Using Helper Tools from RSF

IN PROGRESS

- Overview
- Strategy
- Example: Permissions Helper
- Example: Attachments Helper
- Helper Theory
- Writing Helpers in RSF

Overview

This is a small guide to using Sakai Helpers in your RSF tools. Helpers are things like the page most tools use for adding attachments to their items. The Strategy section covers the basic methodology and contains a concise list of steps to use a Helper from your code. There are two examples so far, the Permissions and Attachments Helpers.

The theory section describes how Helpers actually work. You can safely skip this if you just want to use them from your RSF tools. Like all things, the information and code contained here are always changing and evolving. We will most likely be adding nicer scaffolding in the future for using Helpers in RSF, but even then, these existing examples should still work with little or no modification.

Lastly, there is a section on using RSF to write your own helper.

Strategy

A Helper tool in Sakai allows you to outsource part of your work to another tool built for handling that work. The Helper will take over the entire screen of your tool, perform it's work, and then return the window back to your tool. Typically you will pass it some information, and it can pass some back to you. This is done by storing data in the current ToolSession.

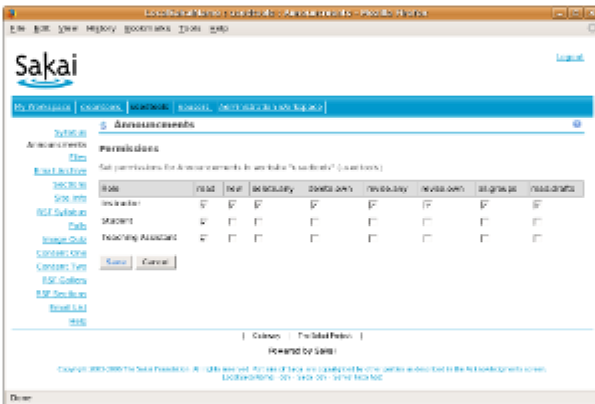
To use a Helper in RSF, you go about it much as if you were adding a normal page to your application. You create a ViewComponentProducer, a Template, and perhaps some actions. The Helper then fills in for that page.

The following steps are a general outline of using a Helper from your RSF tool:

1. Add a ViewComponentProducer and HTML Template for the view that you want filled in by the Helper. These will be slightly different than usual, and contain a few special Helper semantics.
2. The ViewComponentProducer must report as accepting a ViewParamaters of type HelperViewParameters. It's fine to make your own ViewParameters class , but it must extend HelperViewParameters at some point.
3. Determine what parameters the helpers accepts and put them in the ToolSession. (May be replaced with better system in future)
4. Add a request scope <bean> entry for the View Producer as usual.
5. If you need to perform some action on return of the Helper, make a bean to perform the action as you usually would for a POST form or anything else that alters server state. In the case of the Attachments Helper, you will probably want to do something with the attached files. In the case of the Permissions Helper you may not want to do anything.
6. That's it.

Example: Permissions Helper

TODO: Commit Permissions example to the sakai.rsfgallery



The permissions helper allows you to set all the permissions with a given prefix. In the above example the prefix in "annc". You can also set part of the instruction text that appears.

The following example code shows how to use the Permissions Helper for the Mail Archive Permissions:

AssignListPermissions.html

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>
<body>
  <h3>Email List Permissions Helper Stub</h3>
  <input type="text" rsf:id="helper-id" />
  <input type="submit" rsf:id="helper-binding" />
</body>
</html>
```

As always in RSF you start out with a template. However, when using a Helper, this is more of a "fake" view. During excecution, the Helper is going to take over for this view. At the moment, all helper templates need to correspond closely to this. The base requirement is that there are 2 inputs in the body. One of type text with rsf:id helper-id, and one of type submit with rsf:id helper-binding. These don't need to actually be in a form. Again, this is really a stub that will be replaced by the Helper.

ViewComponentProducer.java

```
public class AssignListPermissionsProducer implements
  ViewComponentProducer,
  ViewParamsReporter,
  NavigationCaseReporter
{
  public static final String HELPER = "sakai.permissions.helper";
  public static final String VIEWID = "AssignListPermissions";

  // Injection
  public SessionManager sessionManager;
  public Site site;
  public MessageLocator messageLocator;

  public String getViewID() {
    return VIEWID;
  }

  public void fillComponents(UIContainer tofill, ViewParameters viewparams, ComponentChecker checker) {
    ToolSession session = sessionManager.getCurrentToolSession();

    session.setAttribute(PermissionsHelper.TARGET_REF, site.getReference());
    session.setAttribute(PermissionsHelper.DESCRIPTION, "Set mail permissions for " +  site.getTitle());
    session.setAttribute(PermissionsHelper.PREFIX, "mail.");

    UIOutput.make(tofill, HelperViewParameters.HELPER_ID, HELPER);
    UICommand.make(tofill, HelperViewParameters.POST_HELPER_BINDING, "", null);
  }

  public ViewParameters getViewParameters() {
    return new HelperViewParameters();
  }

  public List reportNavigationCases() {
    List l = new ArrayList();
    l.add(new NavigationCase(null, new SimpleViewParameters(ViewListProducer.VIEWID)));
    return l;
  }

}
```

Above is the producer code. As usual, the ViewID should correspond with the template name (even if it's just a tricksy fake template). Note that we report as accepting HelperViewParameters. This is what tells SakaiRSF that we are a stub for a Sakai Helper. If you don't report HelperViewParameters (or a subclass of it), RSF will try to render this as a normal page.

Next, we grab the ToolSession and fill it with the initial parameters for the helper. The names and expected values will vary from Helper to Helper. For the Permissions Helper we set some instruction text and the prefix of the permissions it should handle.

After that, it's important to bind the 2 special fields from the Template. The Helper ID should be the ID of the helper, in this case sakai. permissions. helper. You can find a Helper's ID in it's tool registration. For the permissions helper, the registration file is conveniently located at webapps/sakai-authz-tool/tools/sakai.permissions.helper.xml in the installed Tomcat.

The second special field is the method binding. This method binding will be called when the helper is finished. For this case, I don't want anything special to happen so I pass null in as the last parameter. We will make more use of this when using the Attachments Helper.

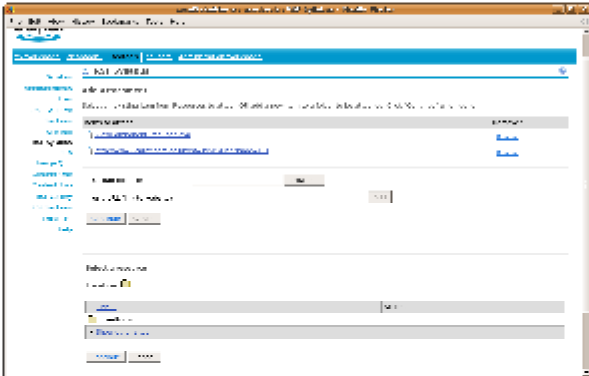
ViewProducers used for Helpers can take full advantage of Navigation Cases and ActionResultInterceptors as usual. In this example, we tell it return to the main ViewListProducer when it's done.

RequestContext.xml

```
<bean class="org.sakaiproject.tool.maillist.producers.AssignListPermissionsProducer">
  <property name="sessionManager" ref="org.sakaiproject.tool.api.SessionManager" />
  <property name="site" ref="sakai-Site" />
  <property name="messageLocator" ref="messageLocator" />
</bean>
```

As usual, you have to register your ViewComponentProducer as a bean. Nothing special or different here.

Example: Attachments Helper



Using the Attachments Helper follows the same formula. The only major difference here is that we are specifying an action to be executed when the Helper finishes. TODO: Commit updates and Link to Syllabus or Gallery Example

Again, we start with the template. All Helper templates will look the same and have the 2 special inputs as before:

AddAttachment.html

```
<html>
<head>
</head>
<body>
<h1>Fake File Picker Template</h1>
  <input type="text" rsf:id="helper-id" value="" />
  <input type="submit" rsf:id="helper-binding" value="" />
</body>
</html>
```

Then the producer:

AttachmentsHelperProducer.java

```
public class AttachmentsHelperProducer implements
  ViewComponentProducer,
  ViewParamsReporter,
  DynamicNavigationCaseReporter,
  ActionResultInterceptor
{
  public static final String VIEWID = "AddAttachment";

  public String getViewID() {
    return VIEWID;
  }

  public void fillComponents(UIContainer tofill, ViewParameters viewparams, ComponentChecker checker) {
    AttachmentsHelperParams params = (AttachmentsHelperParams) viewparams;

    UIOutput.make(tofill, HelperViewParameters.HELPER_ID, "sakai.filepicker");
    UICommand goattach = UICommand.make(tofill, HelperViewParameters.POST_HELPER_BINDING, "fileAttachments.
process");
    goattach.parameters = new ParameterList();
    goattach.parameters.add(new UIELBinding("fileAttachments.syllabusdataid", params.syllabusdataid));
  }

  public ViewParameters getViewParameters() {
    return new AttachmentsHelperParams();
  }

  public List reportNavigationCases() {
    List l = new ArrayList();
    l.add(new NavigationCase("processed", new EditSyllabusDataParams()));
    return l;
  }

  public void interceptActionResult(ARIResult result, ViewParameters incoming, Object actionReturn) {
    EditSyllabusDataParams resultview = (EditSyllabusDataParams) result.resultingView;
    AttachmentsHelperParams incomingview = (AttachmentsHelperParams) incoming;
    resultview.syllabusDataId = incomingview.syllabusdataid;
  }

}
```

The usual checklist applies:

- ViewID name corresponds with the Template name
- We report to RSF that we accept HelperViewParameters. In this case AttachmentsHelperParams subclasses HelperViewParameters.
- We bind the Helper ID <input> to sakai.filepicker. This one is declared at webapps/sakai-content-tool/tools/sakai.filepicker.xml in your installed Tomcat.
- We take advantage of Navigation Cases and a ActionResultInterceptor to return to a custom view afterword. The use of these is the same for non Helper Views so their usage can be discussed elsewhere.

In this example we want to handle any attached files, so we put in our action method for the Helper Binding. We also attach some Parameters to this binding. It is important to note that full range of UICommand semantics and decorators are not supported for Helper Bindings. If you find you need more functionality beyond a method binding and optional binding parameters, please leave a comment below.

ProcessFileAttachments.java

```
public class ProcessFileAttachments {
  //Injection
  public SessionManager sessionManager;
  public SyllabusManager syllabusManager;
  public SyllabusAttachments attachments;
  public SyllabusPermissions syllabusPermissions;
  public String syllabusdataid;

  public String process() {
    if (!syllabusPermissions.canUpdate())
      return "processed"; // Should throw something

    ToolSession session = sessionManager.getCurrentToolSession();
    if (session.getAttribute(FilePickerHelper.FILE_PICKER_CANCEL) == null &&
        session.getAttribute(FilePickerHelper.FILE_PICKER_ATTACHMENTS) != null)
    {
      List refs = (List)session.getAttribute(FilePickerHelper.FILE_PICKER_ATTACHMENTS);
      for (int i = 0; i < refs.size(); i++) {
        Reference ref = (Reference) refs.get(i);
        SyllabusAttachment thisAttach = syllabusManager.createSyllabusAttachmentObject(
          ref.getId(), ref.getProperties().getProperty(ref.getProperties().getNamePropDisplayName()));
        attachments.addAttachmentToSyllabii(syllabusdataid, thisAttach);
      }

    }
    session.removeAttribute(FilePickerHelper.FILE_PICKER_ATTACHMENTS);
    session.removeAttribute(FilePickerHelper.FILE_PICKER_CANCEL);

    return "processed";
  }
}
```

And our action code for the binding. The FilePicker includes parameters that indicate whether there are attachments and if the user clicked "Canceled" rather than "Continue". If FILE_PICKER_CANCEL is not null, you probably want to throw away the attachments, since the user didn't want them.

The FilePicker has a number of other Parameters you can use for customization. They are detailed here: https://source.sakaiproject.org/svn/content/trunk/content-api/api/src/java/org/sakaiproject/content/api/FilePickerHelper.java

If you need to change any of these from the defaults, you could inject the SessionManager as in the Permissions Helper example to set the parameters on the ToolSession.

We still need to register our View Producer and our Action Bean:

RequestContext.xml

```
<bean class="org.sakaiproject.tool.syllabus.producers.AttachmentsHelperProducer" />
<bean id="fileAttachments" class="org.sakaiproject.tool.syllabus.ProcessFileAttachments">
  <property name="sessionManager" ref="org.sakaiproject.tool.api.SessionManager" />
  <property name="syllabusManager" ref="org.sakaiproject.api.app.syllabus.SyllabusManager" />
  <property name="attachments" ref="SyllabusAttachments" />
  <property name="syllabusPermissions" ref="syllabusPermissions" />
</bean>
```

And of course, don't forget to add your Action Bean to the list of beans that are allowed to be accessed by the request!

ApplicationContext.xml

```
<bean name="requestAddressibleBeans" class="uk.org.ponder.springutil.StringListFactory">
  <property name="strings">
    <value>SyllabusList,upDownMover,redirectUpdater,SyllabusDataEdit,syllabusDataActions,attachmentRemover<
/value>
  </property>
</bean>
```

That's it. Using Sakai Helpers from RSF leverages the same workflow as adding a page to your app, in an even more lightweight fashion.

Template, Producer, Action, Repeat!

Helper Theory

Writing Helpers in RSF