How to develop a Hyperlink Options Filter

In this tutorial, we will follow the Guideline for Developing a Plugin to develop our Hyperlink Options Filter plugin. Please also refer to the very first tutorial How to develop a Bean Shell Hash Variable for more details steps.

1. What is the problem?

We want to have a filter similar to the following.

All (18) | Published (17) | Draft (1) | Trash (2)

2. How to solve the problem?

We will develop a List Filter Type Plugin to render our Hyperlink Options Filter.

3. What is the input needed for your plugin?

To develop a Hyperlink Options Filter plugin, we will need to provide some inputs as following.

1. The options to populate as links
2. Whether or not to show the data count of each options.

4. What is the output and expected outcome of your plugin?

A list of hyperlinks which will list all the options with its data count. When click the hyperlink will filter the datalist.

All (18) | Published (17) | Draft (1) | Trash (2)

5. Are there any resources/API that can be reused?

Refer to List Filter Type Plugin.

6. Prepare your development environment

We need to always have our Joget Workflow Source Code ready and builded by following this guideline.

The following tutorial is prepared with a Macbook Pro and the Joget Source Code is version 8.0-Snapshot. Please refer to the Guideline for Developing a Plugin article for other platform commands.

Let's say our folder directory is as follows.

```
- Home
  - joget
    - plugins
    - jw-community
```

The "plugins" directory is the folder we will create and store all our plugins and the "jw-community" directory is where the Joget Workflow Source code is stored.

Run the following command to create a maven project in "plugins" directory.

```
cd joget/plugins/
~/joget/jw-community/wflow-plugin-archetype/create-plugin.sh org.joget hyperlink_options_filter 8.0-Snapshot
```

Then, the shell script will ask us to key in a version number for the plugin and ask us for a confirmation before it generates the maven project.

```
Define value for property 'version':  1.0-SNAPSHOT: : 8.0-Snapshot
[INFO] Using property: package = org.joget
Confirm properties configuration:
groupId: org.joget
artifactId: hyperlink_options_filter
version: 5.0.0
package: org.joget
Y: : y
```

We should get a "BUILD SUCCESS" message shown in our terminal and a "hyperlink_options_filter" folder created in the "plugins" folder.

Open the maven project with your favourite IDE. I will be using NetBeans.

7. Just code it!

a. Extending the abstract class of a plugin type

Create a "HyperlinkOptionsFilter" class under "org.joget" package. Then, extend the class with **org.joget.apps.datalist.model.DataListFilterTypeDefault** abstract class. Please refer to List Filter Type Plugin.

b. Implement all the abstract methods

As usual, we have to implement all the abstract methods. We will using AppPluginUtil.getMessage method to support i18n and using constant variable MESSAGE_PATH for message resource bundle directory.

Implementation of all basic abstract methods

```
package org.joget;

import org.joget.apps.app.service.AppPluginUtil;
import org.joget.apps.app.service.AppUtil;
import org.joget.apps.datalist.model.DataListFilterTypeDefault;

public class HyperlinkOptionsFilter extends DataListFilterTypeDefault {
    private final static String MESSAGE_PATH = "message/HyperlinkOptionsFilter";

    public String getName() {
        return "Hyperlink Options Filter Type";
    }

    public String getVersion() {
        return "5.0.0";
    }

    @Override
    public String getLabel() {
        //support i18n
        return AppPluginUtil.getMessage("org.joget.HyperlinkOptionsFilter.pluginLabel", getClassName(),
MESSAGE_PATH);
    }

    @Override
    public String getDescription() {
        //support i18n
        return AppPluginUtil.getMessage("org.joget.HyperlinkOptionsFilter.pluginDesc", getClassName(),
MESSAGE_PATH);
    }

    public String getClassName() {
        return this.getClass().getName();
    }

    public String getPropertyOptions() {
        return AppUtil.readPluginResource(getClass().getName(), "/properties/hyperlinkOptionsFilter.json",
null, true, MESSAGE_PATH);
    }
}
```

Now, we have to create a UI for admin user to provide inputs for our plugin. In getPropertyOptions method, we already specify our Plugin Properties Options definition file is located at "/properties/hyperlinkOptionsFilter.json". Let us create a directory "resources/properties" under "hyperlink_options_filter/src/main" directory. After creating the directory, create a file named "hyperlinkOptionsFilter.json" in the "properties" folder.

In the properties definition options file, we will need to provide options as below. Please note that we can use "@@message.key@@" syntax to support i18n in our properties options.

```
[{
    title : '@@HyperlinkOptionsFilter.config@@',
    properties : [{
        name:'defaultValue',
        label:'@@HyperlinkOptionsFilter.defaultValue@@',
        type:'textfield'
    },
    {
        name : 'showLabel',
        label : '@@HyperlinkOptionsFilter.showLabel@@',
        type : 'checkbox',
        options : [{
            value : 'true',
            label : ''
        }]
    },
    {
        name : 'displayFull',
        label : '@@HyperlinkOptionsFilter.displayFull@@',
        type : 'checkbox',
        value : 'true',
        options : [{
            value : 'true',
            label : ''
        }]
    },
    {
        name : 'showCount',
        label : '@@HyperlinkOptionsFilter.showCount@@',
        type : 'checkbox',
        value : '',
        options : [{
            value : 'true',
            label : ''
        }]
    },
    {
        name : 'options',
        label : '@@HyperlinkOptionsFilter.options@@',
        type : 'grid',
        columns : [{
            key : 'value',
            label : '@@HyperlinkOptionsFilter.value@@'
        },
        {
            key : 'label',
            label : '@@HyperlinkOptionsFilter.label@@'
        }]
    }]
},
{
    title : '@@HyperlinkOptionsFilter.chooseOptionsBinder@@',
    properties : [{
        name : 'optionsBinder',
        label : '@@HyperlinkOptionsFilter.optionsBinder@@',
        type : 'elementselect',
        options_ajax : '[CONTEXT_PATH]/web/property/json/getElements?classname=org.joget.apps.form.model.
FormLoadOptionsBinder',
        url : '[CONTEXT_PATH]/web/property/json[APP_PATH]/getPropertyOptions'
    }]
}]
```

After completing the properties option to collect input, we can work on the main methods of the plugin which are getTemplate and getQueryObject method. In getTemplate method, we will retrieve options and its count based on the configured plugin properties.

```
    public String getTemplate(DataList datalist, String name, String label) {
        PluginManager pluginManager = (PluginManager) AppUtil.getApplicationContext().getBean("pluginManager");
        Map dataModel = new HashMap();
```

```
        dataModel.put("element", this);
        dataModel.put("name", datalist.getDataListEncodedParamName(DataList.PARAMETER_FILTER_PREFIX+name));
        dataModel.put("label", label);

        Map<String, String> options = getOptionMap();
        if ("true".equalsIgnoreCase(getPropertyString("showCount"))) {
            DataListBinder binder = datalist.getBinder();
            for (String key : options.keySet()) {
                DataListFilterQueryObject filter = getQueryObject(datalist, name, key);
                int count = 0;
                if (binder != null) {
                    if (filter != null) {
                        count = binder.getDataTotalRowCount(datalist, binder.getProperties(), new
DataListFilterQueryObject[]{filter});
                    } else {
                        count = binder.getDataTotalRowCount(datalist, binder.getProperties(), new
DataListFilterQueryObject[]{});
                    }
                }

                options.put(key, options.get(key) + " (" + count + ")");
            }
        }

        String value = getValue(datalist, name, getPropertyString("defaultValue"));
        dataModel.put("value", value);
        dataModel.put("options", options);

        return pluginManager.getPluginFreeMarkerTemplate(dataModel, getClassName(), "/templates
/hyperlinkOptionsFilter.ftl", null);
    }

    protected Map<String, String> getOptionMap() {
        Map<String, String> optionMap = new ListOrderedMap();

        // load from "options" property
        Object[] options = (Object[]) getProperty(FormUtil.PROPERTY_OPTIONS);
        for (Object o : options) {
            Map option = (HashMap) o;
            Object value = option.get(FormUtil.PROPERTY_VALUE);
            Object label = option.get(FormUtil.PROPERTY_LABEL);
            if (value != null && label != null) {
                optionMap.put(value.toString(), label.toString());
            }
        }
        // load from binder if available
        Map optionsBinderProperties = (Map) getProperty("optionsBinder");
        if (optionsBinderProperties != null && optionsBinderProperties.get("className") != null && !
optionsBinderProperties.get("className").toString().isEmpty()) {
            PluginManager pluginManager = (PluginManager) AppUtil.getApplicationContext().getBean
("pluginManager");
            FormBinder optionBinder = (FormBinder) pluginManager.getPlugin(optionsBinderProperties.get
("className").toString());
            if (optionBinder != null) {
                optionBinder.setProperties((Map) optionsBinderProperties.get("properties"));
                FormRowSet rowSet = ((FormLoadBinder) optionBinder).load(null, null, null);
                if (rowSet != null) {
                    optionMap = new ListOrderedMap();
                    for (FormRow row : rowSet) {
                        Iterator<String> it = row.stringPropertyNames().iterator();
                        // get the key based on the "value" property
                        String value = row.getProperty(FormUtil.PROPERTY_VALUE);
                        if (value == null) {
                            // no "value" property, use first property instead
                            String key = it.next();
                            value = row.getProperty(key);
                        }
                        // get the label based on the "label" property
                        String label = row.getProperty(FormUtil.PROPERTY_LABEL);
                        if (label == null) {
```

```
                        // no "label" property, use next property instead
                        String key = it.next();
                        label = row.getProperty(key);
                    }
                    optionMap.put(value, label);
                }
            }
        }
    }

    if (!optionMap.containsKey("")) {
        Map<String, String> tempOptionMap = new ListOrderedMap();
        tempOptionMap.put("", AppPluginUtil.getMessage("HyperlinkOptionsFilter.all", getClassName(),
MESSAGE_PATH));
        tempOptionMap.putAll(optionMap);
        optionMap = tempOptionMap;
    }

    return optionMap;
}

protected DataListFilterQueryObject getQueryObject(DataList datalist, String name, String value) {
    DataListFilterQueryObject queryObject = new DataListFilterQueryObject();
    if (datalist != null && datalist.getBinder() != null && value != null && !value.isEmpty()) {
        String columnName = datalist.getBinder().getColumnName(name);
        List<String> valuesList = new ArrayList<String>();
        String query = "("+columnName+" = ? or "+columnName+" like ? or "+columnName+" like ? or
"+columnName+" like ?)";
        valuesList.add(value);
        valuesList.add(value + ";%");
        valuesList.add("%;" + value + ";%");
        valuesList.add("%;" + value);
        queryObject.setOperator(DataListFilter.OPERATOR_AND);
        queryObject.setQuery(query);
        queryObject.setValues(valuesList.toArray(new String[0]));
        return queryObject;
    }
    return null;
}

public DataListFilterQueryObject getQueryObject(DataList datalist, String name) {
    String value = getValue(datalist, name, getPropertyString("defaultValue"));
    return getQueryObject(datalist, name, value);
}
```

In the getTemplate, we specify the template file to "hyperlinkOptionsFilter.ftl". Let create this file under "hyperlink_options_filter/src/main/resources/templates" directory. Then, using FreeMaker syntax to construct our template as below:

```
<div id="${name!}_container" style="display:none;margin:5px 0;">
    <input id="${name!}" name="${name!}" type="hidden" value="${value!?html}" />
    <#if element.properties.showLabel! == "true" >
        <label><strong>${label!?html} :</strong></label>  
    </#if>
    <#list options?keys as key>
        <a ref="${key?html}" href="${key?html}" class="<#if value! == key >active</#if>"><span><#if value! ==
key ><strong></#if>${options[key]!?html}<#if value! == key ></strong></#if></span></a>  
    </#list>
    <script type="text/javascript">
        $(document).ready(function(){
            <#if element.properties.displayFull! == "true" >
                $('#${name!}_container').insertBefore($('#${name!}_container').closest(".filters"));
            </#if>
            $('#${name!}_container').show();
            $('#${name!}_container a').click(function(){
                var value = $(this).attr("ref");
                $(this).parent().find("input").val(value);
                $(this).closest("form").submit();
                return false;
            });
        });
    </script>
</div>
```

c. Manage the dependency libraries of your plugin

We need to include "commons-collections" library in our POM file.

```
        <!-- Change plugin specific dependencies here -->
        <dependency>
            <groupId>commons-collections</groupId>
            <artifactId>commons-collections</artifactId>
            <version>3.2.1</version>
        </dependency>
        <!-- End change plugin specific dependencies here -->
```

d. Make your plugin internationalization (i18n) ready

We are using i18n message key in getLabel and getDescription method. We will use i18n message key in our properties options definition as well. Then, we will need to create a message resource bundle properties file for our plugin.Create a directory, "resources/message", under "hyperlink_options_filter/src/main" directory. Then, create a "HyperlinkOptionsFilter.properties" file in the folder. In the properties file, add all the message keys and its label as below.

```
org.joget.HyperlinkOptionsFilter.pluginLabel=Hyperlink Options
org.joget.HyperlinkOptionsFilter.pluginDesc=Show options as Hyperlink to perform filter.
HyperlinkOptionsFilter.all=All
HyperlinkOptionsFilter.config=Configure Hyperlink Options Filter
HyperlinkOptionsFilter.options=Options
HyperlinkOptionsFilter.value=Value
HyperlinkOptionsFilter.label=Label
HyperlinkOptionsFilter.chooseOptionsBinder=Choose Options Binder
HyperlinkOptionsFilter.optionsBinder=Options Binder
HyperlinkOptionsFilter.defaultValue=Default Value
HyperlinkOptionsFilter.showCount=Show Data Count?
HyperlinkOptionsFilter.displayFull=Display in full width (Above other filters)
HyperlinkOptionsFilter.showLabel=Show label?
```

e. Register your plugin to the Felix Framework

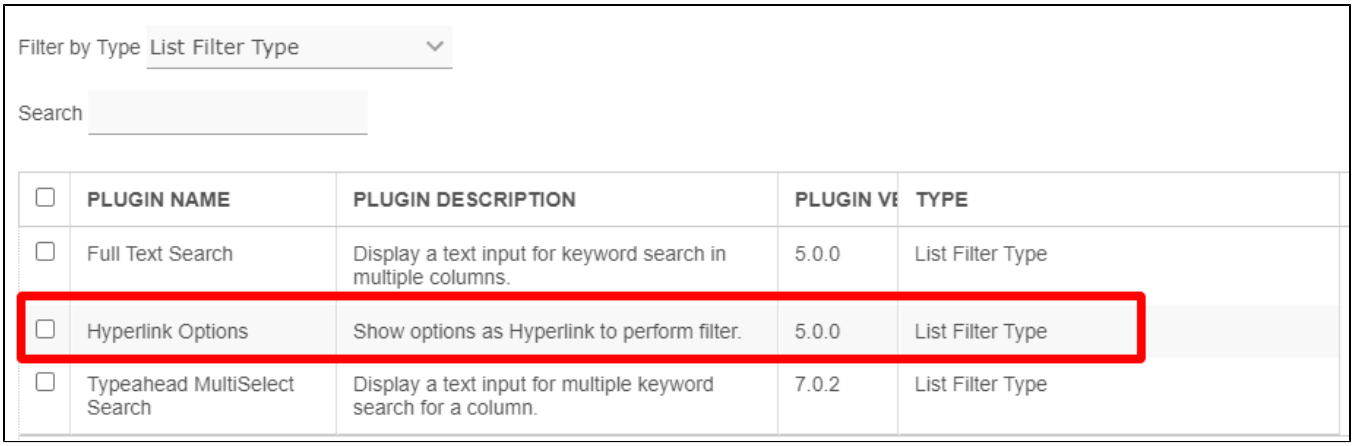
Next, we will have to register our plugin class in the Activator class (Auto generated in the same class package) to tell the Felix Framework that this is a plugin.

```
    public void start(BundleContext context) {
        registrationList = new ArrayList<ServiceRegistration>();
        //Register plugin here
        registrationList.add(context.registerService(HyperlinkOptionsFilter.class.getName(), new
HyperlinkOptionsFilter(), null));
    }
```

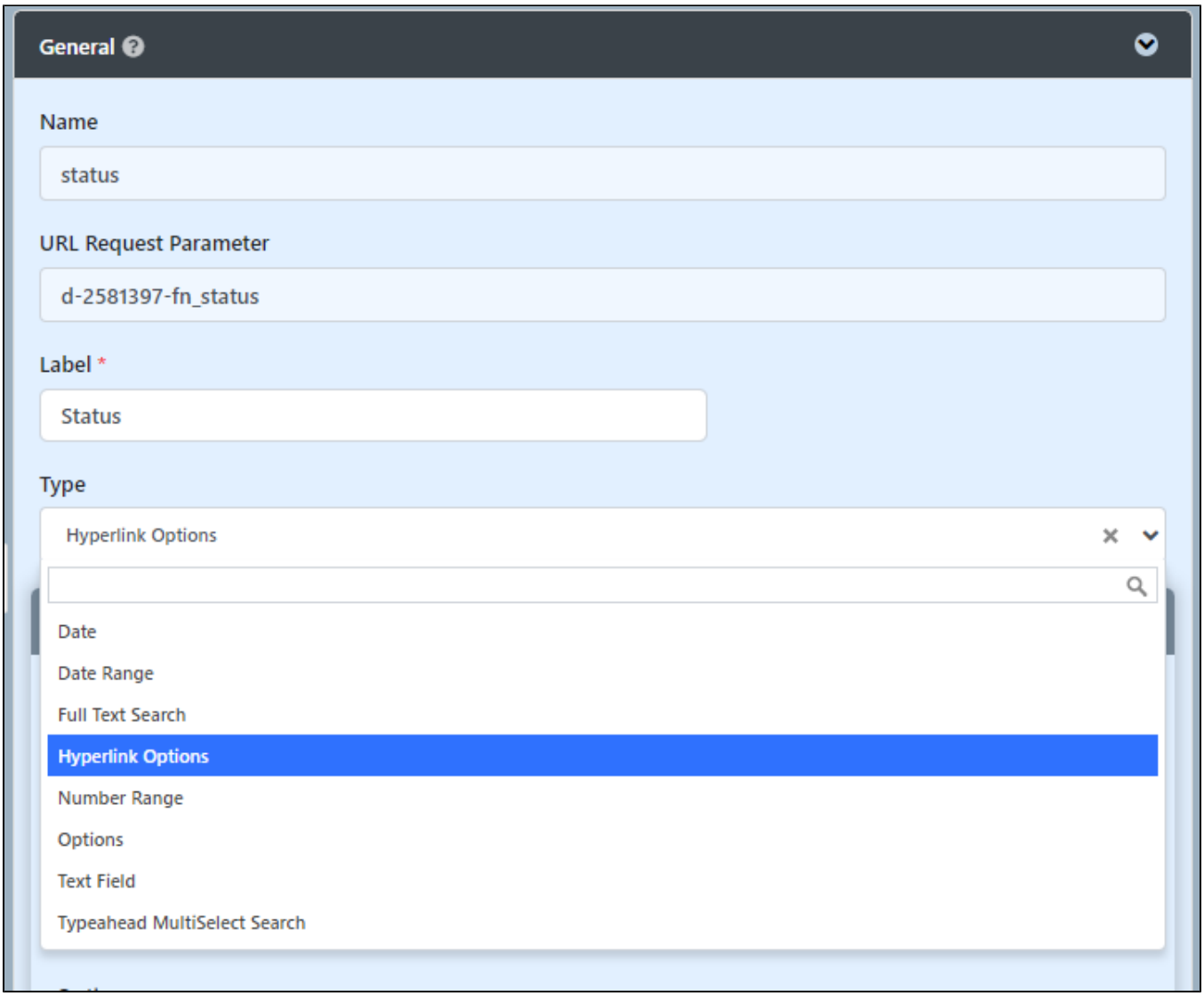
f. Build it and test

Let's build our plugin. Once the building process is done, we will find a "hyperlink_options_filter-5.0.0.jar" file created under "hyperlink_options_filter/target" directory.

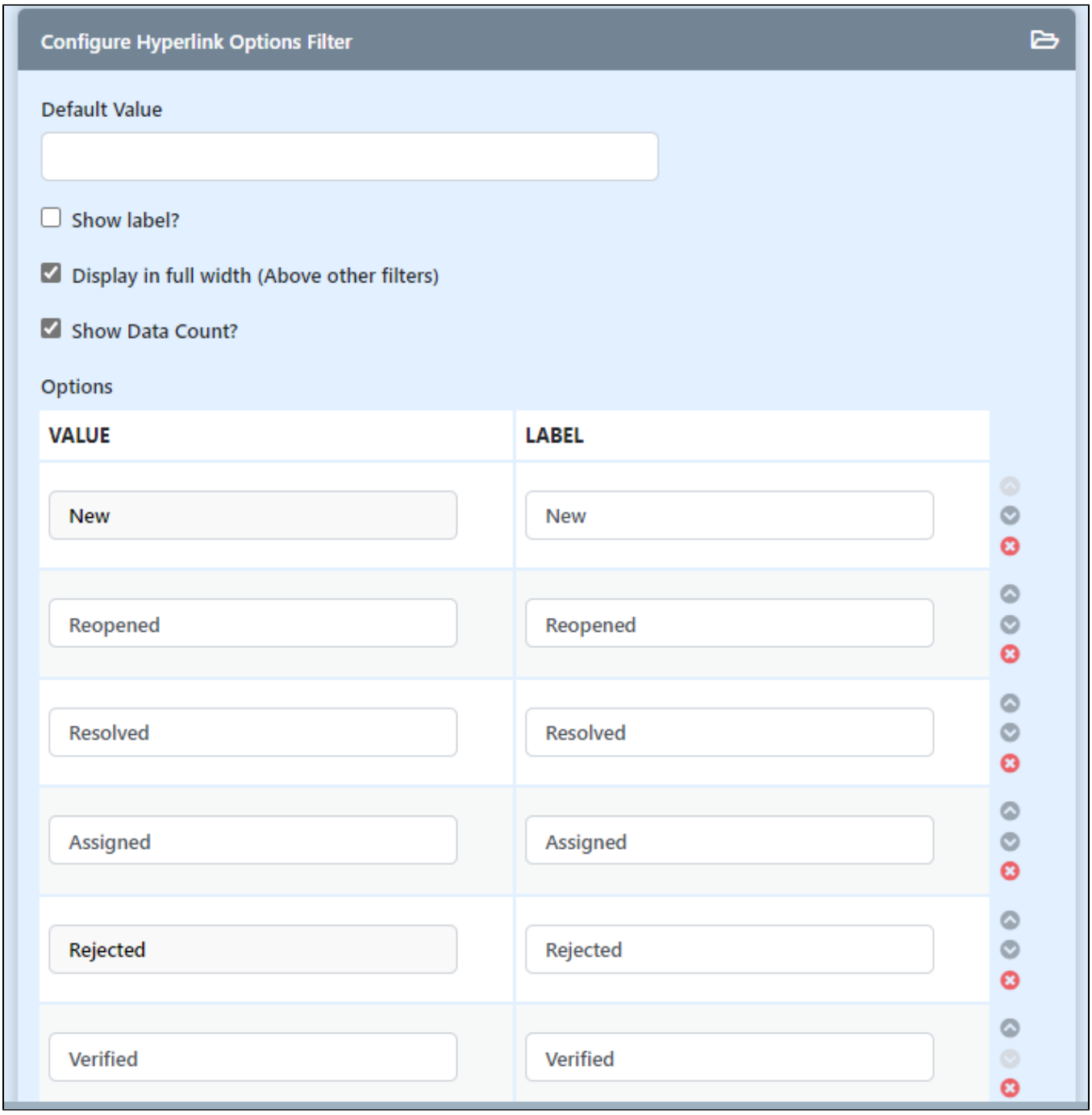
Then, let's upload the plugin jar to Manage Plugins. After uploading the jar file, double check that the plugin is uploaded and activated correctly.



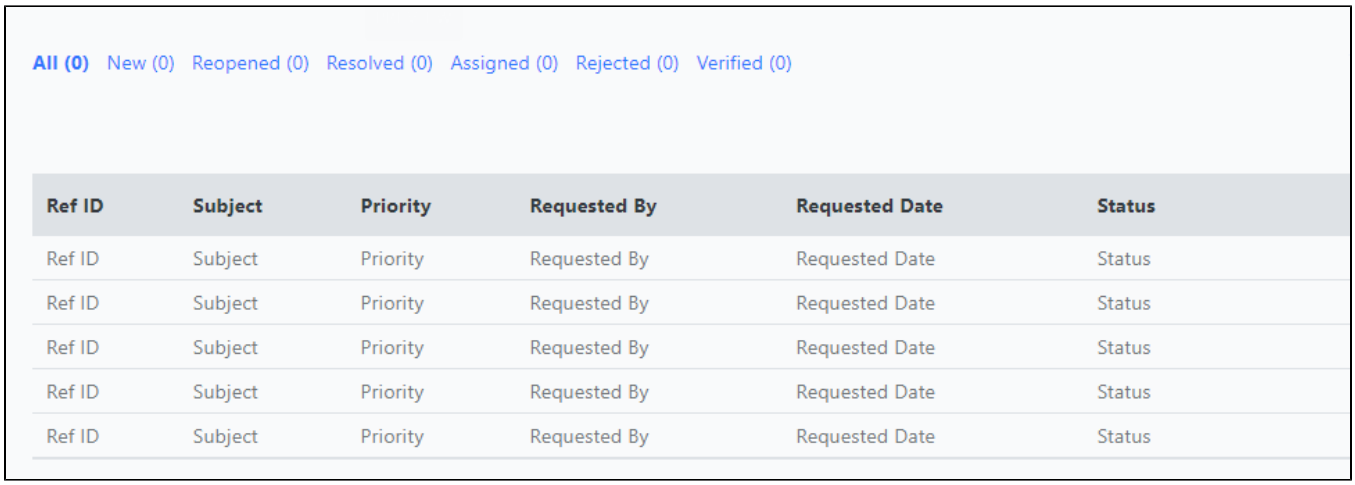
Then, check the Hyperlink Options Filter is shown as a selection of filter type in the List Builder.



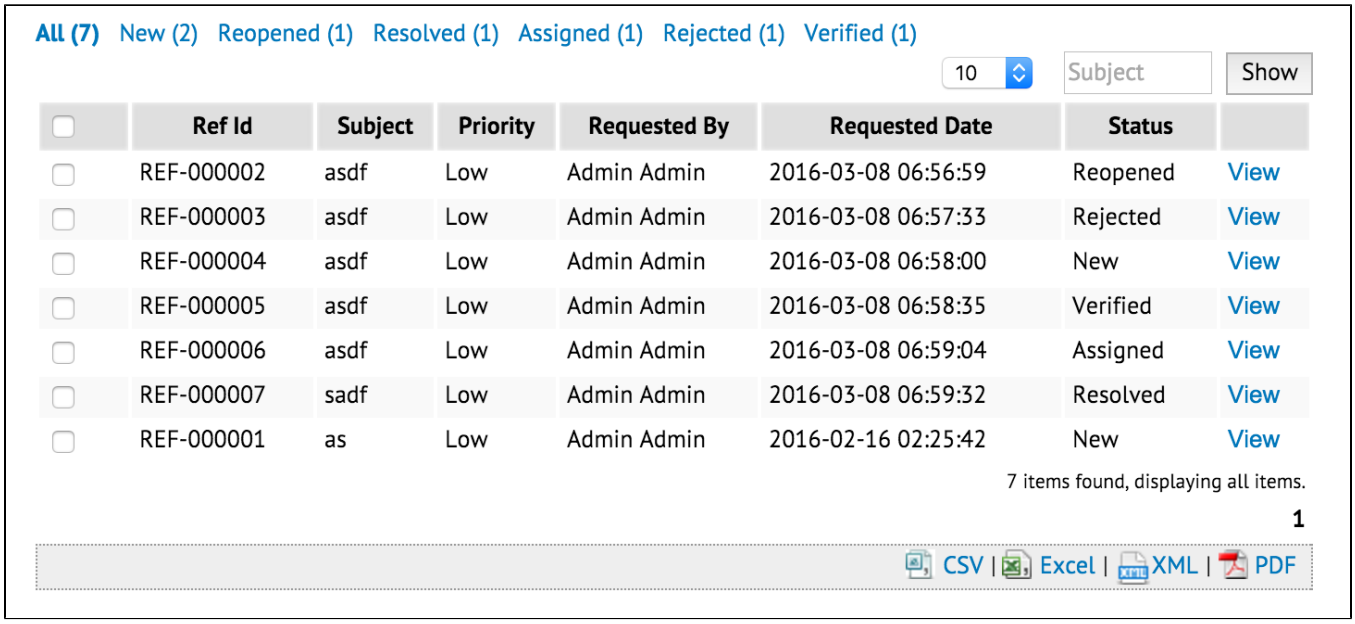
Configure its properties.



Save the properties and check the filter is render in canvas as following.



Check and test the filter in datalist.



8. Take a step further, share it or sell it

You can download the source code from hyperlink_options_filter_src.zip.

To download the ready-to-use plugin jar, please find it in Hyperlink Options Filter Plugin.