

# How to develop a Download PDF Datalist Action

- 1. What is the problem?
- 2. How to solve the problem?
- 3. What is the input needed for your plugin?
- 4. What is the output and expected outcome of your plugin?
- 5. Are there any resources/API that can be reused?
- 6. Prepare your development environment
- 7. Just code it!
  - a. Extending the abstract class of a plugin type
  - b. Implement all the abstract methods
  - c. Manage the dependency libraries of your plugin
  - d. Make your plugin internationalization (i18n) ready
  - e. Register your plugin to the Felix Framework
  - f. Build it and test
- 8. Take a step further, share it or sell it

In this tutorial, we will follow the [guideline for developing a plugin](#) to develop our Download PDF Datalist Action 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 require the ability to download form data as a PDF file from the datalist.

## 2. How to solve the problem?

We will develop a [Datalist Action Plugin](#) to display a button to generate a form PDF file.

## 3. What is the input needed for your plugin?

To develop a PDF Download Datalist Action plugin, we will consider providing the following as input.

1. Form ID : The form that will be used to generate the PDF file.
2. Record ID Column : Use the id of the datalist row or a column value to load the record.
3. Formatting options : Options to format and customise the PDF output.

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

When the PDF Download Datalist Action is used as a datalist row action or column action, a normal user will see a link to download the PDF file in every row of the datalist. Once the link is clicked, a PDF file will be prompted for download for that specific row.

When the plugin is used for multiple datalist rows (whole list action), a zip file containing all the generated PDFs of every selected rows will be prompted for download when the button is clicked.

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

To develop the PDF Download Datalist Action plugin, we can reuse the methods in [FormPdfUtil](#) to generate a form as PDF. We can also refer to the [source code](#) of the Datalist Form Data Delete Action plugin as well. Other than that, we can refer to the [Export Form Email Tool](#) on what kind of plugin properties options we can provide in the plugin as the Export Form Email Tool are using the methods in FormPdfUtil as well.

## 6. Prepare your development environment

We need to always have our Joget Workflow Source Code ready and build by following [this guideline](#).

The following tutorial is prepared with a Macbook Pro and the Joget Source Code is version 5.0.0. 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
    - 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 is 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  
download_pdf_datalist_action 5.0-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: : 5.0.0  
[INFO] Using property: package = org.joget.tutorial  
Confirm properties configuration:  
groupId: org.joget.tutorial  
artifactId: download_pdf_datalist_action  
version: 5.0.0  
package: org.joget.tutorial  
Y: : y
```

We should get a "BUILD SUCCESS" message shown in our terminal and a "download\_pdf\_datalist\_action" 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 "DownloadPdfDatalistAction" class under "org.joget.tutorial" package. Then, extend the class with [org.joget.apps.datalist.model.DataListActionDefault](#) abstract class. Please refer to [Datalist Action Plugin](#).

#### b. Implement all the abstract methods

As usual, we have to implement all the abstract methods. We will use 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.tutorial;  
  
import org.joget.apps.app.service.AppPluginUtil;  
import org.joget.apps.datalist.model.DataListActionDefault;  
  
public class DownloadPdfDatalistAction extends DataListActionDefault {  
  
    private final static String MESSAGE_PATH = "messages/DownloadPdfDatalistAction";  
  
    public String getName() {  
        return "Download PDF Datalist Action";  
    }  
    public String getVersion() {  
        return "5.0.0";  
    }  
  
    public String getClassName() {  
        return getClass().getName();  
    }  
  
    public String getLabel() {  
        //support i18n  
        return AppPluginUtil.getMessage("org.joget.tutorial.DownloadPdfDatalistAction.pluginLabel",  
getClassName(), MESSAGE_PATH);  
    }  
  
    public String getDescription() {  
        //support i18n  
        return AppPluginUtil.getMessage("org.joget.tutorial.DownloadPdfDatalistAction.pluginDesc",  
getClassName(), MESSAGE_PATH);  
    }  
}
```

```

    }

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

    public String getLinkLabel() {
        return getPropertyString("label"); //get label from configured properties options
    }

    public String getHref() {
        return getPropertyString("href"); //Let system to handle to post to the same page
    }

    public String getTarget() {
        return "post";
    }

    public String getHrefParam() {
        return getPropertyString("hrefParam"); //Let system to set the parameter to the checkbox name
    }

    public String getHrefColumn() {
        String recordIdColumn = getPropertyString("recordIdColumn"); //get column id from configured properties
options
        if ("id".equalsIgnoreCase(recordIdColumn) || recordIdColumn.isEmpty()) {
            return getPropertyString("hrefColumn"); //Let system to set the primary key column of the binder
        } else {
            return recordIdColumn;
        }
    }

    public String getConfirmation() {
        return getPropertyString("confirmation"); //get confirmation from configured properties options
    }

    public DataListActionResult executeAction(DataList dataList, String[] rowKeys) {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}

```

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/downloadPdfDatalistAction.json`. Let us create a directory `resources/properties` under `download_pdf_datalist_action/src/main` directory. After creating the directory, create a file named `downloadPdfDatalistAction.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.downloadPdf.config@@',
    properties : [
        {
            name : 'label',
            label : '@@datalist.downloadPdf.label@@',
            type : 'textfield',
            value : '@@datalist.downloadPdf.download@@'
        },
        {
            name : 'formDefId',
            label : '@@datalist.downloadPdf.form@@',
            type : 'selectbox',
            options_ajax : '[CONTEXT_PATH]/web/json/console/app[APP_PATH]/forms/options',
            required : 'True'
        },
        {
            name : 'recordIdColumn',
            label : '@@datalist.downloadPdf.recordIdColumn@@',
            description : '@@datalist.downloadPdf.recordIdColumn.desc@@',
            type : 'textfield'
        },
        {
            name : 'confirmation',

```



```

    }

    // check for submitted rows
    if (rowKeys != null && rowKeys.length > 0) {
        try {
            //get the HTTP Response
            HttpServletResponse response = WorkflowUtil.getHttpServletResponse();
            if (rowKeys.length == 1) {
                //generate a pdf for download
                singlePdf(request, response, rowKeys[0]);
            } else {
                //generate a zip of all pdfs
                multiplePdfs(request, response, rowKeys);
            }
        } catch (Exception e) {
            LogUtil.error(getClassName(), e, "Fail to generate PDF for " + ArrayUtils.toString(rowKeys));
        }
    }

    //return null to do nothing
    return null;
}

/**
 * Handles for single pdf file
 * @param request
 * @param response
 * @param rowKey
 * @throws IOException
 * @throws javax.servlet.ServletException
 */
protected void singlePdf(HttpServletRequest request, HttpServletResponse response, String rowKey) throws
IOException, ServletException {
    byte[] pdf = getPdf(rowKey);
    writeResponse(request, response, pdf, rowKey+".pdf", "application/pdf");
}

/**
 * Handles for multiple files download. Put all pdfs in zip.
 * @param request
 * @param response
 * @param rowKeys
 * @throws java.io.IOException
 * @throws javax.servlet.ServletException
 */
protected void multiplePdfs(HttpServletRequest request, HttpServletResponse response, String[] rowKeys)
throws IOException, ServletException {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    ZipOutputStream zip = new ZipOutputStream(baos);

    try {
        //create pdf and put in zip
        for (String id : rowKeys) {
            byte[] pdf = getPdf(id);
            zip.putNextEntry(new ZipEntry(id+".pdf"));
            zip.write(pdf);
            zip.closeEntry();
        }

        zip.finish();
        writeResponse(request, response, baos.toByteArray(), getLinkLabel() + ".zip", "application/zip");
    } finally {
        baos.close();
        zip.flush();
    }
}

/**
 * Generate PDF using FormPdfUtil
 * @param id
 * @return

```

```

    */
protected byte[] getPdf(String id) {
    AppDefinition appDef = AppUtil.getCurrentAppDefinition();
    String formDefId = getPropertyString("formDefId");

    Boolean hideEmptyValueField = null;
    if (getPropertyString("hideEmptyValueField").equals("true")) {
        hideEmptyValueField = true;
    }
    Boolean showNotSelectedOptions = null;
    if (getPropertyString("showNotSelectedOptions").equals("true")) {
        showNotSelectedOptions = true;
    }
    Boolean repeatHeader = null;
    if ("true".equals(getPropertyString("repeatHeader"))) {
        repeatHeader = true;
    }
    Boolean repeatFooter = null;
    if ("true".equals(getPropertyString("repeatFooter"))) {
        repeatFooter = true;
    }
    String css = null;
    if (!getPropertyString("formatting").isEmpty()) {
        css = getPropertyString("formatting");
    }
    String header = null;
    if (!getPropertyString("headerHtml").isEmpty()) {
        header = getPropertyString("headerHtml");
        header = AppUtil.processHashVariable(header, null, null, null);
    }
    String footer = null;
    if (!getPropertyString("footerHtml").isEmpty()) {
        footer = getPropertyString("footerHtml");
        footer = AppUtil.processHashVariable(footer, null, null, null);
    }

    return FormPdfUtil.createPdf(formDefId, id, appDef, null, hideEmptyValueField, header, footer, css,
showNotSelectedOptions, repeatHeader, repeatFooter);
}

/**
 * Write to response for download
 * @param response
 * @param bytes
 * @param filename
 * @param contentType
 * @throws IOException
 */
protected void writeResponse(HttpServletRequest request, HttpServletResponse response, byte[] bytes, String
filename, String contentType) throws IOException, ServletException {
    OutputStream out = response.getOutputStream();
    try {
        String name = URLEncoder.encode(filename, "UTF8").replaceAll("\\\\+", "%20");
        response.setHeader("Content-Disposition", "attachment; filename="+name+"; filename*=UTF-8'" +
name);

        response.setContentType(contentType+"; charset=UTF-8");

        if (bytes.length > 0) {
            response.setContentLength(bytes.length);
            out.write(bytes);
        }
    } finally {
        out.flush();
        out.close();

        //simply foward to a
        request.getRequestDispatcher(filename).forward(request, response);
    }
}

```

c. Manage the dependency libraries of your plugin

Our plugin is using javax.servlet.http.HttpServletRequest and javax.servlet.http.HttpServletResponse class, so we will need to add jsp-api library to our POM file.

```
<!-- Change plugin specific dependencies here -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jsp-api</artifactId>
  <version>2.0</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/messages", under "download\_pdf\_datalist\_action/src/main" directory. Then, create a "DownloadPdfDatalistAction.properties" file in the folder. In the properties file, add all the message keys and its label as below.

```
org.joget.tutorial.DownloadPdfDatalistAction.pluginLabel=Download PDF
org.joget.tutorial.DownloadPdfDatalistAction.pluginDesc=Support to download form PDF from datalist
datalist.downloadPdf.download=Download
datalist.downloadPdf.config=Configure Download PDF Action
datalist.downloadPdf.label=Label
datalist.downloadPdf.form=Form
datalist.downloadPdf.recordIdColumn=Record Id Column
datalist.downloadPdf.recordIdColumn.desc=Default to the primary key of the configured binder
datalist.downloadPdf.confirmationMessage=Confirmation Message
datalist.downloadPdf.hideEmptyValueField=Hide field that without value
datalist.downloadPdf.showNotSelectedOptions=Show unselected options for multi options field
datalist.downloadPdf.advanced=Advanced
datalist.downloadPdf.formatting=Formatting (CSS)
datalist.downloadPdf.headerHtml=Header (HTML)
datalist.downloadPdf.repeatHeader=Repeat header on every page?
datalist.downloadPdf.footerHtml=Footer (HTML)
datalist.downloadPdf.repeatFooter=Repeat footer on every page?
```

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(DownloadPdfDatalistAction.class.getName(), new
DownloadPdfDatalistAction(), null));
}
```

f. Build it and test

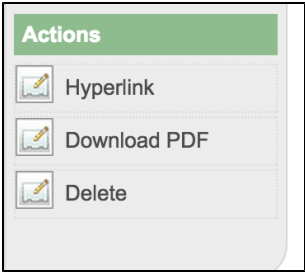
Let's build our plugin. Once the building process is done, we will find a "download\_pdf\_datalist\_action-5.0.0.jar" file created under "download\_pdf\_datalist\_action/target" directory.

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.

Filter by Type Datalist Action

<input type="checkbox"/>	Plugin Name	Plugin Description	Plugin Version	
<input type="checkbox"/>	Hyperlink	Data List Hyperlink Action	5.0.0	
<input type="checkbox"/>	Download PDF	Support to download form	5.0.0	
<input type="checkbox"/>	Delete	Form Row Delete Action	5.0.0	

Then, let's try it in one of the datalist. You can see our new plugin shown under "Actions" in [Datalist Builder](#).



Once we drag and drop the "Download PDF" action into the datalist builder canvas, we can edit the action. The following configuration page will be shown based on our properties option definition.

Property Editor close

**Configure Download PDF Action**

Configure Download PDF Action > Advanced > Visibility Control

Label

Form

Record Id Column

Confirmation Message

< Prev Next > OK Cancel

Property Editor close

**Advanced**

Configure Download PDF Action > Advanced > Visibility Control

Formatting (CSS)

1	
---	--

Header (HTML)

1	
---	--

Repeat header on every page? ☐

Footer (HTML)

1	
---	--

< Prev Next > OK Cancel

Let's add the "Download PDF" action as row action and also the whole list action for testing. We can see the "Download" button shown correctly in the userview screenshot below.



10

Account Nar

Show

	ID	Account Name	Country	State			
<input type="checkbox"/>	Jack	jack	local	Test	Download	Contacts	New Contact
<input type="checkbox"/>	owen	Owen Ong	local	Test	Download	Contacts	New Contact

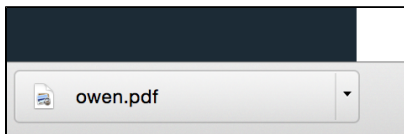
2 items found, displaying all items.

1

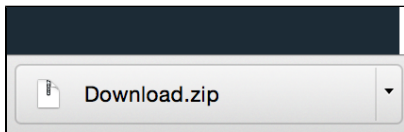
CSV | Excel | XML | PDF

DeleteDownload

When row action is clicked, a pdf is downloaded.



When the whole list action is clicked, a zip file is downloaded.



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

You can download the source code from [download\\_pdf\\_datalist\\_action.zip](#).

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