Bean Shell Programming

Plugin Type

- Application
- Form Variable
- Participant
- Activity Tool

What is it for?

BeanShell is a small, embeddable Java source interpreter with object scripting language features written in Java. BeanShell dynamically executes standard Java syntax. So, by using BeanShell Plugin, you can type in valid Java codes in plugin configuration and the statements will be executed when the plugin is triggered. No compilation cycle is needed.

Configurations

1. Script

The script can be composed of:

- Java syntax supported by the version of JDK used
- Usage of the following libraries:
    - Libraries available in JDK
    - org.joget.workflow.model.*
    - org.joget.workflow.util.*
    - JavaBeans Activation Framework 1.1
    - Commons Email 1.1
    - JavaMail 1.4
    - MySQL JDBC Driver 3.1.14
    - Oracle JDBC Driver 10.2.0.2
    - Microsoft SQL Server JDBC Driver 1.0
- Usage of Hash Variables

Example

Set value to a workflow variable:

```
import org.joget.workflow.model.service.*;

WorkflowManager wm = (WorkflowManager) pluginManager.getBean("workflowManager");

wm.activityVariable(workflowAssignment.getActivityId(), "variableId", "variableValue");
```

Execute SQL statement through JDBC:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import javax.sql.DataSource;
import org.joget.apps.app.service.AppUtil;

Connection con = null;
try {
    // retrieve connection from the default datasource
    DataSource ds = (DataSource)AppUtil.getApplicationContext().getBean("setupDataSource");
    con = ds.getConnection();

    // execute SQL query
    if(!con.isClosed()) {
        PreparedStatement stmt = con.prepareStatement("UPDATE table1 SET column1='value1'");
        stmt.executeUpdate();
    }
} catch(Exception e) {
    System.err.println("Exception: " + e.getMessage());
} finally {
    try {
        if(con != null) {
            con.close();
        }
    } catch(SQLException e) {
    }
}
```

Participant Type Plugin

A participant type plugin should return a collection of usernames. In the participant plugin, there are two context variables available for the script to use :

- pluginManager
- workflowActivity

A Beanshell participant type plugin returning one participant:

```
import java.util.ArrayList;

a = new ArrayList();

a.add("jack"); // one username

return a;
```

Get all users from the DirectoryManager and assigning them:

```
import java.util.Collection;
import java.util.ArrayList;
import org.joget.apps.app.service.AppUtil;
import org.joget.directory.model.service.ExtDirectoryManager;
import org.joget.directory.model.User;
import org.springframework.context.ApplicationContext;

ApplicationContext ac = AppUtil.getApplicationContext();
ExtDirectoryManager directoryManager = (ExtDirectoryManager) ac.getBean("directoryManager");

Collection results = new ArrayList();

Collection userList = directoryManager.getUserList();

for (User u : userList) {
        results.add(u.getUsername());
}
return results;
```

Activity Tool Type Plugin

Activity Tool Type Plugin is a plugin that will be executed when the workflow reaches a System Tool activity. System Tool activities must be placed in the workflow diagram and the Beanshell plugin configured in the Process's Activity Mapping for this to work. In this plugin, there are two context variables available for the script to use :

- pluginManager
- workflowAssignment
- appDef - AppDefinition

A Beanshell activity tool type plugin that sets a form data:

```
import java.util.HashMap;
import java.util.Map;
import org.joget.apps.app.model.AppDefinition;
import org.joget.apps.app.service.AppService;
import org.joget.apps.app.service.AppUtil;
import org.joget.apps.form.model.FormRow;
import org.joget.apps.form.model.FormRowSet;
import org.joget.apps.form.service.FormUtil;
import org.joget.workflow.model.WorkflowAssignment;
import org.joget.workflow.util.WorkflowUtil;

//Constant variable
String formDefId = "approvalForm";

//Service bean
AppService appService = (AppService) AppUtil.getApplicationContext().getBean("appService");

//Get primary key
String id = appService.getOriginProcessId(workflowAssignment.getProcessId());

//Get existing data
FormRowSet rowSet = appService.loadFormData(appDef.getAppId(), appDef.getVersion().toString(), formDefId, id);
FormRow row = null;
if (rowSet == null || rowSet.isEmpty()) {
    rowSet = new FormRowSet();
    row = new FormRow();
    row.setId(id);
    rowSet.add(row);
} else {
    row = rowSet.get(0);
}

//Set values
row.setProperty("field1", "value 1");
row.setProperty("field2", "value 2");
row.setProperty("field3", "value 3");

//Save
appService.storeFormData(appDef.getAppId(), appDef.getVersion().toString(), formDefId, rowSet, id);
```

Increment a workflow variable value:

```
import org.joget.workflow.model.service.WorkflowManager;

WorkflowManager workflowManager = (WorkflowManager) pluginManager.getBean("workflowManager");
String approvalLvl = workflowManager.getProcessVariable(workflowAssignment.getProcessId(), "ApprovalLevel");
String newApprovalLvl = String.valueOf(Integer.parseInt(approvalLvl)+1);
workflowManager.activityVariable(workflowAssignment.getActivityId(), "ApprovalLevel", newApprovalLvl);
```

This beanshell script retrieves workflow variable 'ApprovalLevel', converts it to an integer, adds 1, and stores it back as the same workflow variable.