How to Develop Multi Store Binders

In this tutorial, we will following the guideline of developing a plugin to develop our multi store binders plugin. Please also refer to the very first tutorial How to develop a Bean Shell Hash Variable and also the following store binder related plugin for more detailed steps.

- How to develop a JDBC Form Store Binder

1. What is the problem?

We would like to run multiple store binders plugins in a single store binder selection. This would open up more use cases with the ability to store form data into multiple destinations without the need for relying on other approaches such as JSON Tool or SOAP Tool.

2. What is your idea to solve the problem?

Joget has a plugin type called Form Store Binder Plugin. We will develop one based on this plugin category to support multiple selections and execution of store binders to store form data.

3. What is the input needed for your plugin?

To develop a Multi Store binders, we will need a multi-selector in a grid form and a text editor for additional notes for app designers.

1. Binders: Displays a list of store binders available.
2. Comments: Additional notes users might wish to add.

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

All submitted data will store accordingly based on the configuration of each of the store binders selected.

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

We can refer to the implementation of other available Form Store Binder plugins. We can also refer to how Multi Tools plugin for the design:

https://github.com/jogetworkflow/jw-community/blob/7.0-SNAPSHOT/wflow-core/src/main/java/org/joget/apps/app/lib/MultiTools.java

https://github.com/jogetworkflow/jw-community/blob/7.0-SNAPSHOT/wflow-core/src/main/resources/properties/app/multiTools.json

6. Prepare your development environment

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

The following of this tutorial is prepared with Joget Source Code version 7.0.0. Please refer to Guideline for Developing a Plugin for other platform commands.

Let's use the following folder structure.

```
- Home
  - joget
    - plugins
    - jw-community
      -7.0.0
```

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 the "plugins" directory.

```
cd joget/plugins/
~/joget/jw-community/7.0.0/wflow-plugin-archetype/create-plugin.sh org.joget.marketplace multi_store_binder
7.0.0
```

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: : 7.0.0
[INFO] Using property: package = org.joget.marketplace
Confirm properties configuration:
groupId: org.joget.marketplace
artifactId: multi_store_binder
version: 7.0.0
package: org.joget.marketplace
Y: : y
```

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

Open the maven project with your favorite IDE. We will be using NetBeans.

7. Just code it!

a. Extending the abstract class of a plugin type

Create a "MultiStoreBinders" class under "org.joget.marketplace" package. Then, extend the class with **org.joget.apps.form.model.FormBinder** abstract class.

To make it work as a Form Store Binder, we will need to implement **org.joget.apps.form.model.FormStoreBinder** interface. Then, we need to implement **org.joget.apps.form.model.FormStoreElementBinder** interface to make this plugin show as a selection in store binder select box and implement **org.joget.apps.form.model.FormStoreMultiRowElementBinder** interface to list it under the store binder select box of grid element.

Please refer to Form Store Binder Plugin .

b. Implement all the abstract methods

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

Implementation of all basic abstract methods

```
package org.joget.marketplace;

import org.joget.apps.app.service.AppPluginUtil;
import org.joget.apps.app.service.AppUtil;
import org.joget.apps.form.model.Element;
import org.joget.apps.form.model.FormBinder;
import org.joget.apps.form.model.FormData;
import org.joget.apps.form.model.FormRowSet;
import org.joget.apps.form.model.FormStoreBinder;
import org.joget.apps.form.model.FormStoreElementBinder;
import org.joget.apps.form.model.FormStoreMultiRowElementBinder;

public class MultiStoreBinders extends FormBinder implements FormStoreBinder, FormStoreElementBinder,
FormStoreMultiRowElementBinder {

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

    public String getName() {
        return "multi store binders";
    }

    public String getVersion() {
        return "7.0.0";
    }

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

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

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

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

    public FormRowSet store(Element element, FormRowSet rows, FormData formData) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods,
choose Tools | Templates.
    }
}
```

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 located at "/properties/multiStoreBinders.json". Let us create a directory "resources/properties" under "multi_store_binder/src/main" directory. After creating the directory, create a file named "multiStoreBinders.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" : "@@org.joget.marketplace.MultiStoreBinders.config@@",
        "properties" : [
            {
                "name" : "binders",
                "label" : "@@org.joget.marketplace.MultiStoreBinders.storeBinder@@",
                "type" : "elementmultiselect",
                "options_ajax" : "[CONTEXT_PATH]/web/property/json/getElements?classname=org.joget.apps.form.
model.FormStoreBinder&exclude=org.joget.sample.MultiStoreBinders",
                "url" : "[CONTEXT_PATH]/web/property/json[APP_PATH]/getPropertyOptions",
                "default_property_values_url" : "[CONTEXT_PATH]/web/property/json[APP_PATH]
/getDefaultProperties",
                "required" : "true"
            },{
                "name" : "comment",
                "label" : "@@org.joget.marketplace.MultiStoreBinders.comment@@",
                "type" : "codeeditor"
            }
        ]
    }
]
```

Similar to the Multi Tool plugin, we will need a multi selector to list the available store binders excluding the multi store binders. Refer to Plugin Properties Options#MultiselectinGridInterface(New)

Once we are done, we can work on the main method of the plugin which is store method.

```
    public FormRowSet store(Element element, FormRowSet rows, FormData formData) {
        final Object[] binders = (Object[]) getProperty("binders");
        if (binders != null && binders.length > 0) {
            Thread newThread;
            final PluginManager pluginManager = (PluginManager) AppUtil.getApplicationContext().getBean
("pluginManager");
            newThread = new PluginThread(new Runnable() {
                    public void run() {
                        for (Object binder : binders) {
                            if (binder != null && binder instanceof Map) {
                                Map binderMap = (Map) binder;
                                if (binderMap != null && binderMap.containsKey("className") && !binderMap.get
("className").toString().isEmpty()) {
                                    String className = binderMap.get("className").toString();
                                    FormStoreBinder p = (FormStoreBinder) pluginManager.getPlugin(className);
                                    if (p != null) {
                                        Map properties = new HashMap();
                                        properties.putAll((Map) binderMap.get
("properties"));
                                        if (p instanceof PropertyEditable) {
                                            ((PropertyEditable) p).setProperties(properties);
                                        }
                                        p.store(element, rows, formData);
                                    }
                                }
                            }
                        }
                    }
                });
                newThread.start();
        }
        return null;
    }
```

c. Manage the dependency libraries of your plugin

Our plugin is using dbcp, javax.servlet.http.HttpServletRequest and javax.servlet.http.HttpServletResponse class, so we will need to add jsp-api and commons-dbcp library 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>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20080701</version>
    <scope>provided</scope>
</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 "jdbc_store_binder/src/main" directory. Then, create a "multiStoreBinders.properties" file in the folder. In the properties file, let add all the message keys and its label as below.

```
org.joget.marketplace.MultiStoreBinders.pluginLabel=multi store binders
org.joget.marketplace.MultiStoreBinders.pluginDesc=Enable the use of multiple store binders
org.joget.marketplace.MultiStoreBinders.config=Configure multi store binders
org.joget.marketplace.MultiStoreBinders.storeBinder=Store Binders
org.joget.marketplace.MultiStoreBinders.comment=Comment
```

e. Register your plugin to Felix Framework

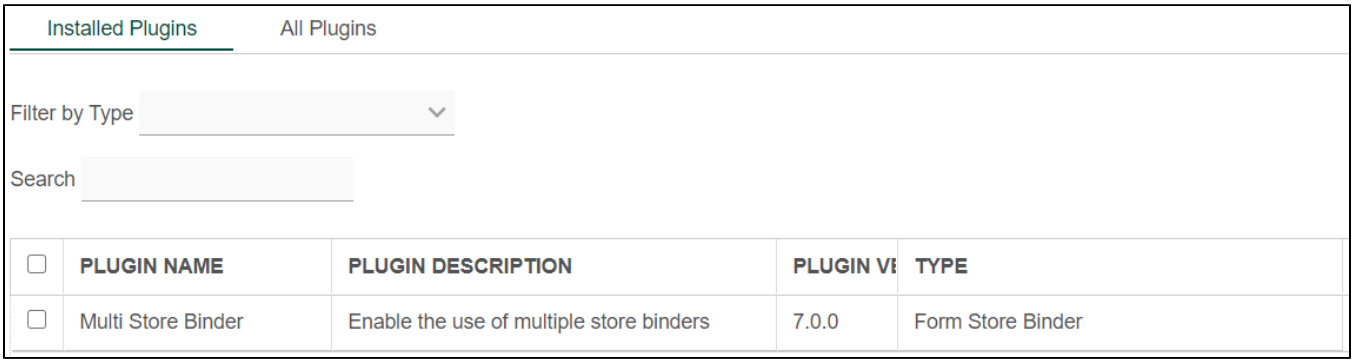
We will have to register our plugin class in Activator class (Auto generated in the same class package) to tell Felix Framework that this is a plugin.

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

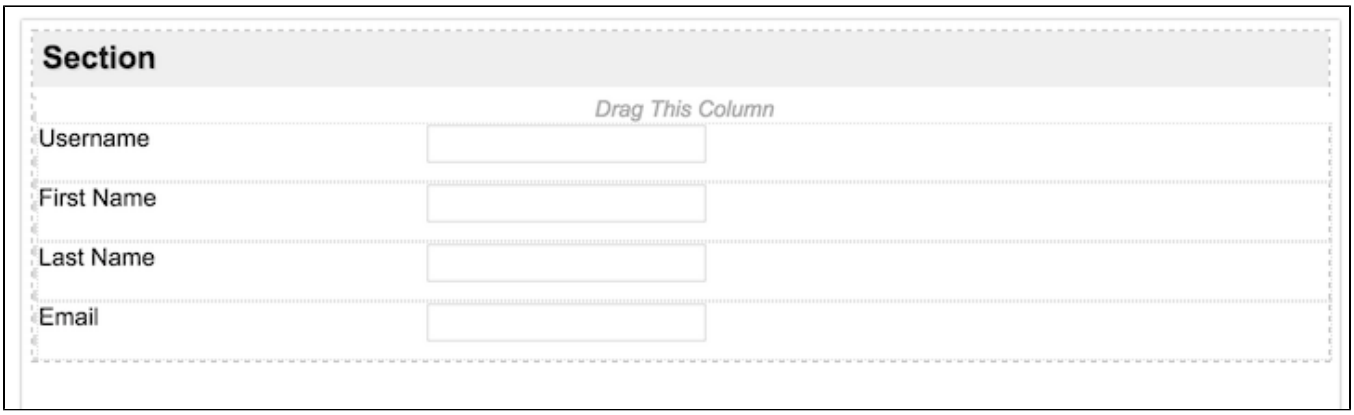
f. Build it and testing

Let build our plugin. Once the building process is done, we will found a "multi_store_binder-7.0.0.jar" file is created under the "multi_store_binder/target" directory.

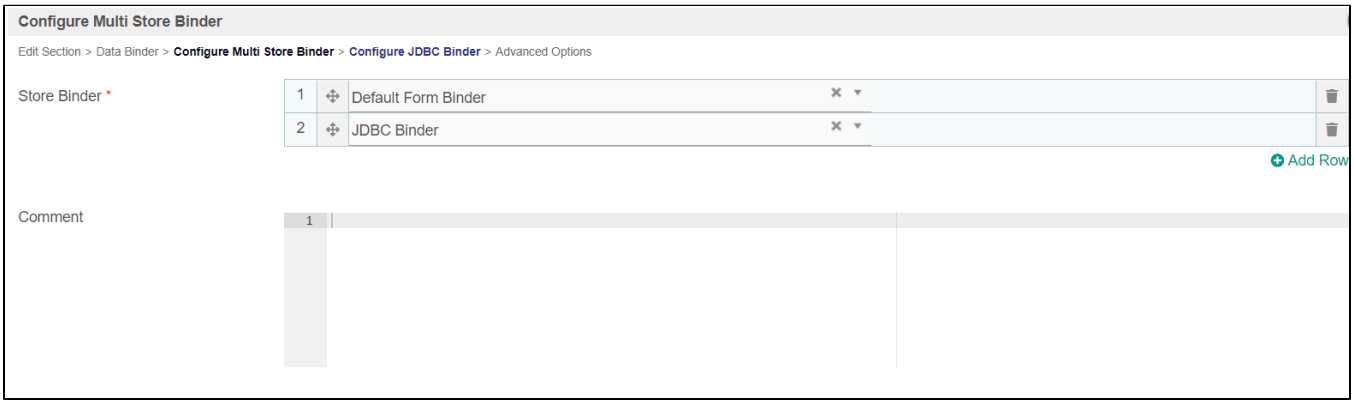
Then, let upload the plugin jar to Manage Plugins. After upload, the jar file, double-check the plugin is uploaded and activated correctly.



Let create a form to create and update user to dir_user table.



Then, configure the store binder of the section to Default Form Binder and JDBC Binder.



Check Select Query

```
select username from dir_user where username = {id}
```

Insert Query

```
insert into dir_user
(id, username, firstName, lastName, email, active)
values
({id}, {id}, {firstName}, {lastName}, {email}, 1)
```

Note

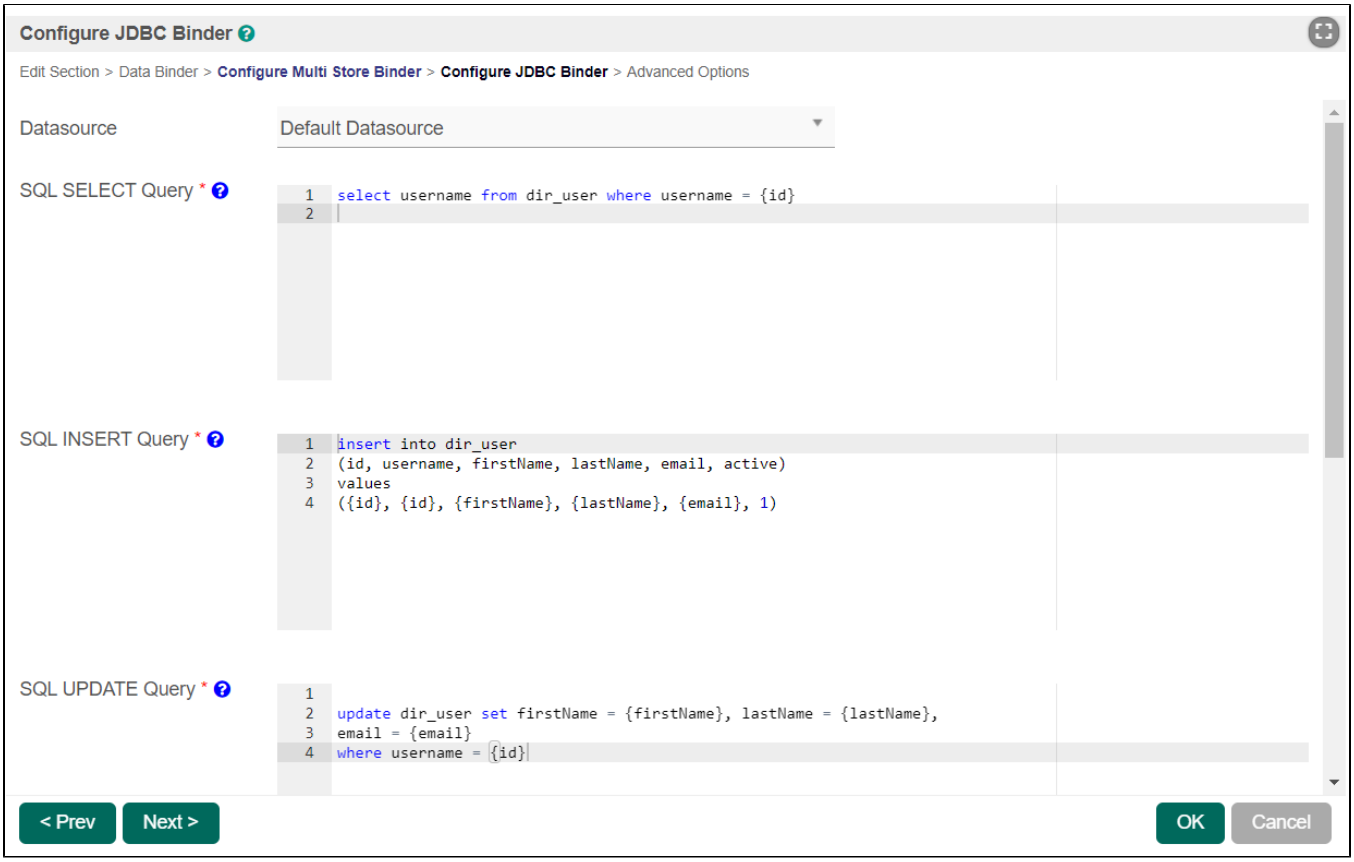
{uuid} can be used to generate a unique id

Update Query

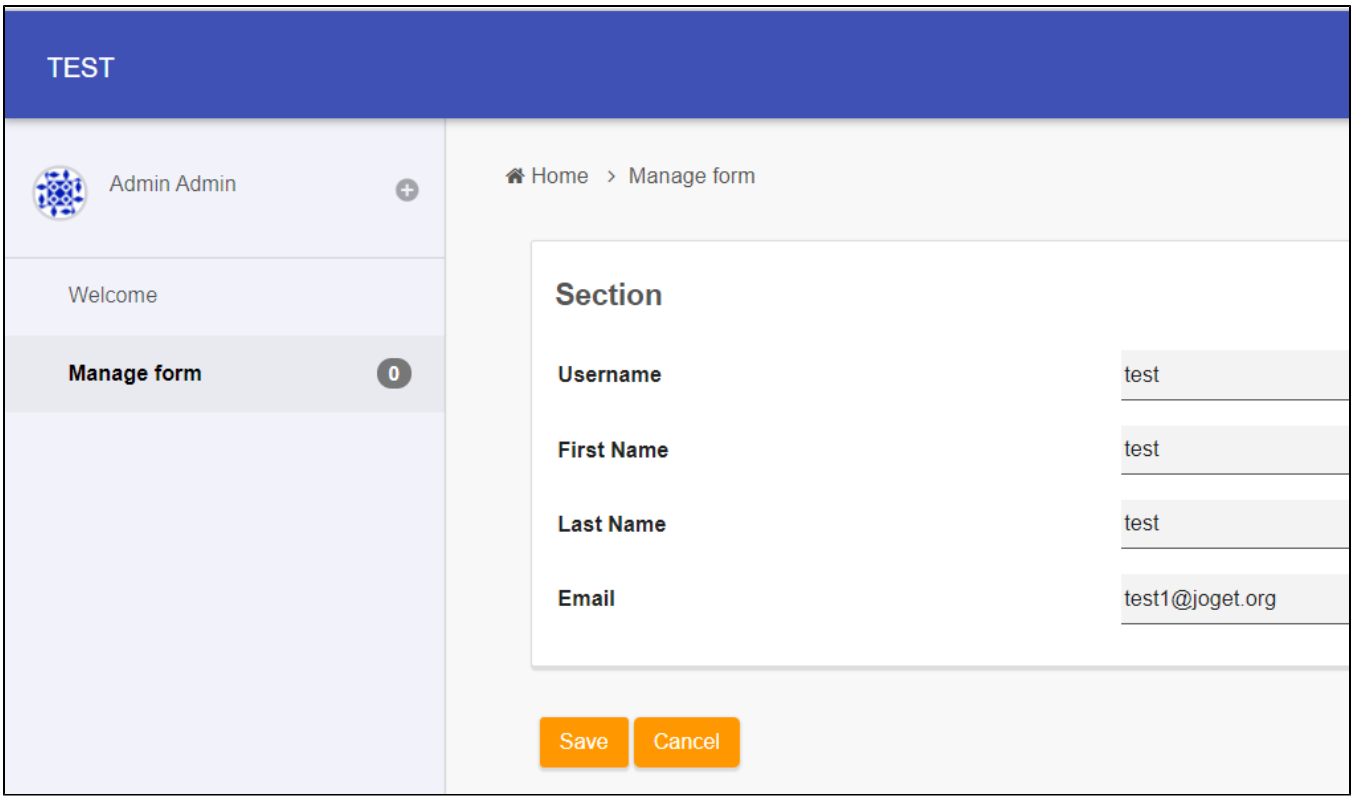
```
update dir_user set firstName = {firstName}, lastName = {lastName},
email = {email}
where username = {id}
```

Delete Query

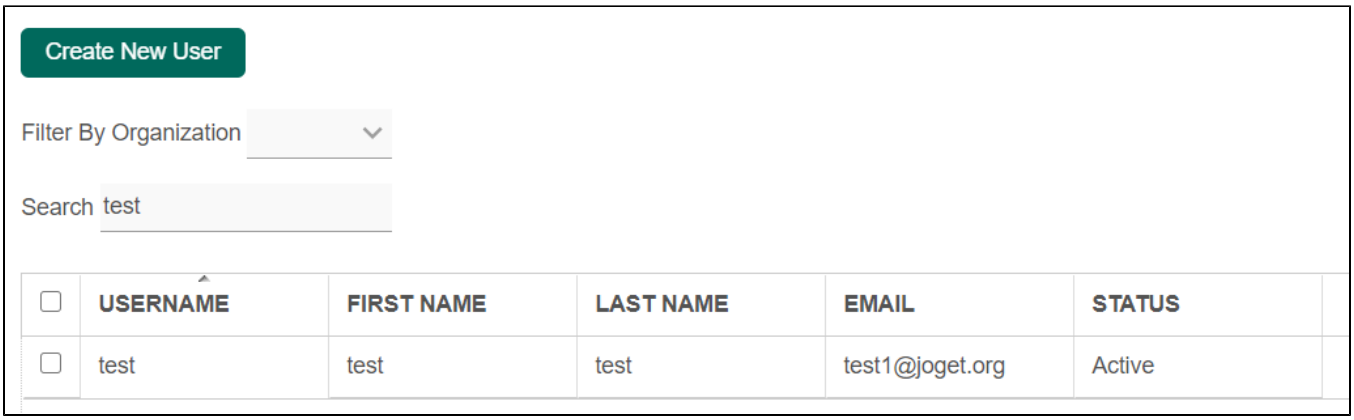
```
delete from TABLE_NAME where id = {id}
```



Now, let test to add a user.



Check the user is created in dir_user table.



It works! Please remember to test the other features of the plugin as well. 😃

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

You can download the source code from multi_store_binder.zip

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