How to Sync Elements Connect Fields with Different Key Values in HP ALM/QC

Last Modified on 04/09/2024 6:04 am EDT

Warning: We are moving Exalate for HP QC/ALM into basic maintenance mode. This transition period will last until November 2024, after which support will be halted completely. For more information, please see https://exalate.com/blog/end-of-support-hp/.

Note: You can sync Elements Connect fields versions 5.12 and lower.

Elements Connect (previously known as nFeed) is a Jira add-on which allows the integration of external information into the context of an issue. This information can come from databases, web-services or other sources. For instance, it can be used to show customer information such as a telephone number, address and so on, which has been looked-up from a CRM database.

This add-on is particular in the way that the information is stored. Instead of storing the information which is showed in the issue, it only stores an identifier of that information allowing to lookup the actual values.

There are 2 challenges manipulating this data, especially when each side of a synchronisation relation is using a different data store.

Looking up Elements Connect Values

The challenge when synchronizing these custom fields between two instances is that the values need to be looked up in the same way as Elements Connect does. As the add-on doesn't provide a JIRA level API to actually do this lookup, a workaround needs to be used.

Elements Connect provides a REST API allowing to extract the values as shown in the issue. Below is a snippet allowing to use this REST API to extract the required information.

```
// NFeed REST Api returns a json which needs to be parsed.  Using the JsonSlurper
import groovy.json.JsonSlurper


// A generic function which can be reused for accessing rest resourses
def getNfeedValue =  {
  String issueKey,  customFieldId ->

  URL url;
  def baseURL = "http://foo.exalate.net/rest/nfeed/1.0/customfield";


  String userpass = "user:password";
  String basicAuth = "Basic " + new String(new Base64().encoder.encode(userpass.getBytes()));


  // GET the value using the 'Export' method
  url = new URL(baseURL + "/${issueKey}/${customFieldId}/EXPORT");
  URLConnection connection = url.openConnection();
  connection.requestMethod = "GET"
  connection.setRequestProperty ("Content-Type", "application/json;charset=UTF-8")
  connection.setRequestProperty ("Authorization", basicAuth);
  connection.connect();

  // Parse the JSON
  String jsonString = new Scanner(connection.getContent(),"UTF-8").useDelimiter("\A").next();
  def jsonSlurper = new JsonSlurper()
  def jsonObject = jsonSlurper.parseText(jsonString)

  // the display contains the value displayed in an issue
  return jsonObject.displays

}

replica.customKeys.Products = getNfeedValue(issue.key, "customfield_10302")
```

Storing Elements Connect Values

To be able to store Elements Connect values, one needs to understand how the custom field is configured.
In the example below, Elements Connect depends on a table `audience_d2` that is stored in the JIRA database.
The code:

- Opens a connection to the database.
- Retrieves the right id from the table based on values provided by the remote instance.
- Sets the local custom fields to the found id.

```
/*
** Import the necessary packages allowing to look up data in a table
*/

import groovy.sql.Sql
import groovy.sql.GroovyRowResult
import java.sql.Connection
import org.ofbiz.core.entity.DelegatorInterface
import com.atlassian.jira.ofbiz.OfBizDelegator
import com.atlassian.jira.component.ComponentAccessor
import org.ofbiz.core.entity.ConnectionFactory


// build connection to the local JIRA database
OfBizDelegator delegator = ComponentAccessor.getOfBizDelegator();
DelegatorInterface delegatorInterface = delegator.getDelegatorInterface();
String helperName = delegatorInterface.getGroupHelperName("default");
Connection connection = ConnectionFactory.getConnection(helperName);


// Retrieve information from replica allowing to look up the right value

def targetProject = replica.customFields."Target Project".value.value
def targetProgram = replica.customFields."Target Program".value.value


def query = "select id " +
    "from audience_d2 " +
    "where upper(iproject) = '" + targetProject.toUpperCase() + "' " +
    "and   upper(itype)   = '" + replica.issueType.name.toUpperCase() + "'" +
    "and   upper(iprogram) = '" + targetProgram.toUpperCase() + "'"

// query for the right data
Sql sql = new Sql(connection);
List<GroovyRowResult> resultRows = sql.rows(query);
sql.close();


if (resultRows.size() != 0) {
  def idRecord = resultRows.get(0).get("id")
  issue.customFields."Target Category".value = idRecord
  issue.customFields."Target Assignment".value = idRecord
}
```

How to connect to a MySQL Database?

Here is an example to connect to a MySQL database:

```
import groovy.sql.Sql

// connect to db
def sql = Sql.newInstance("jdbc:mysql://localhost:3306/test",
        "user", "password", "com.mysql.jdbc.Driver")

// execute a simple query
sql.eachRow("SELECT city, zipcode from cities"){ row ->
   // print data returned by the query
   println "City = $row.city - Zipcode = $row.zipcode"
}

// close the connection - don't forget !!!
sql.close()
```

Have more questions? Ask the community

Product

About Us ↗

Glossary ↗

API Reference ↗

Security ↗

Pricing and Licensing ↗

Resources

Subscribe for a weekly Exalate hack ↗

Academy ↗

Blog ↗

YouTube Channel ↗

Ebooks ↗

Still need help?

Join our Community ↗

Visit our Service Desk ↗

Find a Partner ↗