from=2001-11-03%2014:10:00 HTTP/1.1
Accept: application/json
{
    "target": ["document1", "document2"],
    "user-id": "1",
    "from": "2001-11-03 14:10:00"
}
```

**Protect Against CSRF Attacks**

To help protect against Cross-Site Request Forgery (CSRF) attacks on social web services, App Studio can ensure that all requests are tokenized. That is, each request is provided with a randomized Appkit request token.

How to enable protection against CSRF in your application:

1. If it does not already exist, create a new file named `csrf.conf` and place it under `src/main/resources/conf/security`.
2. In that file, set the property `enabled` to `true`. This tokenizes the requests.
3. By default, the time-to-live of the Appkit request token is 60 minutes. You can change this by setting the property `duration` to some other value; for example, `duration: 30`. Responses received without a request token or with a request token has expired are rejected.

This configuration enables protection against CSRF attacks and sets the time-to-live of the request token to 30 minutes:

```
enabled: true
duration: 30
```
Localise an App

You can localise the UI labels in an app in multiple languages of your choice.
Localisation service

Appkit obtains the correct labels for a language by making calls to an Appkit translation service. The service provides a label in the chosen language, with a fallback to English if no translation exists for the label in the chosen language.

Loading translations

You can use the new `<translations:localize>` JSP tag to load a set of UI label translations into an app. The tag syntax is:

```html
<translations:localize dictionary="*dictionary*" locale="*locale*" />
```

- **dictionary** – Name of the resource bundle from which to load translations. This is also the first part of the name of each property file that contains translations.
  
  Default: translations

- **locale** – Locale
  
  Default: en (English)

The `dictionary` and `locale` are used to find the file as follows:

1. In `src/main/resources`, the Appkit translation service looks for a resource bundle named `dictionary`, for example, `translations`.
2. Within the resource bundle, the translation service looks for property files named `{resource}`, for example, for the files `translations_en.properties` and `translations_fr.properties`.

Enabling translations

1. Find label strings that need to be translated in the UI and identify the key names for the labels.
2. Create translation property files and add them under the `src/main/resources` directory. For example, for a dictionary called `translations` add `translations_en.properties` and `translations_fr.properties`.

Edit the property files and include the label keys and translations. For example:

In `translations_en.properties`:

```properties
# translations_en.properties
...
components.pagination.next = Next
components.pagination.previous = Previous
...
```

In `translations_fr.properties`:
3. Add the following dependency to your app’s `pom.xml`:

```xml
<dependency>
  <groupId>twigkit</groupId>
  <artifactId>twigkit.translations</artifactId>
  <version>${project.parent.version}</version>
</dependency>
```

4. Add this `taglib` directive at the top of your app’s `index.jsp`:

```jsp
<%@ taglib prefix="translations" uri="/twigkit/translations" %>
```

5. Add the Appkit `<translations:localize>` JSP tag inside the `<head>` element of your app’s `index.jsp`:

```jsp
<translation:localize locale="${param.locale}" />
```

6. Add this rule to the `url-rules.xml` file:

```xml
<rule>
  <from>^/lang/(.*)</from>
  <to last="true">/index.jsp?locale=$1</to>
</rule>
```

Note
---
You can modify other URL rules to suit your application logic. The `<translations:localize>` JSP tag works the same.

7. Restart your app.

8. Verify that the translations are present by visiting pages in the app that should be in different languages.
# Localisation Keys

This table identifies key names for UI labels and gives the default English labels.

For information about enabling translations, see Enabling translations.

<table>
<thead>
<tr>
<th>Key</th>
<th>English Label</th>
<th>Additional Context</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>components.search-box.placeholder</code></td>
<td>Search…</td>
<td>Search input placeholder</td>
</tr>
<tr>
<td><code>components.search-box.button-label</code></td>
<td>Search</td>
<td>Search submit button label</td>
</tr>
<tr>
<td><code>components.search-box.saved</code></td>
<td>My Saved Searches</td>
<td></td>
</tr>
<tr>
<td><code>components.search-box.bookmark</code></td>
<td>My bookmarks</td>
<td></td>
</tr>
<tr>
<td><code>components.spelling-suggestions.did-you-mean</code></td>
<td>Did you mean: {correction}.</td>
<td>As in, &quot;Did you mean spelling-corrected query?&quot;</td>
</tr>
<tr>
<td><code>components.spelling-suggestions.auto-correct</code></td>
<td>Showing results for {correction}.</td>
<td>This is displayed above a list of results of a spelling-corrected query.</td>
</tr>
<tr>
<td><code>components.spelling-suggestions.no-results</code></td>
<td>No results found for {query}.</td>
<td></td>
</tr>
<tr>
<td><code>components.response-statistics.showing</code></td>
<td>Showing {first} - {last} of {total}</td>
<td></td>
</tr>
<tr>
<td><code>components.pagination.next</code></td>
<td>Next</td>
<td></td>
</tr>
<tr>
<td><code>components.pagination.previous</code></td>
<td>Previous</td>
<td></td>
</tr>
<tr>
<td><code>components.breadcrumbs.clear-all</code></td>
<td>Clear All</td>
<td></td>
</tr>
<tr>
<td><code>components.facet.show-more</code></td>
<td>Show more</td>
<td></td>
</tr>
<tr>
<td><code>components.facet.show-less</code></td>
<td>Show less</td>
<td></td>
</tr>
<tr>
<td><code>components.no-results.title</code></td>
<td>Sorry: there are no results that match your search criteria.</td>
<td></td>
</tr>
<tr>
<td><code>components.no-results.subtitle</code></td>
<td>If you’ve checked your spelling: try using more general keywords.</td>
<td></td>
</tr>
<tr>
<td>Key</td>
<td>English Label</td>
<td>Additional Context</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>---------------------------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>components.response-statistics.showing-all</td>
<td>Showing All {total}</td>
<td></td>
</tr>
<tr>
<td>collaborate.bookmark.tooltip</td>
<td>Bookmark this?</td>
<td>As in, &quot;Would you like to bookmark this item?&quot;</td>
</tr>
<tr>
<td>collaborate.bookmark.active</td>
<td>You have bookmarked this</td>
<td>As in, this is already bookmarked.</td>
</tr>
<tr>
<td>collaborate.bookmark.single</td>
<td>{count} person has bookmarked this</td>
<td>As in the singular: &quot;1 person has bookmarked this.&quot;</td>
</tr>
<tr>
<td>collaborate.bookmark.plural</td>
<td>{count} people have bookmarked this</td>
<td>As in the plural: &quot;4 people have bookmarked this.&quot;</td>
</tr>
<tr>
<td>collaborate.bookmarklist.empty-text</td>
<td>You don’t have any bookmarks yet</td>
<td></td>
</tr>
<tr>
<td>collaborate.comment.prompt</td>
<td>Please enter a comment</td>
<td>As in a prompt to add a comment.</td>
</tr>
<tr>
<td>collaborate.comment.post</td>
<td>Post</td>
<td>The comment submit button</td>
</tr>
<tr>
<td>collaborate.comment.placeholder</td>
<td>Leave a new note...</td>
<td>The text area placeholder</td>
</tr>
<tr>
<td>collaborate.comment.delete</td>
<td>Delete</td>
<td>The delete comment button</td>
</tr>
<tr>
<td>collaborate.comment.more</td>
<td>Show more comments</td>
<td></td>
</tr>
<tr>
<td>collaborate.comment.less</td>
<td>Show fewer</td>
<td>As in, show fewer comments.</td>
</tr>
<tr>
<td>collaborate.comment.delete-message</td>
<td>Are you sure you want to remove this comment?</td>
<td></td>
</tr>
<tr>
<td>collaborate.like.tooltip</td>
<td>Do you like this?</td>
<td>As in, &quot;Do you like this item/result?&quot;</td>
</tr>
<tr>
<td>collaborate.like.active</td>
<td>You have liked this</td>
<td></td>
</tr>
<tr>
<td>collaborate.like.singular</td>
<td>{count} person has liked this</td>
<td>As in the singular: &quot;1 person has liked this.&quot;</td>
</tr>
<tr>
<td>collaborate.like.plural</td>
<td>{count} people have liked this</td>
<td>As in the plural &quot;2 people have liked this.&quot;</td>
</tr>
<tr>
<td>Key</td>
<td>English Label</td>
<td>Additional Context</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>------------------------------------</td>
<td>------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>collaborate.saved.button</td>
<td>Save</td>
<td>The save query button label</td>
</tr>
<tr>
<td>collaborate.saved.empty-text</td>
<td>You don't have any saved queries yet</td>
<td></td>
</tr>
<tr>
<td>collaborate.saved.error</td>
<td>Please enter a name so this query can be saved</td>
<td>Error message indicating a name is required</td>
</tr>
<tr>
<td>collaborate.saved.error-duplicate</td>
<td>A saved query with that name already exists, please use a different name</td>
<td>Error message flagging duplicate name</td>
</tr>
<tr>
<td>collaborate.saved.error-network</td>
<td>Unable to save query, please try again</td>
<td>General error message</td>
</tr>
<tr>
<td>collaborate.saved.placeholder</td>
<td>Search name</td>
<td>Guidance phrase in search box about searching names of saved searches</td>
</tr>
<tr>
<td>collaborate.topic.button</td>
<td>Save</td>
<td>The save topic button label</td>
</tr>
<tr>
<td>collaborate.topic.prompt</td>
<td>Please enter a topic name</td>
<td></td>
</tr>
<tr>
<td>collaborate.topic.name</td>
<td>Type a name for the new topic...</td>
<td>The text input placeholder</td>
</tr>
<tr>
<td>collaborate.topic.description</td>
<td>Description...</td>
<td>Input placeholder for the topic description text</td>
</tr>
<tr>
<td>collaborate.topic.empty</td>
<td>You don't have any topics yet</td>
<td></td>
</tr>
<tr>
<td>components.breadcrumbs.exclude</td>
<td>not {breadcrumb}</td>
<td>Showing the results that don't include items of this breadcrumb type</td>
</tr>
<tr>
<td>components.facet.apply-filters</td>
<td>Apply Filters</td>
<td></td>
</tr>
<tr>
<td>components.facet.no-results</td>
<td>No results found</td>
<td></td>
</tr>
<tr>
<td>components.facet.search</td>
<td>Search...</td>
<td></td>
</tr>
</tbody>
</table>
Demo Deployment

This deployment type is useful for:

• Evaluating App Studio
• Previewing the UI during development
• Demoing your UI

In order to run your UI in demo mode, you need a downloaded project.

From your project directory, run `app-studio start` or `app-studio.bat start` and go to `http://localhost:8080/` to view your demo deployment. The code editor is available in the demo interface; you can hide this by using the `--production` flag.
Startup in demo mode

Run the app-studio or app-studio.bat script like this:

```
```

After startup, your demo deployment is available at http://localhost:8080/.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--production</td>
<td>Run App Studio in &quot;production mode&quot;, with the code editor and wizard disabled. This gives you a preview of what your UI will look like in a production deployment.</td>
</tr>
<tr>
<td>-p &lt;port&gt;</td>
<td>Specify the port on which to start the App Studio Web server; the default is 8080.</td>
</tr>
<tr>
<td>-m &lt;memory&gt;</td>
<td>Set the minimum (-Xms) and maximum (-Xmx) heap size for the JVM, such as -m 4g which results in -Xms4g -Xmx4g. By default, this script sets the heap size to 512MB.</td>
</tr>
<tr>
<td>-t &lt;timeout&gt;</td>
<td>Set the startup timeout in seconds; the default is 120.</td>
</tr>
</tbody>
</table>
Shutdown in demo mode

To shut down your demo application, run `app-studio stop` or `app-studio.bat stop`:

```
app-studio stop [-p port] [-V]
```

The optional `-p` flag specifies the port to which the App Studio HTTP listener is bound.