How to develop a Slack Webhook Tool

In this tutorial, we will follow the guideline for developing a plugin to develop our Slack Webhook Tool plugin. Please also refer to the very first tutorial How to develop a Bean Shell Hash Variable for more details steps.

1. What is the problem?

We want to send message to Slack when something is completed in Joget Workflow.

2. How to solve the problem?

We will develop a Process Tool/ Post Form Submission Processing Plugin to send message to Slack.

3. What is the input needed for your plugin?

To develop a Slack Webhook Tool plugin, we will consider providing the parameters available in Slack Incoming Webhook as our plugin input.

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

When the tool is execute, a message will send to Slack based on the configuration.

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

We can use slack-webhook library to integrate with Slack.

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 the Joget Source Code is version 5.0.1. Please refer to the Guideline for developing a plugin article for other platform commands.

Let's say our folder directory is as follows.

```
- Home
  - joget
    - plugins
    - jw-community
      -5.0.1
```

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 is stored.

Run the following command to create a maven project in "plugins" directory.

```
cd joget/plugins/
~/joget/jw-community/5.0.1/wflow-plugin-archetype/create-plugin.sh org.joget slack_webhook 5.0.1
```

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: : 5.0.0
[INFO] Using property: package = org.joget
Confirm properties configuration:
groupId: org.joget
artifactId: slack_webhook
version: 5.0.0
package: org.joget
Y: : y
```

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

Open the maven project with your favourite IDE. I will be using NetBeans.

7. Just code it!

a. Extending the abstract class of a plugin type

Create a "SlackWebhookTool" class under "org.joget" package. Then, extend the class with **org.joget.plugin.base.DefaultApplicationPlugin** abstract class. Please refer to Process Tool/ Post Form Submission Processing Plugin. We will need to implement **org.joget.plugin.base.PluginWebSupport** interface class as well to provide a send test message button in plugin properties page. Please refer to Web Service Plugin.

b. Implement all the abstract methods

As usual, we have to implement all the abstract methods. We will 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;

import org.joget.apps.app.service.AppPluginUtil;
import org.joget.apps.app.service.AppUtil;
import org.joget.plugin.base.DefaultApplicationPlugin;
import org.joget.plugin.base.PluginWebSupport;

public class SlackWebhookTool extends DefaultApplicationPlugin implements PluginWebSupport {
    private final static String MESSAGE_PATH = "message/SlackWebhookTool";

    @Override
    public String getName() {
        return "Slack Webhook Tool";
    }
    @Override
    public String getVersion() {
        return "5.0.0";
    }

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

    @Override
    public String getLabel() {
        //support i18n
        return AppPluginUtil.getMessage("org.joget.SlackWebhookTool.pluginLabel", getClassName(), MESSAGE_PATH);
    }
    @Override
    public String getDescription() {
        //support i18n
        return AppPluginUtil.getMessage("org.joget.SlackWebhookTool.pluginDesc", getClassName(), MESSAGE_PATH);
    }

    @Override
    public String getPropertyOptions() {
        return AppUtil.readPluginResource(getClass().getName(), "/properties/slackWebhookTool.json", null,
true, MESSAGE_PATH);
    }
}
```

Now, we have to create 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/slackWebhookTool.json". Let us create a directory "resources/properties" under "slack_webhook/src/main" directory. After creating the directory, create a file named "slackWebhookTool.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 : '@@SlackWebhookTool.config@@',
    properties : [{
        name : 'url',
        label : '@@SlackWebhookTool.url@@',
        type : 'textfield',
        required : 'true'
    },
    {
        label : '@@SlackWebhookTool.from@@',
        type : 'header'
    },
    {
        name : 'username',
        label : '@@SlackWebhookTool.fromUsername@@',
        type : 'textfield',
        value : '@@SlackWebhookTool.fromUsername.value@@'
    },
    {
```

```
        name : 'customIcon',
        label : '@@SlackWebhookTool.customIcon@@',
        type : 'selectbox',
        value : 'joget',
        options : [{
            value : '',
            label : '@@SlackWebhookTool.customIcon.none@@'
        },
        {
            value : 'joget',
            label : '@@SlackWebhookTool.customIcon.joget@@'
        },
        {
            value : 'url',
            label : '@@SlackWebhookTool.customIcon.url@@'
        },
        {
            value : 'emoji',
            label : '@@SlackWebhookTool.customIcon.emoji@@'
        }]
    },
    {
        name : 'iconUrl',
        label : '@@SlackWebhookTool.customIcon.url@@',
        type : 'textfield',
        required : 'true',
        control_field: 'customIcon',
        control_value: 'url',
        control_use_regex: 'false'
    },
    {
        name : 'iconEmoji',
        label : '@@SlackWebhookTool.customIcon.emoji@@',
        type : 'textfield',
        required : 'true',
        control_field: 'customIcon',
        control_value: 'emoji',
        control_use_regex: 'false'
    },
    {
        label : '@@SlackWebhookTool.to@@',
        type : 'header'
    },
    {
        name : 'channels',
        label : '@@SlackWebhookTool.channels@@',
        description : '@@SlackWebhookTool.channels.desc@@',
        type : 'textfield'
    },
    {
        name : 'participantIds',
        label : '@@SlackWebhookTool.participantIds@@',
        description : '@@SlackWebhookTool.participantIds.desc@@',
        type : 'textfield'
    },
    {
        name : 'usernames',
        label : '@@SlackWebhookTool.usernames@@',
        description : '@@SlackWebhookTool.usernames.desc@@',
        type : 'textfield'
    },
    {
        name : 'usernameTransform',
        label : '@@SlackWebhookTool.usernameTransform@@',
        description : '@@SlackWebhookTool.usernameTransform.desc@@',
        type : 'textfield',
        value : '@@SlackWebhookTool.usernameTransform.value@@',
        required : 'True'
    },
    {
        label : '@@SlackWebhookTool.message@@',
```

```
        type : 'header'
    },
    {
        name : 'text',
        label : '@@SlackWebhookTool.text@@',
        description : '@@SlackWebhookTool.text.desc@@',
        type : 'codeeditor',
        required : 'True'
    },
    {
        name : 'unfurl_links',
        label : '@@SlackWebhookTool.unfurl_links@@',
        description : '@@SlackWebhookTool.unfurl_links.desc@@',
        type : 'checkbox',
        value : 'true',
        options : [{
            value : 'true',
            label : ''
        }]
    },
    {
        name : 'unfurl_media',
        label : '@@SlackWebhookTool.unfurl_media@@',
        description : '@@SlackWebhookTool.unfurl_media.desc@@',
        type : 'checkbox',
        value : 'true',
        options : [{
            value : 'true',
            label : ''
        }]
    }],
    buttons : [{
        name : 'sendTestMessage',
        label : '@@SlackWebhookTool.sendTestMessage@@',
        ajax_url : '[CONTEXT_PATH]/web/json/app[APP_PATH]/plugin/org.joget.SlackWebhookTool/service?
action=sendTestMessage',
        fields : ['url'],
        addition_fields : [
            {
                name : 'testChannel',
                label : '@@SlackWebhookTool.sendTestMessage.testChannel@@',
                type : 'textfield'
            }
        ]
    }]
},
{
    title : '@@SlackWebhookTool.attachment@@',
    properties : [{
        name:'fallback',
        label:'@@SlackWebhookTool.attachment.fallback@@',
        description:'@@SlackWebhookTool.attachment.fallback.desc@@',
        type:'textfield',
        value: '@@SlackWebhookTool.attachment.fallback.value@@',
        required : 'True'
    },
    {
        name:'color',
        label:'@@SlackWebhookTool.attachment.color@@',
        description:'@@SlackWebhookTool.attachment.color.desc@@',
        type:'textfield'
    },
    {
        name:'pretext',
        label:'@@SlackWebhookTool.attachment.pretext@@',
        description:'@@SlackWebhookTool.attachment.pretext.desc@@',
        type:'textfield'
    },
    {
        name:'author_name',
        label:'@@SlackWebhookTool.attachment.author_name@@',
```

```
    description:'@@SlackWebhookTool.attachment.author_name.desc@@',
    type:'textfield'
},
{
    name:'author_link',
    label:'@@SlackWebhookTool.attachment.author_link@@',
    description:'@@SlackWebhookTool.attachment.author_link.desc@@',
    type:'textfield'
},
{
    name:'author_icon',
    label:'@@SlackWebhookTool.attachment.author_icon@@',
    description:'@@SlackWebhookTool.attachment.author_icon.desc@@',
    type:'textfield'
},
{
    name:'attachment_title',
    label:'@@SlackWebhookTool.attachment.title@@',
    description:'@@SlackWebhookTool.attachment.title.desc@@',
    type:'textfield'
},
{
    name:'attachment_title_link',
    label:'@@SlackWebhookTool.attachment.title_link@@',
    description:'@@SlackWebhookTool.attachment.title_link.desc@@',
    type:'textfield'
},
{
    name:'attachment_text',
    label:'@@SlackWebhookTool.attachment.text@@',
    description:'@@SlackWebhookTool.attachment.text.desc@@',
    type:'textfield'
},
{
    name:'fields',
    label:'@@SlackWebhookTool.attachment.fields@@',
    description : '@@SlackWebhookTool.attachment.fields.desc@@',
    type:'grid',
    columns:[{
        key:'title',
        label:'@@SlackWebhookTool.attachment.fields.title@@'
    },
    {
        key:'value',
        label:'@@SlackWebhookTool.attachment.fields.value@@'
    },
    {
        key:'short',
        label:'@@SlackWebhookTool.attachment.fields.short@@',
        options: [
        {
            value : 'false',
            label : '@@SlackWebhookTool.attachment.fields.short.no@@'
        },{
            value : 'true',
            label : '@@SlackWebhookTool.attachment.fields.short.yes@@'
        }]
    }]
},
{
    name:'image_url',
    label:'@@SlackWebhookTool.attachment.image_url@@',
    description:'@@SlackWebhookTool.attachment.image_url.desc@@',
    type:'textfield'
},
{
    name:'thumb_url',
    label:'@@SlackWebhookTool.attachment.thumb_url@@',
    description:'@@SlackWebhookTool.attachment.thumb_url.desc@@',
    type:'textfield'
}]
```

```
}]
```

After completing the properties option to collect input, we can work on the main methods of the plugin which is execute method.

```
    private SlackApi api = null;

    @Override
    public Object execute(Map props) {
        final Set<String> channelList = new HashSet<String>();

        String channels = getPropertyString("channels");
        if (!channels.isEmpty()) {
            String[] temp = channels.split(";");
            for (String t : temp) {
                if (t.startsWith("#")) {
                    channelList.add(t);
                }
            }
        }

        WorkflowAssignment wfAssignment = (WorkflowAssignment) getProperty("workflowAssignment");

        String participantIds = getPropertyString("participantIds");
        if (wfAssignment != null && !participantIds.isEmpty()) {
            String[] temp = participantIds.split(";");
            if (temp.length > 0) {
                WorkflowManager workflowManager = (WorkflowManager) AppUtil.getApplicationContext().getBean
("workflowManager");
                WorkflowProcess process = workflowManager.getProcess(wfAssignment.getProcessDefId());
                for (String pid : temp) {
                    pid = pid.trim();
                    if (!pid.isEmpty()) {
                        Collection<String> userList = WorkflowUtil.getAssignmentUsers(process.getPackageId(),
wfAssignment.getProcessDefId(), wfAssignment.getProcessId(), wfAssignment.getProcessVersion(), wfAssignment.
getActivityId(), "", pid);
                        if (userList != null && userList.size() > 0) {
                            for (String username : userList) {
                                channelList.add(getSlackUsername(username, wfAssignment));
                            }
                        }
                    }
                }
            }
        }

        String usernames = getPropertyString("usernames");
        if (!usernames.isEmpty()) {
            String[] temp = usernames.split(";");
            for (String username : temp) {
                channelList.add(getSlackUsername(username, wfAssignment));
            }
        }

        Thread sendMessageThread = new PluginThread(new Runnable() {
            public void run() {
                try {
                    LogUtil.info(SlackWebhookTool.class.getName(), "SlackWebhookTool: Sending message to=" +
channelList.toString());
                    sendMessage(channelList);
                    LogUtil.info(SlackWebhookTool.class.getName(), "SlackWebhookTool: Sending message completed
to=" + channelList.toString());
                } catch (Exception ex) {
                    LogUtil.error(SlackWebhookTool.class.getName(), ex, "");
                }
            }
        });

        sendMessageThread.setDaemon(true);
        sendMessageThread.start();
```

```
        return null;
    }

    protected SlackApi getApi() {
        if (api == null) {
            api = new SlackApi(getPropertyString("url"));
        }
        return api;
    }

    protected String getSlackUsername(String username, WorkflowAssignment assignment) {
        String syntax = getPropertyString("usernameTransform");
        syntax = syntax.replaceAll(StringUtil.escapeRegex("{username}"), StringUtil.escapeRegex(username));
        return AppUtil.processHashVariable(syntax, assignment, null, null);
    }

    protected void sendMessage(Set<String> channels) {
        SlackMessage message = createMessage();
        if (channels != null && !channels.isEmpty()) {
            for (String channel : channels) {
                if (!channel.isEmpty()) {
                    sendMessage(channel, message);
                }
            }
        } else {
            sendMessage(null, message);
        }
    }

    protected void sendMessage(String channel, SlackMessage message) {
        if (channel != null && !channel.isEmpty()) {
            message.setChannel(channel);
        }

        getApi().call(message);
    }

    protected SlackMessage createMessage() {
        SlackMessage message = new SlackMessage(getPropertyString("text"));

        String username = getPropertyString("username");
        if (!username.isEmpty()) {
            message.setUsername(username);
        }

        String customIcon = getPropertyString("customIcon");
        if (!customIcon.isEmpty()) {
            if ("joget".equals(customIcon)) {
                HttpServletRequest request = WorkflowUtil.getHttpServletRequest();
                if (request != null) {
                    String url = request.getScheme() + "://" + request.getServerName() + ":" + request.
getServerPort() + request.getContextPath() + "/images/v3/logo.png";
                    message.setIcon(url);
                }
            } else if ("url".equals(customIcon)) {
                message.setIcon(getPropertyString("iconUrl"));
            } else {
                message.setIcon(getPropertyString("iconEmoji"));
            }
        }

        message.setUnfurlLinks("true".equalsIgnoreCase(getPropertyString("unfurl_links")));
        message.setUnfurlMedia("true".equalsIgnoreCase(getPropertyString("unfurl_media")));

        boolean hasAttachment = false;
        SlackAttachment atatchment = new SlackAttachment();

        String fallback = getPropertyString("fallback");
        if (!fallback.isEmpty()) {
            atatchment.setFallback(fallback);
```

```
}

String color = getPropertyString("color");
if (!color.isEmpty()) {
    atatchment.setColor(color);
}

String pretext = getPropertyString("pretext");
if (!pretext.isEmpty()) {
    atatchment.setPretext(pretext);
    hasAttachment = true;
}

String author_name = getPropertyString("author_name");
if (!author_name.isEmpty()) {
    atatchment.setAuthorName(author_name);
}

String author_link = getPropertyString("author_link");
if (!author_link.isEmpty()) {
    atatchment.setAuthorLink(author_link);
}

String author_icon = getPropertyString("author_icon");
if (!author_icon.isEmpty()) {
    atatchment.setAuthorIcon(author_icon);
}

String attachment_title = getPropertyString("attachment_title");
if (!attachment_title.isEmpty()) {
    atatchment.setTitle(attachment_title);
    hasAttachment = true;
}

String attachment_title_link = getPropertyString("attachment_title_link");
if (!attachment_title_link.isEmpty()) {
    atatchment.setTitleLink(attachment_title_link);
}

String attachment_text = getPropertyString("attachment_text");
if (!attachment_text.isEmpty()) {
    atatchment.setText(attachment_text);
    hasAttachment = true;
}

String image_url = getPropertyString("image_url");
if (!image_url.isEmpty()) {
    atatchment.setImageUrl(image_url);
    hasAttachment = true;
}

String thumb_url = getPropertyString("thumb_url");
if (!thumb_url.isEmpty()) {
    atatchment.setThumbUrl(thumb_url);
    hasAttachment = true;
}

Object[] fields = null;
Object fieldsProperty = getProperty("fields");
if (fieldsProperty != null && fieldsProperty instanceof Object[]){
    fields = (Object[]) fieldsProperty;
}

if (fields != null && fields.length > 0) {
    for (Object o : fields) {
        Map mapping = (HashMap) o;
        String title = mapping.get("title").toString();
        String value = mapping.get("value").toString();
        String isShort = mapping.get("short").toString();
        SlackField field = new SlackField();
        field.setTitle(title);
```

```
                field.setValue(value);
                field.setShorten("true".equalsIgnoreCase(isShort));

                atatchment.addFields(field);
                hasAttachment = true;
            }
        }

        if (hasAttachment) {
            message.addAttachments(atatchment);
        }

        return message;
    }
```

In our plugin properties, we have a button to send test message. Let implement the webService method to provide an API to send test message.

```
    public void webService(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
        boolean isAdmin = WorkflowUtil.isCurrentUserInRole(WorkflowUserManager.ROLE_ADMIN);
        if (!isAdmin) {
            response.sendError(HttpServletResponse.SC_UNAUTHORIZED);
            return;
        }

        String action = request.getParameter("action");
        if ("sendTestMessage".equals(action)) {
            String message = "";
            try {
                AppDefinition appDef = AppUtil.getCurrentAppDefinition();

                String url = AppUtil.processHashVariable(request.getParameter("url"), null, null, null, appDef);
                String testChannel = AppUtil.processHashVariable(request.getParameter("testChannel"), null,
null, null, appDef);

                setProperty("url", url);
                setProperty("text", AppPluginUtil.getMessage("SlackWebhookTool.testMessage", getClassName(),
MESSAGE_PATH));

                if (testChannel != null && !testChannel.isEmpty()) {
                    Set<String> channels = new HashSet<String>();
                    channels.add(testChannel);
                    sendMessage(channels);
                } else {
                    sendMessage(null);
                }

                message = AppPluginUtil.getMessage("SlackWebhookTool.sendTestMessage.success", getClassName(),
MESSAGE_PATH);
            } catch (Exception e) {
                LogUtil.error(this.getClassName(), e, "Fail to send Test Message to Slack");
                message = AppPluginUtil.getMessage("SlackWebhookTool.sendTestMessage.fail", getClassName(),
MESSAGE_PATH) + "\n" + StringEscapeUtils.escapeJavaScript(e.getMessage());
            }
            try {
                JSONObject jsonObject = new JSONObject();
                jsonObject.accumulate("message", message);
                jsonObject.write(response.getWriter());
            } catch (Exception e) {
                //ignore
            }
        } else {
            response.setStatus(HttpServletResponse.SC_NO_CONTENT);
        }
    }
```

c. Manage the dependency libraries of your plugin

We need to include "jsp-api" and "slack-webhook" libraries in our POM file.

```
        <!-- Change plugin specific dependencies here -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jsp-api</artifactId>
            <version>2.0</version>
        </dependency>
        <dependency>
            <groupId>net.gpedro.integrations.slack</groupId>
            <artifactId>slack-webhook</artifactId>
            <version>1.1.1</version>
        </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 will use i18n message key in our properties options definition as well. Then, we will need to create a message resource bundle properties file for our plugin.

Create a directory, "resources/message", under "slack_webhook/src/main" directory. Then, create a "SlackWebhookTool.properties" file in the folder. In the properties file, add all the message keys and its label as below.

```
org.joget.SlackWebhookTool.pluginLabel=Slack Webhook Tool
org.joget.SlackWebhookTool.pluginDesc=Used to send message to Slack through incoming webhook
SlackWebhookTool.config=Configure Slack Webhook Tool
SlackWebhookTool.url=Webhook URL
SlackWebhookTool.from=From
SlackWebhookTool.fromUsername=Username
SlackWebhookTool.fromUsername.value=Joget Workflow
SlackWebhookTool.customIcon=Custom Icon
SlackWebhookTool.customIcon.none=None
SlackWebhookTool.customIcon.joget=Joget Workflow Logo
SlackWebhookTool.customIcon.url=Image URL
SlackWebhookTool.customIcon.emoji=Emoji Code
SlackWebhookTool.to=To
SlackWebhookTool.channels=Channels
SlackWebhookTool.channels.desc=Separated by semicolon(;). E.g. #general;#info
SlackWebhookTool.participantIds=Participant Ids
SlackWebhookTool.participantIds.desc=Separated by semicolon(;). E.g. applicant;approver
SlackWebhookTool.usernames=Usernames
SlackWebhookTool.usernames.desc=Separated by semicolon(;). E.g. admin;cat
SlackWebhookTool.usernameTransform=Transform username to Slack username
SlackWebhookTool.usernameTransform.desc=Hash Variable can be used to transform username to Slack username. Eg.
@#form.slack.username[{username}]#
SlackWebhookTool.usernameTransform.value=@{username}
SlackWebhookTool.message=Message
SlackWebhookTool.text=Text
SlackWebhookTool.text.desc=Refer to <a href="https://api.slack.com/docs/formatting" target="_blank">Slack
Message Formatting</a>.
SlackWebhookTool.unfurl_links=Unfurling Links
SlackWebhookTool.unfurl_links.desc=Automatically find URLs in a message and create attachments based on the
content of those URLs
SlackWebhookTool.unfurl_media=Unfurling Media
SlackWebhookTool.unfurl_media.desc=Automatically find Media URLs in a message and create attachments based on
the media of those URLs
SlackWebhookTool.attachment=Attachment
SlackWebhookTool.attachment.fallback=Fallback
SlackWebhookTool.attachment.fallback.desc=A plain-text summary of the attachment. This text will be used in
clients that don't show formatted text (eg. IRC, mobile notifications) and should not contain any markup.
SlackWebhookTool.attachment.fallback.value=Attachment fail to display.
SlackWebhookTool.attachment.color=Color
SlackWebhookTool.attachment.color.desc=An optional value that can either be one of good, warning, danger, or
any hex color code (eg. #439FE0). This value is used to color the border along the left side of the message
attachment.
SlackWebhookTool.attachment.pretext=Pretext
SlackWebhookTool.attachment.pretext.desc=This is optional text that appears above the message attachment block.
SlackWebhookTool.attachment.author_name=Author Name
SlackWebhookTool.attachment.author_name.desc=Small text used to display the author's name.
```

```
SlackWebhookTool.attachment.author_link=Author Link
SlackWebhookTool.attachment.author_link.desc=A valid URL that will hyperlink the Author Name text mentioned
above. Will only work if Author Name is present.
SlackWebhookTool.attachment.author_icon=Author Icon
SlackWebhookTool.attachment.author_icon.desc=A valid URL that displays a small 16x16px image to the left of the
Author Name text. Will only work if Author Name is present.
SlackWebhookTool.attachment.title=Attachment Title
SlackWebhookTool.attachment.title.desc=The Attachment Title is displayed as larger, bold text near the top of a
message attachment.
SlackWebhookTool.attachment.title_link=Attachment Title Link
SlackWebhookTool.attachment.title_link.desc=By passing a valid URL in the Attachment Title Link parameter
(optional), the Attachment Title text will be hyperlinked.
SlackWebhookTool.attachment.text=Text
SlackWebhookTool.attachment.text.desc=This is the main text in a message attachment, and can contain standard
message markup. The content will automatically collapse if it contains 700+ characters or 5+ linebreaks, and
will display a "Show more..." link to expand the content.
SlackWebhookTool.attachment.fields=Fields
SlackWebhookTool.attachment.fields.desc=Fields will be displayed in a table inside the message attachment.
SlackWebhookTool.attachment.fields.title=Title
SlackWebhookTool.attachment.fields.value=Value
SlackWebhookTool.attachment.fields.short=Short
SlackWebhookTool.attachment.fields.short.no=No
SlackWebhookTool.attachment.fields.short.yes=Yes
SlackWebhookTool.attachment.image_url=Image URL
SlackWebhookTool.attachment.image_url.desc=A valid URL to an image file that will be displayed inside a message
attachment. We currently support the following formats: GIF, JPEG, PNG, and BMP. Large images will be resized
to a maximum width of 400px or a maximum height of 500px, while still maintaining the original aspect ratio.
SlackWebhookTool.attachment.thumb_url=Thumbnail URL
SlackWebhookTool.attachment.thumb_url.desc=A valid URL to an image file that will be displayed as a thumbnail
on the right side of a message attachment. We currently support the following formats: GIF, JPEG, PNG, and BMP.
The thumbnail's longest dimension will be scaled down to 75px while maintaining the aspect ratio of the image.
The filesize of the image must also be less than 500 KB.
SlackWebhookTool.sendTestMessage=Send Test Message
SlackWebhookTool.sendTestMessage.testChannel=Test Channel
SlackWebhookTool.sendTestMessage.success=Test message sent.
SlackWebhookTool.sendTestMessage.fail=Fail to sent test message. Error:
SlackWebhookTool.testMessage=Test Message
```

e. Register your plugin to the Felix Framework

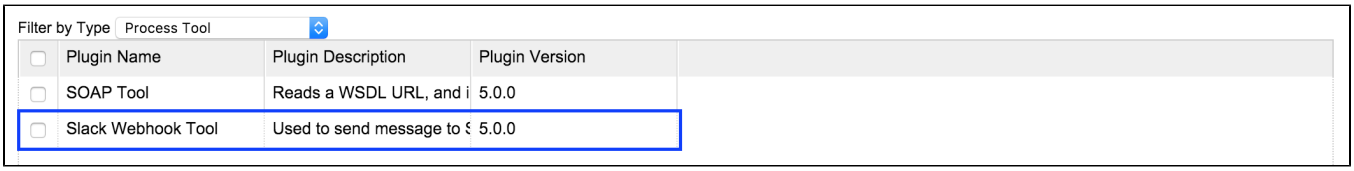
Next, we will have to register our plugin class in the Activator class (Auto generated in the same class package) to tell the Felix Framework that this is a plugin.

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

f. Build it and test

Let's build our plugin. Once the building process is done, we will find a "slack_webhook-5.0.0.jar" file created under "slack_webhook/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.

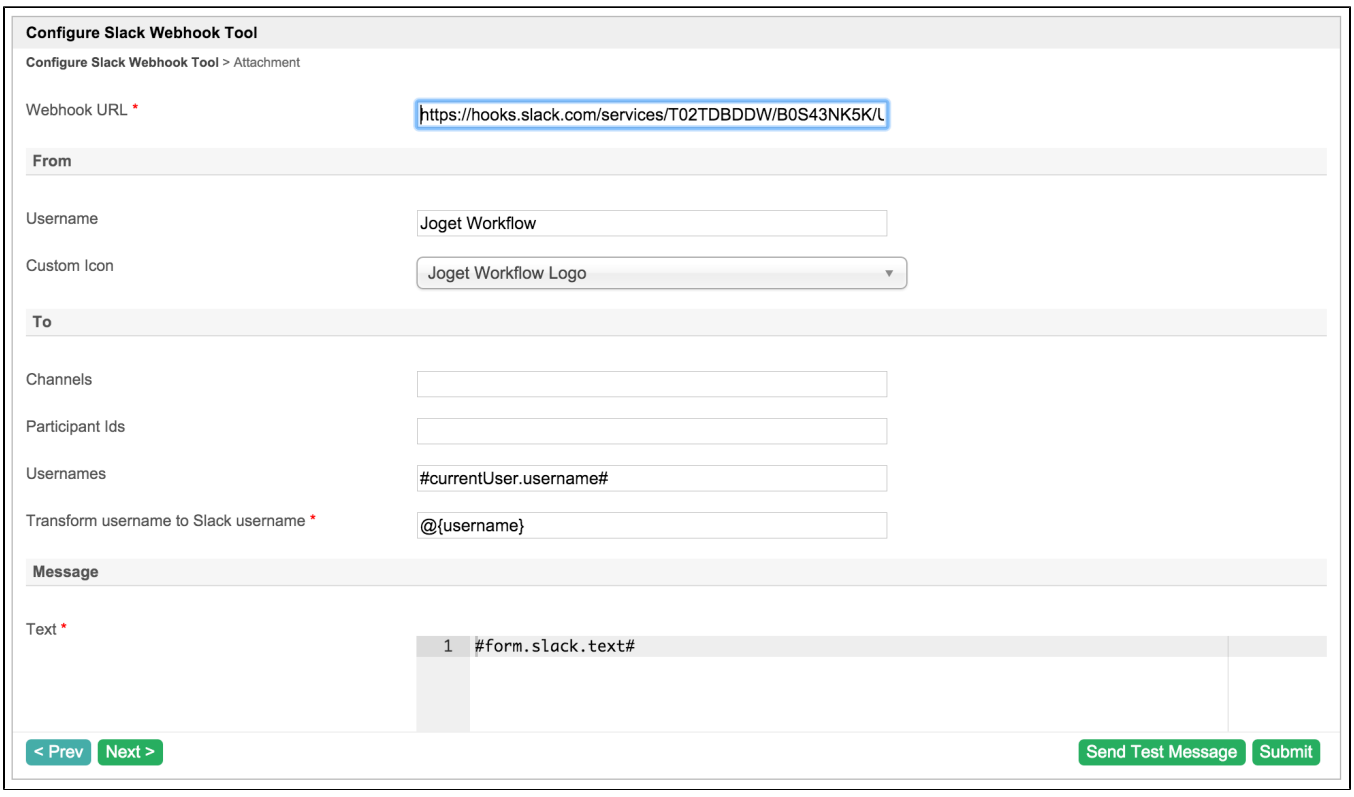


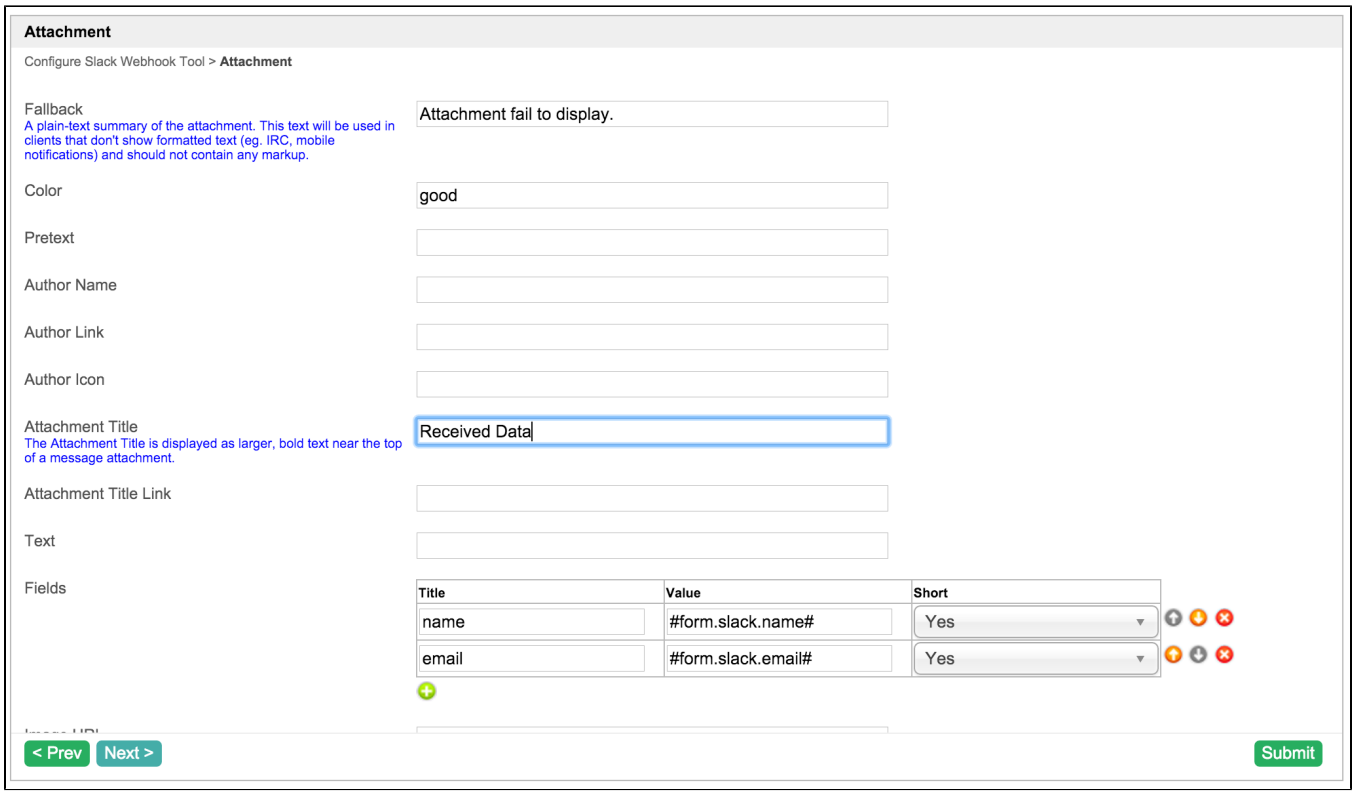
Check the Slack Webhook Tool is available in process tool mapping.

Now, let us configure the Incoming Webhooks in Slack platform.

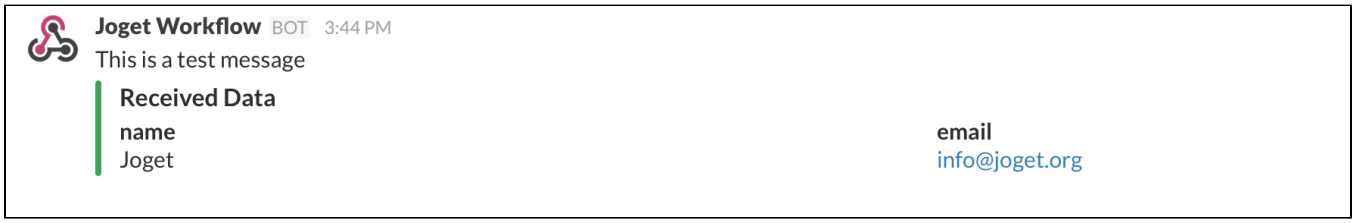
1. Go to *your_team.slack.com/services/new*.
2. Search for *Incoming WebHook* and click in `Add`
3. Choose Channel to Post and press `Add Incoming WebHooks Integration`
4. Into *Setup Instructions*, you've a WebHook URL. This is the argument will use for the "Webhook URL" later. Then, copy it.

Configure the Slack Webhook Tool.





The message is received in Slack.



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

You can download the source code from slack_webhook_src.zip.

To download the ready-to-use plugin jar, please find it in http://marketplace.joget.org/. (Coming Soon)