Bean Shell Programming Guide

What is it for?

- Joget Workflow provided Bean Shell implementation as several Plugin Types. Please refer to usages section.
- BeanShell is a small, embeddable Java source interpreter with object scripting language features written in Java.
- BeanShell dynamically executes standard Java syntax in runtime.
- 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.

What is the code syntax?

- Java syntax supported by the version of JDK used
- Usage of all libraries that Joget used. For details please refer to NOTICE.txt in the release installer/bundle.
- Usage of all Joget Workflow Utility and Service Methods.
- Usage of Hash Variable.

Usages

Use as Form Load Binder

Injected Variables:

- element - Element that this binder is tie to. (org.joget.apps.form.model.Element)
- primaryKey - The primary key provided by the element to load data. (java.lang.String)

- formData - The data holder of the whole form. (org.joget.apps.form.model.FormData)

Expected Return Object:

- An org.joget.apps.form.model.FormRowSet object which contains one org.joget.apps.form.model.FormRow object.

Samples:

Load user data using jdbc.

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.sql.DataSource;
import org.joget.apps.app.service.AppUtil;
import org.joget.apps.form.model.Element;
import org.joget.apps.form.model.FormData;
import org.joget.apps.form.model.FormRow;
import org.joget.apps.form.model.FormRowSet;
import org.joget.commons.util.LogUtil;

public FormRowSet load(Element element, String username, FormData formData) {
    FormRowSet rows = new FormRowSet();
    if (username != null && !username.isEmpty()) {
        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("SELECT username, firstName, lastName, email from
dir_user where username=?");
                stmt.setObject(1, username);
                ResultSet rs = stmt.executeQuery();
                while (rs.next()) {
                    FormRow row = new FormRow();
                    System.out.println(rs.getObject("username") );
                    row.setProperty("username", (rs.getObject("username") != null)?rs.getObject("username").
toString():"");
                    row.setProperty("firstName", (rs.getObject("firstName") != null)?rs.getObject("firstName").
toString():"");
                    row.setProperty("lastName", (rs.getObject("lastName") != null)?rs.getObject("lastName").
toString():"");
                    row.setProperty("email", (rs.getObject("email") != null)?rs.getObject("email").
toString():"");

                    rows.add(row);
                    break;
                }
            }
        } catch(Exception e) {
            LogUtil.error("Sample app - Form 1", e, "Error loading user data in load binder");
        } finally {
            //always close the connection after used
            try {
                if(con != null) {
                    con.close();
                }
            } catch(SQLException e) {/* ignored */}
        }
    }
    return rows;
}

//call load method with injected variable
return load(element, primaryKey, formData);
```

Use as Form Options Binder

Injected Variables:

- element - Element that this binder is tie to. (org.joget.apps.form.model.Element)
- primaryKey - The primary key provided by the element to load data. (java.lang.String)

- formData - The data holder of the whole form. (org.joget.apps.form.model.FormData)

Expected Return Object:

- An org.joget.apps.form.model.FormRowSet object which contains one or more org.joget.apps.form.model.FormRow object. All FormRow objects are **expected** to have **"value"** and **"label"** property.

Samples:

Load time zone as the select box options.

```
import java.util.Map;
import org.joget.apps.app.service.AppUtil;
import org.joget.apps.form.model.Element;
import org.joget.apps.form.model.FormData;
import org.joget.apps.form.model.FormRow;
import org.joget.apps.form.model.FormRowSet;
import org.joget.apps.form.service.FormUtil;
import org.joget.commons.util.TimeZoneUtil;

public FormRowSet load(Element element, String primaryKey, FormData formData) {
    FormRowSet rows = new FormRowSet();

    //Get timezones using timezone util
    for(Map.Entry entry : TimeZoneUtil.getList().entrySet()){
        FormRow option = new FormRow();
        option.setProperty(FormUtil.PROPERTY_VALUE, (String) entry.getKey());
        option.setProperty(FormUtil.PROPERTY_LABEL, (String) entry.getValue());
        rows.add(option);
    }

    return rows;
}

//call load method with injected variable
return load(element, primaryKey, formData);
```

Use as Form Ajax Options Binder

Injected Variables:

- values - Dependency values of the controlling field. (java.lang.String[])

Expected Return Object:

- An org.joget.apps.form.model.FormRowSet object which contains one or more org.joget.apps.form.model.FormRow object. All FormRow objects are **expected** to have **"value"** and **"label"** property.

Samples:

Load user as options based on the group id passed by the controlling field.

```
import java.util.Collection;
import org.joget.apps.app.service.AppUtil;
import org.joget.apps.form.model.Element;
import org.joget.apps.form.model.FormData;
import org.joget.apps.form.model.FormRow;
import org.joget.apps.form.model.FormRowSet;
import org.joget.apps.form.service.FormUtil;
import org.joget.directory.model.User;
import org.joget.directory.model.service.ExtDirectoryManager;

public FormRowSet load(String[] values) {
    FormRowSet rows = new FormRowSet();

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

    //set groupId based on dependency value
    String groupId = null;
    if (values != null && values.length > 0) {
        groupId = values[0];
    }

    //Get users using directory manager
    Collection userList = directoryManager.getUsers(null, null, null, null, groupId, null, null, "firstName",
false, null, null);
    for(Object u : userList){
        User user = (User) u;
        FormRow option = new FormRow();
        option.setProperty(FormUtil.PROPERTY_VALUE, user.getUsername());
        option.setProperty(FormUtil.PROPERTY_LABEL, user.getFirstName() + " " + user.getLastName());
        rows.add(option);
    }

    return rows;
}

//call load method with injected variable
return load(values);
```

Use as Form Store Binder

Injected Variables:

- element - Element that this binder is tie to. (org.joget.apps.form.model.Element)
- rows - Data to be store. Contains only one org.joget.apps.form.model.FormRow object. (org.joget.apps.form.model.FormRowSet)
- formData - The data holder of the whole form. (org.joget.apps.form.model.FormData)

Expected Return Object:

- Same org.joget.apps.form.model.FormRowSet object which stored.

Samples:

Do some calculation before storing the data using Workflow Form Binder.

```
import java.text.DecimalFormat;
import org.joget.apps.app.service.AppUtil;
import org.joget.apps.form.model.Element;
import org.joget.apps.form.model.FormData;
import org.joget.apps.form.model.FormRow;
import org.joget.apps.form.model.FormRowSet;
import org.joget.apps.form.model.FormStoreBinder;
import org.joget.plugin.base.PluginManager;

public FormRowSet store(Element element, FormRowSet rows, FormData formData) {

    //check the rows is not empty before store it
    if (rows != null && !rows.isEmpty()) {
        //Get the submitted data
        FormRow row = rows.get(0);

        String unitPrice = row.getProperty("unitPrice");
        String unit = row.getProperty("unit");

        double total = 0;
        try {
            double price = Double.parseDouble(unitPrice);
            int unit = Integer.parseInt(unit);

            total = price * unit;
        } catch (Exception e) {}

        //format the total to 2 decimal
        DecimalFormat df = new DecimalFormat("#.00");
        row.setProperty("total", df.format(total));

        //Reuse Workflow Form Binder to store data
        PluginManager pluginManager = (PluginManager) AppUtil.getApplicationContext().getBean("pluginManager");
        FormStoreBinder binder = (FormStoreBinder) pluginManager.getPlugin("org.joget.apps.form.lib.
WorkflowFormBinder");
        binder.store(element, rows, formData);
    }

    return rows;
}

//call store method with injected variable
return store(element, rows, formData);
```

Use as Form Validator

Injected Variables:

- element - Element that this validator is tie to. (org.joget.apps.form.model.Element)
- values - The submitted values of the element. (java.lang.String[])
- formData - The data holder of the whole form. (org.joget.apps.form.model.FormData)

Expected Return Object:

- A boolean value to indicate the validation pass or fail.

Samples:

Compare the submitted value with the value of another field.

```
import java.util.Arrays;
import org.joget.apps.app.service.AppUtil;
import org.joget.apps.form.model.Element;
import org.joget.apps.form.model.Form;
import org.joget.apps.form.model.FormData;
import org.joget.apps.form.service.FormUtil;

public boolean validate(Element element, FormData formData, String[] values) {
    boolean result = true;

    //get field 1 value from form data object
    String field1Id = "field1";
    Form form = FormUtil.findRootForm(element);
    Element field1 = FormUtil.findElement(field1Id, form, formData);

    if (field1 != null) {
        //get value of field 1
        String[] compareValues = FormUtil.getElementPropertyValues(field1, formData);

        //compare the value of field 2 and field 1 are equals
        if (!Arrays.equals(values, compareValues)) {
            String id = FormUtil.getElementParameterName(element);
            formData.addFormError(id, "Value not equal!!!!");
            result = false;
        }
    } else {
        //ignore if the field 1 not exist
    }

    return result;
}

//call validate method with injected variable
return validate(element, formData, values);
```

Textfield / Textarea length not exceeding 500 characters

```
import java.util.Arrays;
import org.joget.apps.form.model.Element;
import org.joget.apps.form.model.FormData;
import org.joget.apps.form.service.FormUtil;

if(values[0].length() > 500){
    String id = FormUtil.getElementParameterName(element);
    formData.addFormError(id, "Value cannot be longer than 500 characters. You have entered " + values[0].
length() + " characters.");
    return false;
}else{
    return true;
}
```

Field value does not exceed 500 in numerical value

```
import java.util.Arrays;
import org.joget.apps.form.model.Element;
import org.joget.apps.form.model.FormData;
import org.joget.apps.form.service.FormUtil;

String id = FormUtil.getElementParameterName(element);
String amount = "";
if(values.length > 0){
    amount = values[0];
}

if(amount.isEmpty()){
    formData.addFormError(id, "Please key in a amount.");
    return false;
}else{
    float amountF = Float.parseFloat(amount);
    if(amountF >=500){
        formData.addFormError(id, "Amount cannot be more than 500.");
        return false;
    }
}

return true;
```

Use as Form Multi Row Load Binder

Injected Variables:

- element - Element that this binder is tie to. (org.joget.apps.form.model.Element)
- primaryKey - The primary key provided by the element to load data. (java.lang.String)
- formData - The data holder of the whole form. (org.joget.apps.form.model.FormData)

Expected Return Object:

- An org.joget.apps.form.model.FormRowSet object which contains one or more org.joget.apps.form.model.FormRow object.

Samples:

Load default grid data from another table if current record does not have any grid data.

```
import org.joget.apps.form.model.Element;
import org.joget.apps.form.model.FormData;
import org.joget.apps.form.model.FormRow;
import org.joget.apps.form.model.FormRowSet;
import org.joget.apps.form.service.FormUtil;
import org.joget.plugin.base.PluginManager;
import org.joget.apps.form.model.FormLoadBinder;

public FormRowSet load(Element element, String primaryKey, FormData formData) {
    String defaultFormDefId = "default_grid_entry"; //change this to the form id used to store default grid data
    String formDefId = "grid_entry";  //change this to the form id used to store grid data
    String foreignKey = "fk"; //change this to the foreign key

    FormRowSet f = new FormRowSet();
    f.setMultiRow(true);

    // Reuse Multi Row Binder to load data
    PluginManager pluginManager = (PluginManager) FormUtil.getApplicationContext().getBean("pluginManager");
    FormLoadBinder binder = (FormLoadBinder) pluginManager.getPlugin("org.joget.plugin.enterprise.
MultirowFormBinder");

    //Load from the grid table
    binder.setProperty("formDefId", formDefId);
    binder.setProperty("foreignKey", foreignKey);
    f = binder.load(element, primaryKey, formData);

    //if no grid data is retrieved, get from default table
    if (f == null || f.isEmpty()) {
        binder.setProperty("formDefId", defaultFormDefId);

        //set the foreign key value to empty
        f = binder.load(element, "", formData);
    }
    return f;
}

//call load method with injected variable
return load(element, primaryKey, formData);
```

Use as Form Multi Row Store Binder

Injected Variables:

- element - Element that this binder is tie to. (org.joget.apps.form.model.Element)
- rows - Data to be store. Contains one or more org.joget.apps.form.model.FormRow object. (org.joget.apps.form.model.FormRowSet)
- formData - The data holder of the whole form. (org.joget.apps.form.model.FormData)

Expected Return Object:

- A org.joget.apps.form.model.FormRowSet object.

Samples:

Bulk create users based on the grid data.

```
import java.util.HashSet;
import java.util.Set;
import org.joget.apps.app.service.AppUtil;
import org.joget.apps.form.model.Element;
import org.joget.apps.form.model.FormData;
import org.joget.apps.form.model.FormRow;
import org.joget.apps.form.model.FormRowSet;
import org.joget.commons.util.LogUtil;
import org.joget.commons.util.StringUtil;
import org.joget.directory.model.Role;
import org.joget.directory.model.User;
import org.joget.directory.dao.RoleDao;
import org.joget.directory.dao.UserDao;
import org.joget.directory.model.service.DirectoryUtil;
import org.joget.directory.model.service.UserSecurity;

public FormRowSet store(Element element, FormRowSet rows, FormData formData) {
    try {
        UserSecurity us = DirectoryUtil.getUserSecurity();
        RoleDao roleDao = (RoleDao) AppUtil.getApplicationContext().getBean("roleDao");
        UserDao userDao = (UserDao) AppUtil.getApplicationContext().getBean("userDao");

        for (FormRow row : rows) {
            //Get the submitted data for each grid row
            String username = row.getProperty("username");
            String firstName = row.getProperty("firstName");
            String lastName = row.getProperty("lastName");
            String email = row.getProperty("email");
            String password = row.getProperty("password");

            User user = new User();
            user.setId(username);
            user.setUsername(username);
            user.setTimeZone("0");
            user.setActive(1);
            user.setFirstName(firstName);
            user.setLastName(lastName);
            user.setEmail(email);

            //Check if there is user security implementation, using it to encrypt password
            if (us != null) {
                user.setPassword(us.encryptPassword(username, password));
            } else {
                user.setPassword(StringUtil.md5Base16(password));
            }
            user.setConfirmPassword(password);

            //set user role
            Set roleSet = new HashSet();
            roleSet.add(roleDao.getRole("ROLE_USER"));
            user.setRoles(roleSet);

            userDao.addUser(user);
            if (us != null) {
                us.insertUserPostProcessing(user);
            }
        }
    } catch (Exception e) {
        LogUtil.error("Sample app - Bulk Create Users form", e, "Store user error!!");
    }
    return rows;
}

//call store method with injected variable
return store(element, rows, formData);
```

Use as Form Multi Row Validator

Injected Variables:

- element - Element that this validator is tie to. (org.joget.apps.form.model.Element)
- rows - Submitted data. Contains one or more org.joget.apps.form.model.FormRow object. (org.joget.apps.form.model.FormRowSet)
- formData - The data holder of the whole form. (org.joget.apps.form.model.FormData)

Expected Return Object:

- A boolean value to indicate the validation pass or fail.

Samples:

Validate the sum of a column values are less than 1000.

```
import java.util.Arrays;
import org.joget.apps.app.service.AppUtil;
import org.joget.apps.form.model.Element;
import org.joget.apps.form.model.Form;
import org.joget.apps.form.model.FormData;
import org.joget.apps.form.model.FormRow;
import org.joget.apps.form.model.FormRowSet;
import org.joget.apps.form.service.FormUtil;

public boolean validate(Element element, FormRowSet rows, FormData formData) {
    boolean result = true;
    if (rows != null && !rows.isEmpty()) {
        int total = 0;

        //Sum the values from column "amount"
        for (FormRow row : rows) {
            try {
                int amount = Integer.parseInt(row.getProperty("amount"));
                total += amount;
            } catch (Exception e) {}
        }

        //if amount larger than 1000
        if (total > 1000) {
            String id = FormUtil.getElementParameterName(element);
            formData.addFormError(id, "Total amount should not larger than 1000!!!!");
            result = false;
        }
    }

    return result;
}

//call validate method with injected variable
return validate(element, rows, formData);
```

Use as Form Permission

Injected Variables:

- user - User object of current logged in user (org.joget.directory.model.User)
- requestParams - Request parameters map of current HTTP Request (java.util.Map)

Expected Return Object:

- A boolean value to indicate the user is authorized.

Samples:

Check the current user's username are same with the form field "creator" value. The following sample using Form Hash Variable to retrieve form field value.

```
import java.util.Map;
import org.joget.directory.model.User;

public boolean isAuthorized(User user, Map params) {
    //using hash variable to get "creator" field value and escapes it with java syntax, then compare with
current username
    return "#form.crm_account.creator?java#".equals(user.getUsername());
}

//call isAuthorized method with injected variable
return isAuthorized(user, requestParams);
```

Use as Form Post Submission Processing Tool

Injected Variables:

- workflowAssignment - The workflow activity assignment object of the saving form. Null when the form is not an assignment form. (org.joget.workflow.model.WorkflowAssignment)
- pluginManager - Plugin Manager service bean for convenient usage. (org.joget.plugin.base.PluginManager)
- appDef - App definition of the process. (org.joget.apps.app.model.AppDefinition)
- request - Http Request object of current Http Request. (javax.servlet.http.HttpServletRequest)

Expected Return Object:

- None

Samples:

Reuse Email tool to send separate email to each users. The following script is for a form not mapped to workflow assignment, therefore workflowAssignment is not available.

```
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import org.joget.apps.app.model.AppDefinition;
import org.joget.apps.app.service.AppPluginUtil;
import org.joget.apps.app.service.AppUtil;
import org.joget.plugin.base.ApplicationPlugin;
import org.joget.plugin.base.Plugin;
import org.joget.plugin.base.PluginManager;
import org.joget.plugin.property.model.PropertyEditable;

public Object execute(AppDefinition appDef, HttpServletRequest request) {
    String[] emails = new String[]{"test1@joget.org", "test2@joget.org"};

    //Reuse Email Tool to send separated email to a list of users;
    Plugin plugin = pluginManager.getPlugin("org.joget.apps.app.lib.EmailTool");

    //Get default properties (SMTP setting) for email tool
    Map propertiesMap = AppPluginUtil.getDefaultProperties(plugin, null, appDef, null);
    propertiesMap.put("pluginManager", pluginManager);
    propertiesMap.put("appDef", appDef);
    propertiesMap.put("request", request);

    ApplicationPlugin emailTool = (ApplicationPlugin) plugin;

    //send email
    for (String email : emails) {
        propertiesMap.put("toSpecific", email);
        propertiesMap.put("subject", "This is a test email for " + email);
        propertiesMap.put("message", "Email content for " + email);

        //set properties and execute the tool
        ((PropertyEditable) emailTool).setProperties(propertiesMap);
        emailTool.execute(propertiesMap);
    }

    return null;
}

//call execute method with injected variable
return execute(appDef, request);
```

Use as Datalist Action

Injected Variables:

- httpRequest - HTTP Request object of current HTTP Request. (javax.servlet.http.HttpServletRequest)
- datalist - Datalist object of the current datalist (org.joget.apps.datalist.model.Datalist)
- rowKeys - A String array of record ID(s) for datalist row(s) that is selected for this action

Expected Return Object:

- None

Samples:

```
for (String key : rowKeys) {
        System.out.println("Record key is " + key);
}
```

Use as Datalist Formatter

Note:

- The entire script is defaulted to be applied to every row in the specified column, hence it is not needed to perform looping to manually apply formatting to every row.
- It is also possible to use hash variable in the returned value.

Injected Variables:

- datalist - Datalist object of the current datalist (org.joget.apps.datalist.model.Datalist)
- column - The current datalist column object (org.joget.apps.datalist.model.DataListColumn)
- row - row Object of current record row in the datalist.
  To retrieve the property value from ***Object*** *row*, use this service method: DataListService.evaluateColumnValueFromRow(**Object** row, **String** propertyName)
- value - value of the current row as String

Expected Return Object:

- A String value to replace the original row value

Samples:

1) Example code using DataListService service method to evaluate column value "Name" from row object.

```
//import the necessary classes
import org.joget.apps.datalist.service.DataListService;
import org.joget.apps.app.service.AppUtil;

DataListService dataListService = (DataListService) AppUtil.getApplicationContext().getBean("dataListService");

//since this entire bean shell applies to every row, "row" is automatically iterated here.
//"name" is the column id
return dataListService.evaluateColumnValueFromRow(row, "name");
```

2) Format the row value by appending hash variable.

```
return value + " #currentUser.firstName#";
```

Use as Process Participant

Injected Variables:

- pluginManager - Plugin Manager service bean for convenient usage. (org.joget.plugin.base.PluginManager)
- workflowActivity - Workflow Activity that trying to retrieves assignee. (org.joget.workflow.model.WorkflowActivity)

Expected Return Object:

- A java.util.Collection of username in java.lang.String to be assign to the Workflow Activity.

Samples:

Randomly assign a user in a department to a workflow activity.

```
import java.util.ArrayList;
import java.util.Collection;
import org.joget.apps.app.service.AppUtil;
import org.joget.directory.model.User;
import org.joget.directory.model.service.ExtDirectoryManager;
import org.joget.workflow.model.WorkflowActivity;

public Collection getAssignees(WorkflowActivity activity) {
    Collection assignees = new ArrayList();

    ExtDirectoryManager directoryManager = (ExtDirectoryManager) pluginManager.getBean("directoryManager");

    String deptId = "D-005";

    //Get total user in department
    Long total = directoryManager.getTotalUsers(null, null, deptId, null, null, null, null);

    //Get random number from 0 to the total number of users in department
    int random = (int) (Math.random() * total);

    //Get users using directory manager
    Collection userList = directoryManager.getUsers(null, null, deptId, null, null, null, null, "firstName",
false, random, 1);
    for(Object u : userList){
        User user = (User) u;
        assignees.add(user.getUsername());
    }

    return assignees;
}

//call getAssignees method with injected variable
return getAssignees(workflowActivity);
```

Use as Process Tool

Injected Variables:

- workflowAssignment - The workflow tool activity assignment object. (org.joget.workflow.model.WorkflowAssignment)
- pluginManager - Plugin Manager service bean for convenient usage. (org.joget.plugin.base.PluginManager)
- appDef - App definition of the process. (org.joget.apps.app.model.AppDefinition)
- request - Http Request object of current HTTP Request. Not available if the tool is trigger by Deadline. (javax.servlet.http.HttpServletRequest)

Expected Return Object:

- None

Samples:

Start a new process in the same app with current record id.

```
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import org.joget.apps.app.model.AppDefinition;
import org.joget.apps.app.service.AppService;
import org.joget.apps.app.service.AppUtil;
import org.joget.workflow.model.service.WorkflowManager;
import org.joget.workflow.model.WorkflowAssignment;
import org.joget.workflow.model.WorkflowProcess;

public Object execute(WorkflowAssignment assignment, AppDefinition appDef, HttpServletRequest request) {
    AppService appService = (AppService) AppUtil.getApplicationContext().getBean("appService");
    WorkflowManager workflowManager = (WorkflowManager) AppUtil.getApplicationContext().getBean
("workflowManager");

    //get current record id
    String recordId = appService.getOriginProcessId(assignment.getProcessId());

    //get process
    WorkflowProcess process = appService.getWorkflowProcessForApp(appDef.getAppId(), appDef.getVersion().
toString(), "process2");

    //start process
    workflowManager.processStart(process.getId(), null, null, null, recordId, false);

    return null;
}

//call execute method with injected variable
return execute(workflowAssignment, appDef, request);
```

Update Form Field Value

```
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.workflow.model.WorkflowAssignment;

String formDefId = "hr_expense_new";
String fieldId = "status";
String fieldValue = "Submitted";

//Get record Id from process
AppService appService = (AppService) AppUtil.getApplicationContext().getBean("appService");
String id = appService.getOriginProcessId(workflowAssignment.getProcessId());

//retrieve data
FormRow row = new FormRow();
FormRowSet rowSet = appService.loadFormData(appDef.getAppId(), appDef.getVersion().toString(), formDefId, id);
if (!rowSet.isEmpty()) {
    row = rowSet.get(0);
    //update field value
    row.setProperty(fieldId, fieldValue);
}

//Store the updated data
rowSet.set(0, row);
appService.storeFormData(appDef.getAppId(), appDef.getVersion().toString(), formDefId, rowSet, id);
```

Start a new process instance for each row of data loaded using foreign key and value

```
import org.joget.apps.form.model.FormRow;
import org.joget.apps.form.model.FormRowSet;
import org.joget.apps.app.service.AppUtil;
import org.joget.plugin.base.PluginManager;
import org.joget.apps.form.model.FormLoadBinder;
import org.joget.workflow.model.service.WorkflowManager;

    String formDefId = "ExpensesClaimEntry";  //change this to the form id used to load grid data
    String foreignKey = "claim"; //change this to the foreign key field id

    // Reuse Multi Row Binder to load data
    PluginManager pluginManager = (PluginManager) AppUtil.getApplicationContext().getBean("pluginManager");
    WorkflowManager workflowManager = (WorkflowManager) AppUtil.getApplicationContext().getBean
("workflowManager");
    FormLoadBinder binder = (FormLoadBinder) pluginManager.getPlugin("org.joget.plugin.enterprise.
MultirowFormBinder");

    //Load from the grid table
    binder.setProperty("formDefId", formDefId);
    binder.setProperty("foreignKey", foreignKey);

    FormRowSet rows;
    rows = binder.load(null, "#assignment.processId#", null);

        //loop through records returned
    for (FormRow row : rows) {
        try {
            Map variableMap = new HashMap();
            variableMap.put("status", row.getProperty("purpose"));

                        //start a new process instance with data from each row
            //processStart(String processDefId, String processId, Map<String, String> variables, String
startProcUsername, String parentProcessId, boolean startManually)
            workflowManager.processStart("expenseclaim:latest:process1", null, variableMap, null, row.
getProperty("id"), false);
        } catch (Exception e) {}
    }
```

Use as Process Route Decision

Note:

- By using a decision plugin, the specified route transition rules configured explicitly in the process design will be **ignored**.

Injected Variables:

- appDef - App definition of the process. (org.joget.apps.app.model.AppDefinition)
- pluginManager - Plugin Manager service bean for convenient usage. (org.joget.plugin.base.PluginManager)
- processDefId - The process definition ID of the current process instance
- processId - The process instance ID of the current process instance
- routeId - The ID of the route node currently configured on

Expected Return Object:

- A org.joget.workflow.model.DecisionResult object.

Samples:

Decide to approve or reject a request based on the value of the workflow variable "status".

```
import org.joget.workflow.model.DecisionResult;

//Transition Name to go to
String transitionApprove = "approve";
String transitionReject = "reject";

//Workflow variable ID to set
String workflowVariable = "status";

DecisionResult result = new DecisionResult();

/* Bean Shell Decision supports hash variable, and has access to the process instance context. */
//System.out.println("Value of \'status\' wf-var in this process instance is: " + "#variable.status#");

if ("yes".equalsIgnoreCase("#variable.status#")) {
    result.addTransition(transitionApprove);
    result.setVariable(workflowVariable, "ApprovedByBeanShellDecision");
} else {
    result.addTransition(transitionReject);
    result.setVariable(workflowVariable, "RejectedByBeanShellDecision");
}

/* For use cases requiring multiple node routing, do set boolean below as true. */
//result.setIsAndSplit(true);

return result;
```

Use as Userview Permission

Injected Variables:

- user - User object of current logged in user (org.joget.directory.model.User)
- requestParams - Request parameters map of current HTTP Request (java.util.Map)

Expected Return Object:

- A boolean value to indicate the user is authorized.

Samples:

Check the user is in a group and is an admin user.

```
import java.util.Collection;
import java.util.Map;
import org.joget.apps.app.service.AppUtil;
import org.joget.directory.model.Group;
import org.joget.directory.model.User;
import org.joget.directory.model.service.ExtDirectoryManager;
import org.joget.workflow.model.service.WorkflowUserManager;
import org.joget.workflow.util.WorkflowUtil;
public boolean isAuthorized(User user, Map params) {
    //if no logged in user
    if (user == null) {
        return false;
    }
    //check current user is admin
    boolean isAdmin = WorkflowUtil.isCurrentUserInRole(WorkflowUserManager.ROLE_ADMIN);

    //check current user is in group "G-001"
    boolean inGroup = false;

    ExtDirectoryManager directoryManager = (ExtDirectoryManager) AppUtil.getApplicationContext().getBean
("directoryManager");
    Collection groups = directoryManager.getGroupByUsername(user.getUsername());
    if (groups != null) {
        String groupId = "G-001";
        for (Group g : groups) {
            if (groupId.equals(g.getId())) {
                inGroup = true;
            }
        }
    }

    return isAdmin && inGroup;
}

//call isAuthorized method with injected variable
return isAuthorized(user, requestParams);
```

Best Practices

1. Only import classes that are needed

Do not use wildcard in import statement. It giving a very bad performance in Bean Shell interpreter to search the whole package and loads it in memory.

Don't:

```
import java.util.*;
```

Do:

```
import java.util.Collection;
```

2. Do not need to mention the type of element of a collections

Bean Shell interpreter cannot recognise the element type syntax of collections class like Collection, Map, List, Set and etc.

Don't:

```
Map<String, String> map = new HashMap<String, String>();
```

Do:

```
Map map = new HashMap();
```

3. Indents your script nicely and follows the Java Code Conventions

It will make yours and others life easier to maintain and review the script whenever necessary as Joget Workflow provided flexibility to adapt change quickly.

4. Write your script in functions

If your script is pretty long and some parts of the script are reusable, please make use of function to write your script. It provided better reading and performance.

5. Remember to write some comments

It will helps you and others to understand what is the purpose for the script quickly.

6. Catch the exceptions and give a meaningful message in log.

If you are using a lot of Bean Shell Scripting in your app, a meaningful log message can help you to locate your issue quickly.

Don't

```
try {
    //do something
} catch (Exception e) {
    LogUtil.error("BeanShell", e, "Error executing script");
}
```

Do:

```
try {
    //do something
} catch (Exception e) {
    LogUtil.error("CRM app - Backend userview", e, "Error retrieving user department in Report category
permission");
}
```

7. Consider to make it as a plugin instead of using Bean Shell

If your script need to reuse for multiple times in an app or can be used by others app in future development, please consider to make your Bean Shell script as a plugin instead of copy and paste it multiple times across your app. Bean Shell script is harder to maintain in this case. Imagine that you want to make a change, you will have to update all the places that are using the same script as well. Beside that, a normal plugin implementation will have better performance than a script running by a Bean Shell interpreter.

8. Reuse existing plugins in your code to make it cleaner and easy to maintain

If partial of your script can be done by existing plugins in Joget Workflow, you can reuse the plugin in stead of writting it again.

To reuse a plugin, you can retrieve the plugin using PluginManager, then set it properties and execute it.

Example:

1. Reuse Multi Row Binder plugin to load data
2. Reuse Email Tool to send email

9. How to use 3rd party libraries in Bean Shell

In your webserver, you can add the jar file to your WEB-INF/lib folder and restart the server. Then, you should be able to "import" and use them in your bean shell java code.

More samples

- Booking Recurrence
- File handling in Bean Shell Form Store Binder
- Load Grid Data with Custom Sorting and Filtering
- Manage Environment Variable using Form

- Store Form Field Data to Multiple Tables
- Setting Workflow Variable Value in Process Tool Bean Shell