Guideline for developing a plugin

There following questions and steps are to help you plan and develop a plugin to fulfill a custom requirement.

1. What is the problem?

You have a custom requirement and you found that none of the built-in plugins provided by Joget Workflow nor plugins available in Marketplace are able to fulfill your requirement.

Example 1: Download a PDF version of a form when click on a button in a list.

Example 2: Provide a Gantt Chart view of your collected form data.

Example 3: Hash variable is convenient in use, but it does not provide the ability to do condition checking.

2. How to solve the problem?

Refer to the Plugin Types that are supported by Joget Workflow, find the most appropriate plugin type that can help you to fulfill the custom requirement.

Example 1: Develop a Datalist Action plugin to display a button for generate form PDF.

Example 2: Develop a Userview Menu plugin that can be used to display form data as Gantt Chart.

Example 3: Develop a Hash Variable plugin that can do Bean Shell scripting.

3. What is the input needed for your plugin?

Find out what information are needed by your plugin to function/work. Look at it from the user's perspective; how are you going to use the plugin. Then, look at it from a developer's perspective; to make the plugin reusable in more use cases.

You can refer to Plugin Properties Options on what type of input field you can provide to your plugin user.

Example 1: To develop a PDF Download Datalist Action plugin, we can consider providing the following as input.

a. Form ID : The form that will be used to generate the PDF file.
b. Record ID : Use the id of the datalist row or a column value to load the record.
c. File Name : File name of the the generated PDF file.
d. Formatting options : Options to format and customise the PDF output.

Example 2: To develop an Gantt Chat Userview Menu plugin, we can consider providing the following as input.

a. Datalist Binder : We can reuse datalist binder in our plugin to retrieve the data needed by the Gantt Chart.
b. Mapping : A field to map the columns that will be returned from the datalist binder to the data needed by the Gantt Chart.
c. Styling : Options to style the Gantt Chart.

Example 3: Hash Variable plugin does not provide interface for user to configure, but to develop a Bean Shell Hash Variable plugin, we need somewhere to put our Bean Shell script. We can reuse the Environment Variable to store our scripts. So the Hash Variable syntax will be a prefix with environment variable key.

E.g. #beanshell.EnvironmentVariableKey#

But, this may not be enough. We may need some other way to pass in some variable too. We can consider using a URL query parameters syntax to pass in our variables because it is easier to parse later on.

E.g. #beanshell.EnvironmentVariableKey[name=Joget&email=info@joget.org&message={form.sample.message?url}]#

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

How and what a normal user (Not the admin user who use the plugin to provide functionality) will use and see your plugin result.

Example 1: When 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 rows of a datalist. Once the link is clicked, a PDF will be prompted to be downloaded.

When the plugin is used as a whole datalist action, a zip file containing all the generated PDF of every selected rows will be prompted to be downloaded.

Example 2: A normal user can see a Gantt Chart when a menu using the Gantt Chart Userview Menu plugin is clicked. User may navigate or interact with the Gantt Chart.

Example 3: The Bean Shell Hash Variable plugin is for the admin user. Once it is used, the Hash Variable will be replaced by the output return from the Bean Shell interpreter.

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

Always refer to existing plugins and tutorials to look for a similar plugin/plugin type that you can refer to whenever possible. The Joget team will try their best to enrich the contents in the tutorials section.

You may need to check out the document of Utility & Service Methods, JSON API, Javascript API and Bean Shell Programming Guide as well. These documents may contains some methods/examples that may help you in the development.

Example 1: 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.

Example 2: To develop a Gantt Chart Userview Menu plugin, we can refer to the source code of all the Userview Menu plugin. From there, we can have a better understanding on how to make a template for a plugin using FreeMaker syntax.

Example 3: To develop Bean Shell Hash Variable plugin, we can refer to the source code of all the Hash Variable plugin and Bean Shell plugin. Especially, we can refer to Environment Variable Hash Variable plugin on how to retrieve environment variable using a variable key. We can also refer to Bean Shell Tool or Bean Shell Form Binder plugin on what to execute the script with Bean Shell interpreter.

6. Prepare your development environment

a. You will need to have the Joget Workflow Open Source ready and built. We will use the "wflow-plugin-archetype" module to generate a maven project for our plugin.

b. Generate a maven project.

The project file generated will be placed at the directory specified in the command prompt, eg: running the command in "C:\" will place the project file in the root of C drive.

To get the correct jogetDependencyVersion in order to build a proper project file with correct dependencies, check the file name specified in the ".m2" folder under directory of, eg:

C:\Users\(your computer name)\.m2\repository\org\joget\wflow-core**"5.0.11"**

- Run the following for Window

```
"C:\Joget Source Code\wflow-plugin-archetype\create-plugin.bat" packageName pluginFolderName
jogetDependencyVersion
```

- Run the following for Linux or Mac

```
/joget_src/wflow-plugin-archetype/create-plugin.sh packageName pluginFolderName jogetDependencyVersion
```

- Sample Screenshot in Mac:

```
Owens-MacBook-Pro:plugins owen$ ~/joget/jw-community/branches/5.0-SNAPSHOT/wflow-plugin-archetype/create-plugin.sh org.joget.sample sample-plugins 5.

[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:2.1:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:2.1:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:2.1:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Interactive mode
[WARNING] Archetype not found in any catalog. Falling back to central repository (http://repo1.maven.org/maven2).
[WARNING] Use -DarchetypeRepository=<your repository> if archetype's repository is elsewhere.
Downloading: http://dev.joget.org/archiva/repository/snapshots/org/joget/wflow-plugin-archetype/5.0-SNAPSHOT/maven-metadata.xml
Downloading: http://repo1.maven.org/maven2/org/joget/wflow-plugin-archetype/5.0-SNAPSHOT/maven-metadata.xml
[INFO] Using property: groupId = org.joget.sample
[INFO] Using property: artifactId = sample-plugins
Define value for property 'version':  1.0-SNAPSHOT: : 1.0.0
[INFO] Using property: package = org.joget.sample
Confirm properties configuration:
groupId: org.joget.sample
artifactId: sample-plugins
version: 1.0.0
package: org.joget.sample
 Y: : y
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: wflow-plugin-archetype:5.0-SNAPSHOT
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: org.joget.sample
[INFO] Parameter: packageName, Value: org.joget.sample
[INFO] Parameter: package, Value: org.joget.sample
[INFO] Parameter: artifactId, Value: sample-plugins
[INFO] Parameter: basedir, Value: /Users/owen/joget/plugins
[INFO] Parameter: version, Value: 1.0.0
[INFO] project created from Old (1.x) Archetype in dir: /Users/owen/joget/plugins/sample-plugins
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 51.623 s
[INFO] Finished at: 2015-09-08T23:38:32+08:00
[INFO] Final Memory: 13M/245M
[INFO] ------------------------------------------------------------------------
Owens-MacBook-Pro:plugins owen$
```

c. Open/Import the maven project with your favorite IDE. Joget team recommends NetBeans IDE.

7. Just code it!

a. Extending the abstract class of a plugin type

Refer to the document of the plugin type listed in Plugin Types. Find the abstract class and interface that need to be extended and implemented by your plugin.

Example: To develop a Userview Menu plugin, the plugin class need to extends the **org.joget.apps.userview.model.UserviewMenu** abstract class.

b. Implement all the abstract methods

A plugin will have to implements the abstract method of Plugin Base Abstract Class and Interface and also the abstract method of the individual abstract class and interface for the plugin type.

Example: To develop a Userview Menu plugin, the following methods have to be implemented by the plugin. Please refer to the plugin documents for the details of each methods.

- getCategory
- getClassName
- getDecoratedMenu
- getDescription
- getIcon
- getLabel
- getName
- getPropertyOptions
- getRenderPage
- getVersion
- isHomePageSupported

c. Manage the dependency libraries of your plugin

The generated plugin folder by "wflow-plugin-archetype" module is a maven project. So, we will using the Dependency Mechanism provided by Maven.

d. Make your plugin internationalization (i18n) ready

To make the plugin i18n ready, we need to create a message resource bundle property file for the plugin.

- Create a property file with the plugin class name in "[Plugin project directory]/src/main/resources/message" directory.

  Example: For a plugin named "GanttChartMenu", we need to create a "GanttChartMenu.properties" file under "[Plugin project directory]/src/main/resources/message" directory.

  Sample content for GanttChartMenu.properties file

```
org.joget.sample.GanttChartMenu.pluginLabel=Gantt Chart
org.joget.sample.GanttChartMenu.pluginDesc=To display form data in Gantt Chart layout
userview.ganttChart.label.title=Title
userview.ganttChart.label.week=Week
```

- Use getMessage(String key, String pluginName, String translationPath) of PluginManager or AppPluginUtil to retrieve i18n label.

  Example: Use the getMessage method in getLabel and getDescription methods to return i18n label and description.

```
public String getLabel() {
    return AppPluginUtil.getMessage("org.joget.sample.GanttChartMenu.pluginLabel", getClassName(),
"message/GanttChartMenu");
}
public String getDescription() {
    return AppPluginUtil.getMessage("org.joget.sample.GanttChartMenu.pluginDesc", getClassName(),
"message/GanttChartMenu");
}
```

- Pass a translation file path to readPluginResource(String pluginName, String resourceUrl, Object[] arguments, boolean removeNewLines, String translationFileName) method of AppUtil to provide the plugin properties option with i18n label. We can use "@@message.key@@" in the JSON of Plugin Properties Options.

  Example: For property options of a GanttChartMenu plugin, the following shows the sample code implementation of getPropertyOptions method and the GanttChartMenu.json file

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

```
[{
    title : '@@userview.ganttChart.edit@@',
    properties : [{
        name : 'id',
        label : 'Id',
        type : 'hidden'
    },
    {
        name : 'customId',
        label : '@@userview.ganttChart.customId@@',
        type : 'textfield',
        regex_validation : '^[a-zA-Z0-9_]+$',
        validation_message : '@@userview.ganttChart.invalidId@@'
    },
    {
        name : 'label',
        label : '@@userview.ganttChart.label@@',
        type : 'textfield',
        required : 'True',
        value : '@@userview.ganttChart.label.value@@'
    }]
}]
```

- Pass a translation file path to getPluginFreeMarkerTemplate(Map data, final String pluginName, final String templatePath, String translationPath) method of PluginManager whenever retrieving a HTML template. Once we passed a translation file path, we can use "@@message.key@@" in the freemarker template to retrieve i18n label.

  Example: For getRenderPage method of a GanttChartMenu plugin, the following show the sample code implementation of getRenderPage method and the "GanttChartMenu.ftl" FreeMarker template.

```
    public String getRenderPage() {
        Map model = new HashMap();
        model.put("request", getRequestParameters());
        model.put("element", this);

        PluginManager pluginManager = (PluginManager)AppUtil.getApplicationContext().getBean
("pluginManager");
        String content = pluginManager.getPluginFreeMarkerTemplate(model, getClasstName(), "
/templates/GanttChartMenu.ftl", "message/GanttChartMenu");
        return content;
    }
```

```
<div>
    <h1>@@userview.ganttChart.label.title@@ : ${element.properties.title!}</h1>
</div>
```

- Postfix the plugin class name with an underscore and language code to create a message resource bundle property file for other language.

  Example: GanttChartMenu_zh_CN.properties

e. Register your plugin to the Felix Framework

You will find that a class named "Activator.java" is auto generated in a package of your plugin maven project. The class is used to register your plugin class to the Felix Framework. You do not need to do this if your plugin is not an OSGI Plugin.

In the start method of the activator class, add your plugin class to the "registrationList" variable.

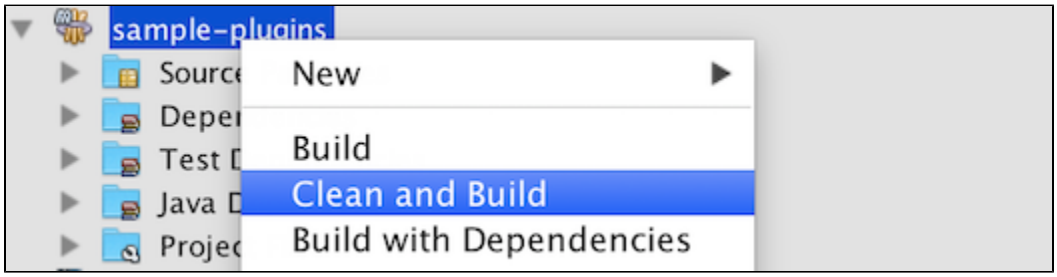
```
public void start(BundleContext context) {
    registrationList = new ArrayList<ServiceRegistration>();

    //Register plugin here
    registrationList.add(context.registerService(MyPlugin.class.getName(), new MyPlugin(), null));
}
```

f. Build it and testing

Once you are done with all the steps above, you can build your project with your IDE using Maven. You can also run "mvn clean install" command in your project directory to build it. After building your project, a jar file is created under "target" folder in your plugin project folder. Upload the plugin jar to Manage Plugins to test your plugin.

Example: In NetBeans, right-click on the project name, then select "Clean and Build".



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

You have completed a very useful plugin. Don't just keep it to yourself, share or sell your plugin in the Joget Marketplace or even better, you can write a tutorial in our Knowledge Base to share your effort with others. To share or sell your plugin, please send an email to info@joget.org.