How to develop a File Link Datalist Formatter

In this tutorial, we will following the guideline of developing a plugin to develop our File Link Datalist Formatter. 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 have a file uploaded using File Upload field in form. We want to show the file name as a link to download the file in our datalist.

2. What is your idea to solve the problem?

We can develop a Datalist Column Formatter Plugin to convert the value to a link.

3. What is the input needed for your plugin?

The file download link format of Joget Workflow is

```
/jw/web/client/app/{appId}/{appVersion}/form/download/{formId}/{recordId}/{fileName}.?attachment=true
```

Sample:

```
http://localhost:8080/jw/web/client/app/test/1/form/download/testFile/f6946830-c0a80070-48cad9b7-8b971682/CHANGES.txt.?attachment=true
```

From the link format, we need to get the following information from admin user.

1. Form Id : the form that contains the File Upload field.
2. Download as Attachment: Whether or not to add "?attachedment=true" in link.

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

The file name is displayed as a link in Datalist.

5. Is there any resources/API that can be reuse?

We can refer to the File Upload Field on how to generate the download link of the file.

6. Prepare your development environment

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

The following of this tutorial is prepared with a Macbook Pro and Joget Source Code version 5.0.0. Please refer to Guideline for developing a

plugin for other platform command.

Let said our folder directory as following.

```
- Home
  - joget
    - plugins
    - jw-community
      -5.0.0
```

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 stored.

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

```
cd joget/plugins/
~/joget/jw-community/5.0.0/wflow-plugin-archetype/create-plugin.sh org.joget.tutorial
file_link_datalist_formatter 5.0-SNAPSHOT
```

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

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

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

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

7. Just code it!

a. Extending the abstract class of a plugin type

Create a "FileLinkDatalistFormatter" class under "org.joget.tutorial" package. Then, extend the class with **org.joget.apps.datalist.model.DataListColumnFormatDefault** abstract class. Please refer to Datalist Column Formatter 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 Expand source

```
package org.joget.tutorial;
import org.joget.apps.app.service.AppPluginUtil;
import org.joget.apps.app.service.AppUtil;
import org.joget.apps.datalist.model.DataList;
import org.joget.apps.datalist.model.DataListColumn;
import org.joget.apps.datalist.model.DataListColumnFormatDefault;

public class FileLinkDatalistFormatter extends DataListColumnFormatDefault {

    private final static String MESSAGE_PATH = "messages/FileLinkDatalistFormatter";

    public String getName() {
        return "File Link Datalist Formatter";
    }

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

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

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

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

    public String getPropertyOptions() {
        return AppUtil.readPluginResource(getClassName(),
"/properties/fileLinkDatalistFormatter.json", null, true, MESSAGE_PATH);
    }

    public String format(DataList dataList, DataListColumn column, Object row, Object
value) {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

Then, we have to do a UI for admin user to provide inputs for our plugin. In getPropertyOptions method, we already specify our Plugin Properties Options definition file is locate at "/properties/fileLinkDatalistFormatter.json". Let us create a directory "resources/properties" under "file_link_datalist_formatter/src/main" directory. After create the directory, create a file named "fileLinkDatalistFormatter.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 : '@@datalist.fileLinkFormatter.config@@',
    properties : [{
        name : 'formDefId',
        label : '@@datalist.fileLinkFormatter.form@@',
        type : 'selectbox',
        options_ajax : '[CONTEXT_PATH]/web/json/console/app[APP_PATH]/forms/options',
        required : 'True'
    },
    {
        name : 'attachment',
        label : '@@datalist.fileLinkFormatter.attachment@@',
        type : 'checkbox',
        options : [{
            value : 'true',
            label : ''
        }]
    }]
}]
```

Once we done the properties option to collect input, we can work on the main method of the plugin which is format method.

```
    public String format(DataList dataList, DataListColumn column, Object row, Object
value) {
        String result = (String) value;

        if (result != null && !result.isEmpty()) {
            try {
                String formDefId = getPropertyString("formDefId");
                AppDefinition appDef = AppUtil.getCurrentAppDefinition();
                result = "";

                String attachment = "";
                if ("true".equals(getPropertyString("attachment"))) {
                    attachment = "?attachment=true";
                }

                //get the id of this record
                String primaryKeyValue = (String) LookupUtil.getBeanProperty(row,
dataList.getBinder().getPrimaryKeyColumnName());

                HttpServletRequest request = WorkflowUtil.getHttpServletRequest();

                //suport for multi values
                for (String v : value.toString().split(";")) {
                    if (!v.isEmpty()) {
                        // determine actual path for the file uploads
                        String fileName = v;
                        String encodedFileName = fileName;

                        try {
                            encodedFileName = URLEncoder.encode(fileName,
"UTF8").replaceAll("\\+", "%20");
                        } catch (UnsupportedEncodingException ex) {
                            // ignore
                        }

                        String filePath = request.getContextPath() +
"/web/client/app/" + appDef.getAppId() + "/" + appDef.getVersion().toString() +
"/form/download/" + formDefId + "/" + primaryKeyValue + "/" + encodedFileName + "." +
attachment;

                        if (!result.isEmpty()) {
                            result += ", ";
                        }
                        result += "<a href=\""+filePath+"\"
target=\"_blank\">"+StringUtil.stripAllHtmlTag(fileName)+"</a>";
                    }
                }
            } catch (Exception e) {
                LogUtil.error(getClassName(), e, "");
            }
        }
        return result;
    }
```

c. Manage the dependency libraries of your plugin

Our plugin is using org.displaytag.util.LookupUtil and class, it required displaytag and jsp-api library. So, we have to add it to our POM file.

```
<!-- Change plugin specific dependencies here -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jsp-api</artifactId>
            <version>2.0</version>
        </dependency>
        <dependency>
            <groupId>displaytag</groupId>
            <artifactId>displaytag</artifactId>
            <version>1.2</version>
            <exclusions>
                <exclusion>
                    <artifactId>slf4j-api</artifactId>
                    <groupId>org.slf4j</groupId>
                </exclusion>
                <exclusion>
                    <artifactId>jcl104-over-slf4j</artifactId>
                    <groupId>org.slf4j</groupId>
                </exclusion>
                <exclusion>
                    <groupId>org.slf4j</groupId>
                    <artifactId>slf4j-log4j12</artifactId>
                </exclusion>
            </exclusions>
        </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 also used i18n message key in our properties options definition as well. So, we will need to create a message resource bundle properties file for our plugin.

Create directory "resources/messages" under "file_link_datalist_formatter/src/main" directory. Then, create a "FileLinkDatalistFormatter.properties" file in the folder. In the properties file, let add all the message keys and its label as below.

```
org.joget.tutorial.FileLinkDatalistFormatter.pluginLabel=File Link Datalist Formatter
org.joget.tutorial.FileLinkDatalistFormatter.pluginDesc=To format the column value as
attachment download link.
datalist.fileLinkFormatter.config=Configure File Link Formatter
datalist.fileLinkFormatter.form=Form
datalist.fileLinkFormatter.attachment=Download as Attachment?
```

e. Register your plugin to Felix Framework

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

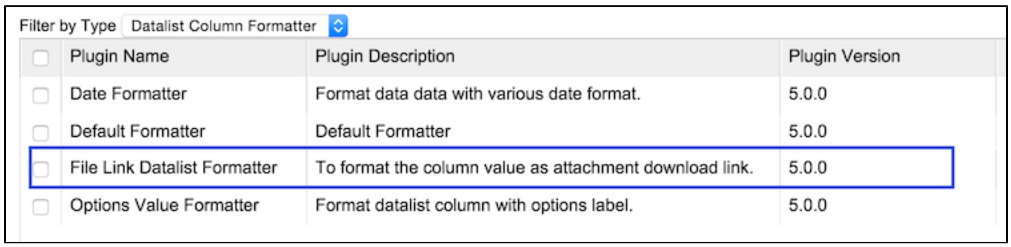
```
public void start(BundleContext context) {
        registrationList = new ArrayList<ServiceRegistration>();
        //Register plugin here

registrationList.add(context.registerService(FileLinkDatalistFormatter.class.getName()
, new FileLinkDatalistFormatter(), null));
    }
```

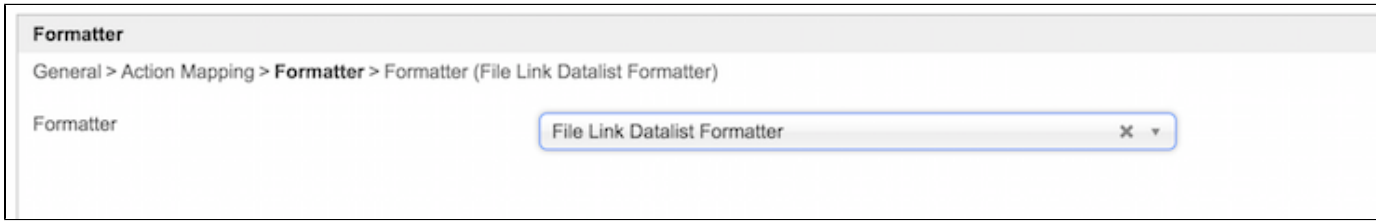
f. Build it and testing

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

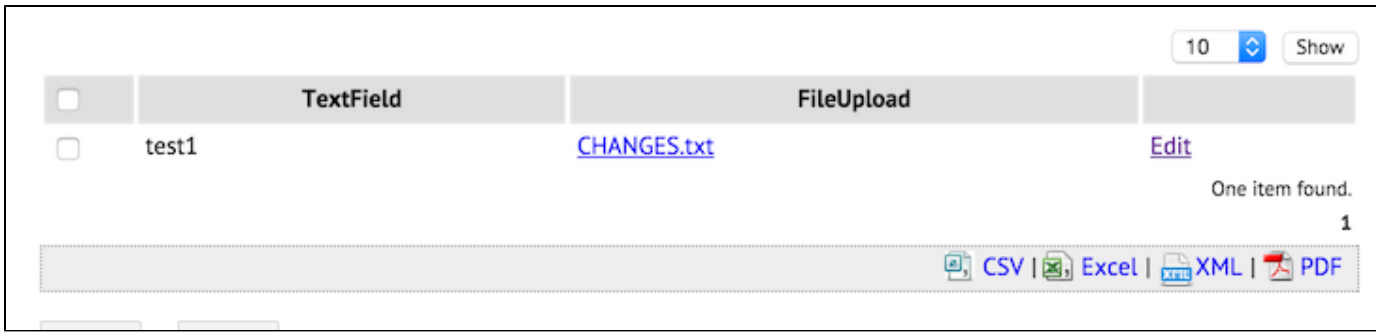
Then, let upload the plugin jar to Manage Plugins. After upload the jar file, double check the plugin is uploaded and activated correctly.



Let create a CRUD in userview to test our plugin. In list, let configure the File Upload field column as following.



In the CRUD list, we can see our file name is converted to a link.



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

You can download the source code from file_link_datalist_formatter.zip.

To download the ready-to-use plugin jar, please find it in http://marketplace.joget.org/.