Lucy

Plug-in Guide

5.1.0-2

escenic

A CCI COMPANY

Table of Contents

1 Introduction

Lucy is an Escenic plug-in that adds search capabilities to Escenic publications. It does this by providing an interface to **Apache Solr**, a high-performance, full-featured text search engine based on the **Apache Lucene** Java search library.

This guide contains information on:

- Installation
- Configuration
- Administration
- Adding search functionality to Escenic templates

A set of general search template components (JSPs and tag libraries) is provided with the Content Engine. These components, which can be used to build search pages for Escenic publications, are based on the **Escenic search framework**. This framework provides a generic search interface to Lucy, but also to several other search engines. The following illustration shows how these components work with Lucy.

For detailed information about Solr, visit http://lucene.apache.org/solr/. For detailed information about Lucene, visit http://lucene.apache.org/.

2 Installation

The following preconditions must be met before you can install Lucy 5.1.0-2:

- The Content Engine is installed and in working order.
- The Escenic assembly tool has been extracted and successfully used to set up a test EAR file as described in the **Escenic Content Engine Installation Guide**.
- You have the correct distribution file (**`lucy-5.1.0-2.zip`**).

Plug-ins must be installed using the Escenic assembly tool.

2.1 Install Lucy

Installing Lucy involves the following steps:

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

   ```
   $ cd /mnt/download
   $ https://user:password@maven.escenic.com/com/escenic/plugins/lucy/lucy/5.1.0-2/
   lucy-5.1.0-2.zip
   ```

   Otherwise, download it to some temporary location of your choice.
3. If the folder **`/opt/escenic/engine/plugins`** does not already exist, create it:

   ```
   $ mkdir /opt/escenic/engine/plugins
   ```
4. Unpack the Lucy distribution file:

   ```
   $ cd /opt/escenic/engine/plugins
   $ unzip /mnt/download/lucy-5.1.0-2.zip
   ```

   This will result in the creation of an **`/opt/escenic/engine/plugins/lucy`** folder.
5. Log in as **`escenic`** on your **assembly-host**.
6. Run the **`ece`** script to re-assemble your Content Engine applications.

   ```
   $ ece assemble
   ```

   This generates an EAR file (**`/var/cache/escenic/engine.ear`**) that you can deploy on all your **engine-host**s.
7. If you have a single-host installation, then skip this step.

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

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

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

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

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

2.1.1 Configuration

Configuring Lucy involves a couple of simple tasks, described in the following sections. In these instructions, the placeholder *escenic-config* is used to represent the path of your Escenic configuration, as defined with the **`com.escenic.config`** property in *escenic-home*`/assemblytool/assemble.properties`. If **`com.escenic.config`** is not defined, then *escenic-config* has a default definition of *escenic-home*`/localconfig`.

2.1.1.1 Set SearchManager Properties

1. Open *escenic-config*`/neo/io/search/SearchManager.properties` for editing.
2. Uncomment the **`searchEngine`** and **`defaultSearchEngine`** property definitions and set them both to **`/com/escenic/lucy/LucySearchEngine`**:

```
searchEngine=/com/escenic/lucy/LucySearchEngine
defaultSearchEngine=/com/escenic/lucy/LucySearchEngine
```

3. Save the file and exit.

2.1.1.2 Set LucySearchEngine Properties

By default, Lucy looks for its search engine at **`http://localhost:8080/solr`**, which is the location of the default Lucy search engine, called **`LucySearchEngine`**. If this is not the Solr engine you want Lucy to use, then you will need to:

1. Copy **`LucySearchEngine.properties`** from *escenic-home*`/engine/plugins/lucy/siteconfig/localconfig/com/escenic/lucy/LucySearchEngine.properties` to *escenic-config*`/com/escenic/lucy/LucySearchEngine.properties` for editing.
2. Open the copied file for editing.
3. Set the **`solrURI`** property to point to the correct Solr engine location. For example:

```
solrURI=http://mysolrhost:8080/solr
```

4. Give the search engine instance a unique name by setting the name property. For example:

```
name=AnotherLucyInstance
```

5. Save the file and exit.

2.1.1.3 Multiple Search Engine Instances

In a development phase it is acceptable to use the same Solr search engine instance for both internal (i.e Content Studio) searches and for publication search functionality. In a production environment, however, this is usually not the case. Internal users need access to all content, both published and unpublished, whereas readers of an organization's publications should only be presented with published search results.

In a production environment, therefore, there should always be at least two Solr instances, one for internal use and one for public use. Lucy should be set up to use the public, filtered Solr

instance. You can change the Solr instance used by Lucy by setting the **`solrURI`** property in **`LucySearchEngine.properties`** (see section 2.1.1.2).

For information about the installation and set-up of Solr instances at Escenic installations, see the **Escenic Content Engine Installation Guide**. For more general information about Solr, see http://lucene.apache.org/solr/.

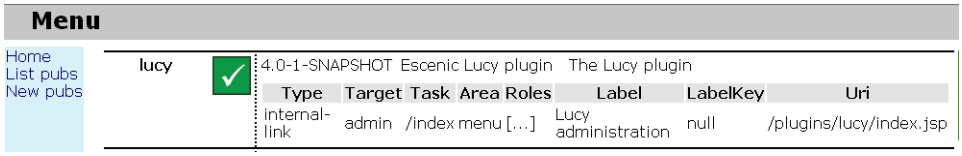
2.2 Verify The Installation

To verify the status of the Lucy installation, open the Escenic Admin web application (usually located at **`http://`***server*`/escenic-admin`) and click on **View installed plugins**. The status of the plug-ins is indicated as follows.

The plug-in is correctly installed.

The plug-in is not correctly installed.

So if all is well, you should see something like this:



You must then verify that Lucy is also correctly configured and actually works:

1. Start a browser.
2. Open the Escenic Admin web application (usually located at **`http://`***server*`/escenic-admin`).
3. Click on **Lucy Administration**. This displays a page from which you can execute a test search.
4. Enter some search criteria and click on **Submit Query**. Lucy should then search all publications and return the results on a new page.

2.3 Troubleshooting

This section lists some common problems and describes how to fix them.

ClassNotFoundException
: This may appear on the plug-in status page or in the application server log. It indicates that the plug-in is not correctly installed or not installed at all. Check the contents of the **`engine.ear`** file and verify that all the libraries located in **`lucy/lib`** are present in the EAR file's lib folder.

404 Not Found
: This may appear in the browser, when accessing a Lucy-related link. It probably means that the web application has not been updated with the plug-in files. Check that the corresponding WAR file has been updated by the assembly tool, and redeploy the web application if necessary. Note that the correct WAR file to deploy is located in the assembly tool's **`dist/war`** folder, not in *escenic-home*`/webapps`.

ClassCastException, NoSuchMethodError
: This may appear in the application server log or in the browser. It means that there is a mismatch between the code version of the Content Engine and the code version of the plug-in. Contact Escenic for an update.

For other problems contact Escenic support.

3 Getting started

The fastest way to get started with the Lucy plug-in is to install the Lucy demo publication and play with the templates it contains. The demo publication is included in the Lucy installation in *escenic-home*`/plugins/lucy/wars/lucy-demo.war`.

3.1 Installing The Demo Publication

To install the demo publication:

1. Copy *escenic-home*`/plugins/lucy/wars/lucy-demo.war` to *escenic-home*`/assemblytool/publications/lucy-demo.war`.
2. Create a `lucy-demo.properties` file in the *escenic-home*`/assemblytool/publications` folder with the following contents:

   ```
   context-root=/lucy-demo
   name=lucy-demo
   source-war=lucy-demo.war
   ```

3. Rebuild and deploy the Content Engine.
4. Restart the Content Engine.
5. Open the Escenic Admin web application (usually located at `http://`*server*`/escenic-admin`).
6. Select **New Pubs** and use the displayed form to upload `lucy-demo.war`.
7. Select **create a publication**.
8. Enter a name (`lucy-demo`) and password for the publication in the displayed form.
9. Select **Submit**.
10. A publication information page for the new publication should now be listed. To open the publication, click on the link under the heading **Browse the publication**.

4 Searching

You can add search functionality to your templates in one of two different ways:

- Use the Struts search forms and actions described in the **Escenic Content Engine Struts Form Reference**.
- Use the lower level **`neo.xredsys.content.search.Expression`** bean.

The recommended method is to use the Struts search forms, as this is the simplest and quickest approach. It is also the method used in the Lucy demo publication.

Whichever method you use, the process involves the following basic steps:

1. Construct the search expression.
2. Execute the search.
3. Present the results.

4.1 Searching Using Struts

The Lucy demo publication's search template contains the following:

```
...taglib declarations...

<TEMPLATE:call file="searchForm_simple.jsp" />

...error handling code...

<TEMPLATE:call file="result.jsp" />
```

The first included file, **`searchForm_simple.jsp`** carries out steps 1 and 2 of the search process (constructing a search expression and executing it), while the second file, **`result.jsp`**, presents the results of the search. This workflow is also defined in the publication's Struts configuration file, **`struts-config.xml`**. Here is an extract from this file:

```
    <action path="/search/simple"
            parameter="method"
            type="com.escenic.search.SearchAction"
            name="SimpleSearchForm"
            input="/template/searchForm_simple.jsp"
            scope="session">
      <forward name="success"
               path="/template/result.jsp"
               redirect="false"/>
    </action>
```

For a proper introduction to Struts, see http://struts.apache.org/primer.html. In short, however, the above code:

- Defines **`searchForm_simple.jsp`** as a **`SimpleSearchForm`**

- Specifies that the data input to the form will be processed by the Struts action **/search/simple**, which is an action of type **com.escenic.search.SearchAction**.
- Specifies that output from the **/search/simple** action will be directed to **result.jsp**.

Here is the contents of **searchForm_simple.jsp**:

```
...taglib declarations...

<HTML:form action="/search/simple">
  <input type="hidden" name="successUrl" value="/template/common.jsp" />
  <input type="hidden" name="errorUrl" value="/template/common.jsp" />
  <HTML:hidden property="pageLength" value="5" />

  <input type="hidden" name="publicationId" value="<BEAN:write name="publication"
 property="id" />" />
  <HTML:hidden property="searchEngineName" value="LucySearchEngine" />
  <HTML:hidden property="includeSubSections" value="true" />

  <SECTION:use uniqueName="ece_frontpage">
    <BEAN:define id="secId" name="section" property="id" toScope="request"/>
  </SECTION:use>
  <BEAN:define id="allSecId" name="secId" scope="request" />
  <HTML:hidden property="includeSectionId" value="<%= String.valueOf(allSecId) %>" />

  <HTML:hidden property="articleType" value="default" />
  <HTML:hidden property="sortString" value="score" />
  <HTML:text property="searchString" />
  <HTML:submit />
</HTML:form>
```

The items to pay particular attention to here are:

- The **HTML**, and **BEAN** prefixes reference Struts tag libraries declared at the top of the file.
- The **SECTION** prefix references an Escenic tag library, also declared at the top of the file.
- All of the form's **HTML:hidden** and **HTML:text** fields are bound to form properties that define search parameters. For descriptions of these properties, see the descriptions of **SearchForm**, **ArticleSearchForm** and **SimpleSearchForm** in the **Escenic Content Engine Struts Form Reference**.
- The **HTML:hidden**, are, as the name suggests, hidden: they are not displayed on the page. The form displayed in the publication therefore consists of a single input field (**searchString**) and a **Submit** button.
- **LucySearchEngine** is the name of the default search engine instance used by Content Studio. If you have created another search engine instance that you want to use (see section 2.1.1.2), then you should specify the name of this instance in the **searchEngineName** property. For example:

  ```
  <HTML:hidden property="searchEngineName" value="AnotherLucyInstance" />
  ```

When the publication reader enters a string in the field and clicks on **Submit**, the **/search/simple** action is executed and returns a set of **com.escenic.search.ResultPage** beans, each representing a page of search results. The search form's **pageLength** property (see above) determines how many results each page contains, and therefore also how many result pages are returned.

The demo publication's **result.jsp** template simply contains code for cycling through the returned **ResultPage** beans and displaying their contents:

```
<div class="result">
  <LOGIC:present name="com.escenic.search.ResultPage">
    <LOGIC:equal value="0" name="com.escenic.search.ResultPage" property="totalHits">
      <h3>No articles found.</h3>
    </LOGIC:equal>
    <LOGIC:greaterThan value="0" name="com.escenic.search.ResultPage"
 property="totalHits">
      <div class="display">
        Showing <BEAN:write name="com.escenic.search.ResultPage" property="fromHits" /
>
              - <BEAN:write name="com.escenic.search.ResultPage" property="toHits" />
              of <BEAN:write name="com.escenic.search.ResultPage"
 property="totalHits" />
      </div>
      <hr/>
      <ul>
        <LOGIC:iterate id="result" name="com.escenic.search.ResultPage"
 type="com.escenic.search.Result">
          <li>
            <h3>
              <a href="<BEAN:write name="result" property="url" />">
                <BEAN:write name="result" property="field(title)" />
              </a>
            </h3>
            <p>
              <LOGIC:notEmpty name="result" property="description">
                <BEAN:write name="result" property="description" />
              </LOGIC:notEmpty>
            </p>
            <p>
              <a href="<BEAN:write name="result" property="url" />">Linktext</a>
            </p>
            <hr/>
          </li>
        </LOGIC:iterate>
      </ul>
      <div class="navigator">
        <UTIL:notEqual name="com.escenic.search.ResultPage" property="pageNumber"
 value="1">
          <a href="<BEAN:write name="com.escenic.search.ResultPage" property='<%=
"url[1]"%>' />">First</a>
        </UTIL:notEqual>
        <BEAN:define id="pageNr"
                     name="com.escenic.search.ResultPage" property="pageNumber"
 type="Integer" />
        <BEAN:define id="pages"
                     name="com.escenic.search.ResultPage" property="numberOfPages"
 type="Integer" />
        <UTIL:loop id="pageNumber" from="1" to="<%= pages %>">
          <UTIL:equal name="pageNumber" value="<%= pageNr.toString() %>">
            <UTIL:notEqual name="com.escenic.search.ResultPage" property="pageNumber"
 value="1">
              |
            </UTIL:notEqual>
            <strong>
              Page <BEAN:write name="com.escenic.search.ResultPage"
 property="pageNumber" />
              of <BEAN:write name="com.escenic.search.ResultPage"
 property="numberOfPages" />
            </strong>
```

```
          </UTIL:equal>
          <UTIL:notEqual name="pageNumber" value="<%= pageNr.toString() %>">
            |
            <a href="<BEAN:write name="com.escenic.search.ResultPage"
                                 property='<%= "url[" + pageNumber + "]"%>' />">
              <BEAN:write name="pageNumber" />
            </a>
          </UTIL:notEqual>
        </UTIL:loop>
        <UTIL:notEqual name="com.escenic.search.ResultPage" property="pageNumber"
 value="<%= pages.toString() %>">
          |
          <a href="<BEAN:write name="com.escenic.search.ResultPage"
                               property='<%= "url[" + pages + "]"%>' />">
            Last
          </a>
        </UTIL:notEqual>
      </div>
    </LOGIC:greaterThan>
  </LOGIC:present>
</div>
```

Each **`ResultPage`** bean has a **`result`** property containing an array of **`com.escenic.search.Result`** beans, each of which contains one of the search results returned by Lucy. The **`Result`** bean's **`url`** property contains the URL of a content item, while the **`description`** property contains the result description (an extract from the content item). The **`field`** property contains all the content item's fields. In the example above, the content items' **`title`** fields are retrieved and used as titles for each result.

For full descriptions of the **`ResultPage`** and **`Result`** beans, see the **Escenic Content Engine Bean Reference**.

4.2 Searching Using The Expression Bean

It is possible, although more difficult, to add search functionality to your templates while avoiding dependency on Struts. In order to do so, you need to make use of the **`neo.xredsys.content.search.Expression`** bean. This bean can be used to build an expression that can be submitted to a search engine (Lucy in this case). It does not, however, provide any assistance with regard to form display or handling; you must manage all that yourself in your templates.

When a search is submitted using the **`Expression`** bean, it returns a single **`neo.xredsys.content.search.SearchResult`** bean. This bean has no inbuilt page structure, it simply contains a list of **`neo.xredsys.content.search.SearchHit`** beans, each of which represents a single search result. You must therefore manage pagination of the results yourself in your templates.

The use of the Java objects on which these beans are based is described below. You can, however, access the objects as beans from JSP in the usual way.

4.2.1 Starting a Search

To execute a search you first need to create a **`neo.xredsys.content.search.Expression`** object as follows:

```
neo.xredsys.content.search.Expression expression = new
 neo.xredsys.content.search.DefaultExpression();
```

Then you need to set a couple of the object's properties as follows:

```
expression.setParameter("searchEngineName", "LucySearchEngine");
expression.setSearchString("search-string");
```

The first property must be either be set to the name of the default search engine instance used by Content Studio, **`LucySearchEngine`**, or to the name of another search engine instance that you have defined (see section 2.1.1.2).

The second property is the string to search for. Specifying **`Escenic*`**, for example, will search for all articles containing words that start with the string **Escenic**.

Finally, you can execute the query as follows:

```
neo.xredsys.content.search.SearchResult result =
 neo.xredys.api.IOAPI.getAPI().getObjectLoader().articleSearch(expression);
```

4.2.1.1 Expression Properties

The **`neo.xredsys.content.search.Expression`** object has a number of properties that you can use to define search parameters. All the search parameter properties supported by Lucy are described below.

No parameters are required, but at least one parameter must be defined. If not, the search will fail.

`searchString`
The search expression. If the expression is empty or *, it is ignored.

```
expression.setSearchString("Escenic");
```

Using very short expressions may cause the search engine to throw an exception. You are therefore recommended not to accept expressions shorter than 3 characters.

`sort`
Sort order for the search results.

```
expression.setSort("by_date_asc");
```

Allowed values are:

`score`
Hits are sorted by their score.

`by_score`
Hits are sorted by their score.

`by_date_asc`
Hits are sorted by date in ascending order (oldest first).

`by_date_desc title`
Hits are sorted by date in descending order (newest first).

title
Hits are sorted alphabetically by content item title.

types
Search only for content items of the specified types.

```
expression.setTypes(new String[]{"news", "facts"});
```

nodes
Search only for content items related to one or more of the specified sections.

```
expression.setNodes(new int[]{3,12,14});
```

The numbers in the supplied array are section IDs.

excludeNodes
Ignore any content items that belong to one of the specified sections (that is, have one of the specified sections as their home section).

```
expression.setExcludeNodes(new int[]{5});
```

The example ensures that any content items with section **5** as their home section will not be included in the search result. Content items with other home sections that simply appear in section 5 will, however, be included.

fromDate
Search only for content items published after the specified date.

```
expression.setFromDate(new Date());
```

toDate
Search only for content items published before the specified date.

```
expression.setToDate(new Date());
```

parameter
Used to hold several named search parameters. The parameters supported by Lucy include:

author
Search only for content items written by the specified author.

```
expression.setParameter("author", "Ben*");
```

Searches for content items written by all authors whose names starts with **Ben**.

resultsPrPage
Limits the number of hits included in the search result.

Sample:

```
expression.setParameter("resultsPrPage", "10");
```

4.2.2 Showing Search Results

Search results are returned in a **neo.xredsys.content.search.SearchResult** object, which implements **java.util.List**. This makes it quite easy to iterate through the results and extract information about each hit, as shown below:

```
      Iterator i = result.iterator();
      while(i.hasNext()) {
        neo.xredsys.content.search.SearchHit hit =
(neo.xredsys.content.search.SearchHit)i.next();
```

```
        System.out.println("ArticleID:" + hit.getObjectId() + " Title " +
  hit.getField("title"));
      }
```

First an iterator is created, and then used to access each **`neo.xredsys.content.search.SearchHit`** in the result set; **`articleID`** and **`title`** properties are then retrieved from each **`SearchHit`** object and displayed. For a full list of all **`neo.xredsys.content.search.SearchHit`** properties, see section 4.2.2.1.

The correct way to construct links from search results is as follows for an Escenic site:

```
http://www.site/current-pub/eceRedirect?articleId=articleIdD&pubId=pubId
```

where:

site is the name of the site.

current-pub is the name of the current publication.

articleID is the **`articleID`** property of the **`SearchHit`** object.

pubID is the **`pubID`** property of the **`SearchHit`** object.

For a portal site, however, you should construct it as follows:

```
http://www.portal/current-pub/Ece2PortalUrl?articleId=articleIdD
```

where:

portal is the name of the portal.

current-pub is the name of the current publication.

articleID is the **`articleID`** property of the **`SearchHit`** object.

When creating a link to the actual article, the HTTP link should point to **http://www.site.com/[pub]/eceRedirect?articleId=[articleId]&pubId?[pubId]** for Escenic sites, or **http://www.site.no/[pub]/Ece2PortalUrl?articleId=[articleId]** for Portal sites. This is also for avoiding the use of the Escenic API during iteration of the search results.

For further information about the **`SearchResult`** object, see the **Escenic Content Engine Bean Reference**.

4.2.2.1 SearchHit Properties

The **`neo.xredsys.content.search.SearchHit`** object has the following properties:

docId
The content item ID.

pubId
The ID of the publication to which the content item belongs.

title
The content item's title.

displayText
A short summary consisting of text from the shortText field, and also from the longText field if the shortText field doesn't provide enough text.

publishedDate
The content item's published date.

homeSection
The ID of the content item's home section.

contentType
The content item's type.

score
The search hit score returned by the search engine. This is a number between 0-100 that provides an indication of the quality of the hit.