How to develop a Bean Shell Hash Variable

In this tutorial, we will be following the guideline for developing a plugin to develop our Bean Shell Hash Variable plugin.

1. What is the problem?

Hash variable is convenient to use, but sometime we want to do some condition check before displaying a value. But, Hash variable does not provide the ability for condition checking.

2. How to solve the problem?

By looking at the Plugin Types that are currently supported by Joget Workflow, we can develop a Hash Variable Plugin to allow us to write our scripting for condition checking. There are quite a number of Bean Shell plugins provided as default plugin for several plugin types. We can do one for Hash Variable plugin as well.

3. What is the input needed for your plugin?

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 App Variable to store our script. 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 also. 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?

What do we expected from this Bean Shell Hash variable plugin? The Bean Shell Hash Variable plugin is for admin/developer user to use when building/developing an app. Once used, the Hash Variable will be replaced by the output return from the Bean Shell interpreter. So that the admin user can do condition check before display something to normal user.

> E.g. Display a welcome message for logged in user but display nothing when the user is an anonymous.

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

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 the Environment Variable Hash Variable plugin on how to retrieve environment variable using a variable key. We can also refer to the Bean Shell Tool or Bean Shell Form Binder plugin on what to execute the script with Bean Shell interpreter.

We can use getUrlParams method from StringUtil to help us parse parameters passed in with URL query parameters syntax.

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 Joget Source Code version 8.0 - Snapshot. Please refer to the Guideline for Developing a Plugin article for other platform commands.

Let say our folder directory is as following.

```
- 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 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.tutorial beanshell_hash_variable 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.tutorial
Confirm properties configuration:
groupId: org.joget.tutorial
artifactId: beanshell_hash_variable
version: 5.0.0
package: org.joget.tutorial
Y: : y
```

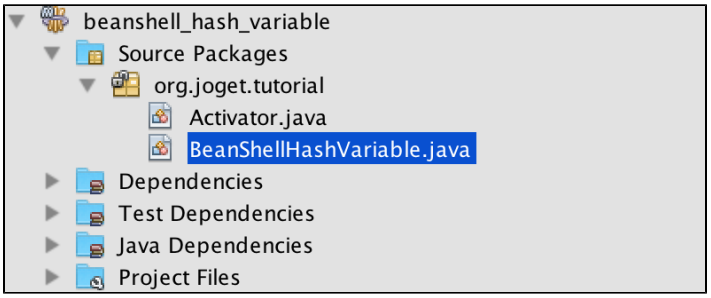
We should get "BUILD SUCCESS" message shown in our terminal and a "beanshell_hash_variable" folder created in "plugins" folder.

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

7. Just code it!

a. Extending the abstract class of a plugin type

Create a "BeanShellHashVariable" class under "org.joget.tutorial" package.



Then, based on Hash Variable Plugin document, we will have to extends **org.joget.apps.app.model.DefaultHashVariablePlugin** abstract class.

b. Implement all the abstract methods

Let us implement all the abstract methods. We will be using AppPluginUtil.getMessage method to support i18n and using constant variable MESSAGE_PATH for message resource bundle directory.

Implementation of all basic abstract methods

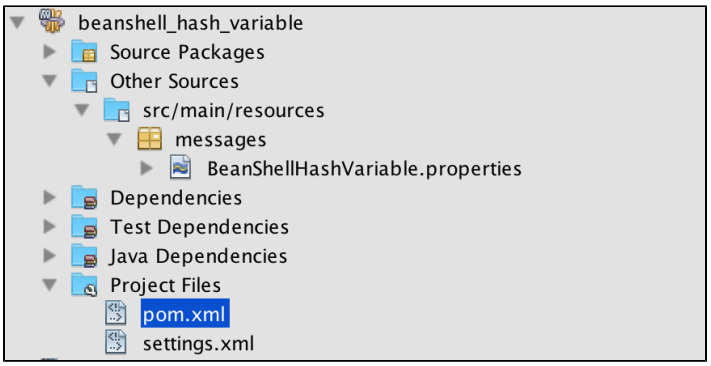
```
package org.joget.tutorial;

import org.joget.apps.app.model.DefaultHashVariablePlugin;
import org.joget.apps.app.service.AppPluginUtil;

public class BeanShellHashVariable extends DefaultHashVariablePlugin {

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

    public String getName() {
        return "BeanShellHashVariable";
    }

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

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

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

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

    public String getPropertyOptions() {
        //Hash variable plugin do not support property options
        return "";
    }

    public String getPrefix() {
        return "beanshell";
    }

    public String processHashVariable(String variableKey) {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

Now, let's focus on the main method of our Hash Variable plugin which is processHashVariable. We will refer to the source code of Environment Variable Hash Variable plugin on how to retrieve the Environment variable. Then, refer to the source code of Bean Shell Form Binder on how to execute a bean shell script.

```
    public String processHashVariable(String variableKey) {
        try {
            String environmentVariableKey = variableKey;

            //first check and retrieve parameters passed in with URL query parameters syntax wrapped in square
bracket []
            String queryParams = null;
            if (variableKey.contains("[") && variableKey.contains("]")) {
                queryParams = variableKey.substring(variableKey.indexOf("[") + 1, variableKey.indexOf("]"));
                environmentVariableKey = variableKey.substring(0, variableKey.indexOf("["));
            }

            //Parse the query parameters to a map
            Map<String, String[]> parameters = null;
            if (queryParams != null && !queryParams.isEmpty()) {
                parameters = StringUtil.getUrlParams(queryParams);

                //put all parameters to plugin properties
                getProperties().putAll(parameters);
            }

            //Retrieve the environment variable based on environmentVariableKey
            AppDefinition appDef = (AppDefinition) getProperty("appDefinition");
            if (appDef != null) {
                ApplicationContext appContext = AppUtil.getApplicationContext();
                EnvironmentVariableDao environmentVariableDao = (EnvironmentVariableDao) appContext.getBean
("environmentVariableDao");
                EnvironmentVariable env = environmentVariableDao.loadById(environmentVariableKey, appDef);
                if (env != null) {
                    String script = env.getValue();
                    //execute the script with all plugin properties
                    return executeScript(script, getProperties());
                } else {
                    //environment variable not found, return empty value
                    return "";
                }
            }
        } catch (Exception e) {
            //log the exception using LogUtil
            LogUtil.error(getClassName(), e, "#beanshell."+variableKey+"# fail to parse.");
        }

        //return null to by pass the replacing
        return null;
    }

    protected String executeScript(String script, Map properties) throws Exception {
        Interpreter interpreter = new Interpreter();
        interpreter.setClassLoader(getClass().getClassLoader());
        for (Object key : properties.keySet()) {
            interpreter.set(key.toString(), properties.get(key));
        }
        LogUtil.debug(getClass().getName(), "Executing script " + script);
        return (String) interpreter.eval(script);
    }
```

c. Manage the dependency libraries of your plugin

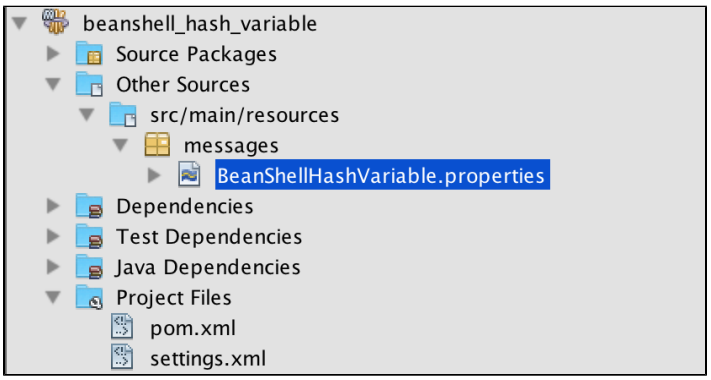
Our plugin class cannot resolve "bsh.Interpreter". So, we will have to add bean shell library to our POM file.




```
<!-- Change plugin specific dependencies here -->
<dependency>
    <groupId>org.beanshell</groupId>
    <artifactId>bsh</artifactId>
    <version>2.0b4</version>
</dependency>
<!-- End change plugin specific dependencies here -->
```

d. Make your plugin internationalization (i18n) ready

We are using AppPluginUtil.getMessage method to display i18n value for our getLabel and getDescription method. We will have to create a message resource bundle properties file for it. Create directory "resources/messages" under "beanshell_hash_variable/src/main" directory. Then, create a "BeanShellHashVariable.properties" file in the folder.

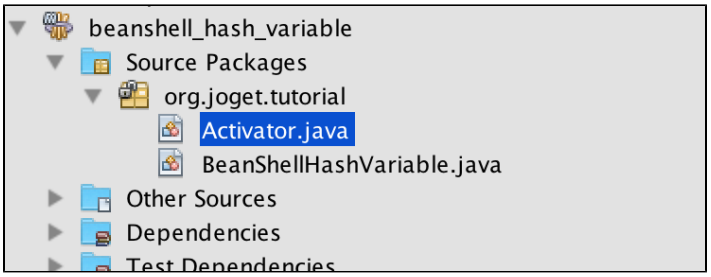


In our properties file, we will need to add the key we have used.

```
org.joget.tutorial.BeanShellHashVariable.pluginLabel=Bean Shell Hash Variable
org.joget.tutorial.BeanShellHashVariable.pluginDesc=Using environment variable to execute bean shell script.
```

e. Register your plugin to the Felix Framework

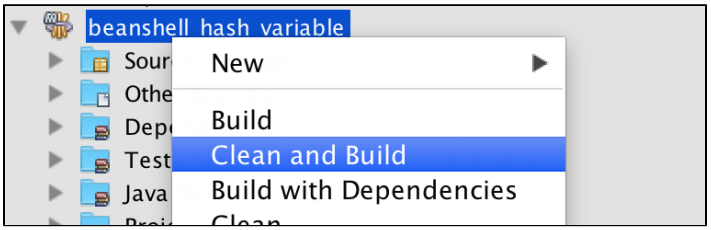
We will have to register our plugin class in Activator class to tell the Felix Framework that this is a plugin.



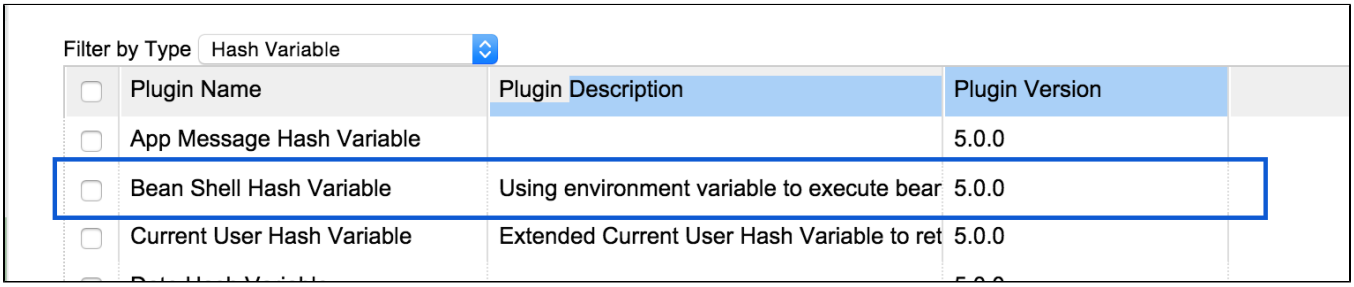
```
public void start(BundleContext context) {
    registrationList = new ArrayList<ServiceRegistration>();

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

f. Build it and test

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

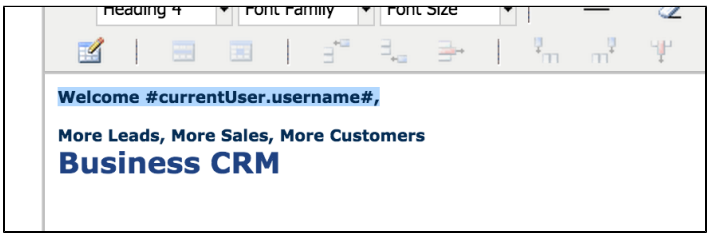


Then, let's upload the plugin jar to Manage Plugins. After uploading the jar file, double check that the plugin is uploaded and activated correctly.



Now, let's test our plugin.

Let assume that we have a HTML menu page in a userview that wants to display the following line to logged in user. Normally, we will use "Welcome #currentUser,username#," to display a welcome message.



But, in this use case there is a problem, which shows "Welcome ," without an username when the user is an anonymous.

Now, change the whole message to our Bean Shell Hash Variable and create an environment variable to put our script.

Change:

```
Welcome #currentUser.username#,
```

to the following. We will need to pass the current user's username as one of our parameters and do not forget to escape it as url.

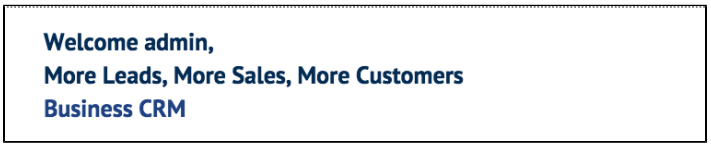
```
#beanshell.welcome[username={currentUser.username?url}]?html#
```

Then, we can create an environment variable with ID "welcome" and use the following script. As we are using getUrlParams method from StringUtil to parse the parameters, all value from parameters are String array.

```
//all parameters passed in from Beanshell Hash Variable will converted to String array
if (username != null && username.length == 1 && !username[0].isEmpty()) {
    return "Welcome " + username[0] + ",";
} else {
    return "";
}
```

Let go back to our HTML menu page to see the result.

When user is logged in, it shows the message correctly.



When no user is logged in, the welcome message is not shown.

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

You can download the source code from beanshell_hash_variable.zip.