ecmarchitect.com

Alfresco Developer: Developing Custom Actions

January, 2007

Jeff Potts

Alfresco Developer: Developing Custom Actions

January, 2007

Jeff Potts

Introduction

Alfresco is a flexible platform for developing content management applications. Clients have several options to consider when selecting a user interface approach. Alfresco comes with a web client that can be used as-is or customized. Alternatively, developers can create custom applications using either the web services API or the foundation API.

Many times, the out-of-the-box web client is sufficient, even if it has to be customized slightly to fit your requirements. This is particularly true when your requirements closely resemble the all-purpose document management use case.

Deciding whether to go with the out-of-the-box web client, a customized web client, or building your own user interface from scratch requires careful thought and analysis. We'll cover that some other time.

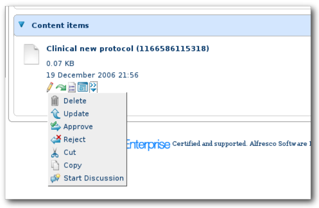
This article is the first in a series that covers Alfresco web client user interface customizations. The focus in this article is on developing custom actions.

What is an Action?

As you might suspect, an action is something a user can do to a piece of content. They are discrete units of work and can optionally be configured at run-time by the user. Some of the out-of-the-box actions include Check-out, Check-in, Update, Cut, Copy, Edit, and Delete.

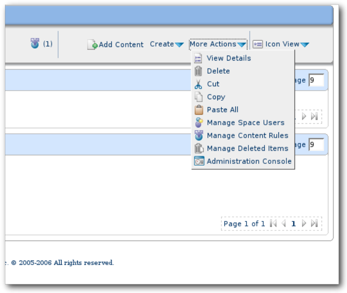
In the Alfresco web client interface, actions are everywhere. Here are some examples...

Within a space there are document specific actions...


Content items
Clinical new protocol (1166586115318)
0.07 KB
19 December 2006 21:56
Delete
Update
Approve
Reject
Cut
Copy
Start Discussion
Enterprise Certified and supported. Alfresco Software

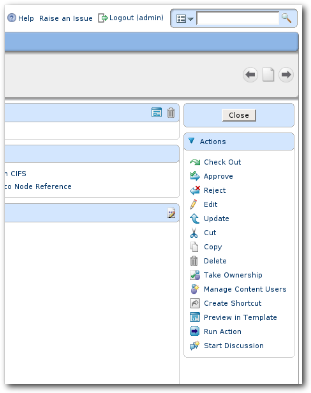
Click image to enlarge

...as well as actions listed under "More Actions".



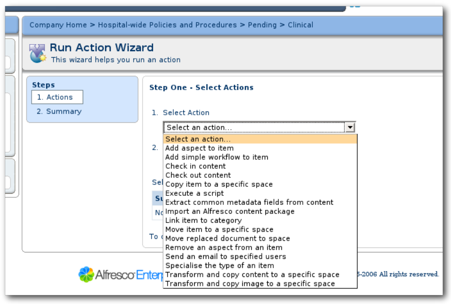
Click image to enlarge

When viewing content properties actions are listed on the right-hand side of the page.



Click image to enlarge

Actions can also be invoked as part of Alfresco's simple workflow mechanism.


Company Home > Hospital-wide Policies and Procedures > Pending > Clinical
Run Action Wizard
This wizard helps you run an action
Steps
1. Actions
2. Summary
Step One - Select Actions
1. Select Action
Select an action...
Add aspect to item
Add simple workflow to item
Check in content
Check out content
Copy item to a specific space
Execute a script
Extract common metadata fields from content
Import an Alfresco content package
Link item to category
Move item to a specific space
Move replaced document to space
Remove an aspect from an item
Send an email to specified users
Specialise the type of an item
Transform and copy content to a specific space
Transform and copy image to a specific space

Click image to enlarge

An Example

Using actions and workflow together can be an effective way to customize the web client for the needs of the business. For example, recently a client implemented Alfresco to manage the policies and procedures for their entire organization. They needed to be able to track when a policy or procedure was published that superseded another document.

There are three areas to look at when considering this requirement: the content model, the business logic, and the user interface.

A "replaces" relationship is a pretty common requirement. So much so that Alfresco has already added a "replaceable" aspect to it standard content model. So without writing any code, the business-specific content model can be written to accommodate documents that replace other documents.

Next, let's look at the business logic. In the case of this particular client, policies and procedures undergo a simple workflow. They are submitted for review and then either approved or rejected. Rejected documents go back to the submitter. Approved documents are copied to a "Published" folder. If an approved document replaces a published document, the published document is moved to an "Archived" folder.

So our business logic will be that when a document is published, if the document has a "replaces" relationship, the documents being replaced need to move to an archived folder.

The user interface can be broken down into a few different scenarios. First, document managers need to be able to establish the "replaces" relationship--for a given document, they need to be able to point to another document in a repository and declare that this document replaces that one. Next, content consumers need to be able to see a list of documents that a particular document replaces. Third, the destination folder for replaced documents should be configurable, and that will need a user interface.

Alfresco already has a component renderer in place for the "replaces" relationship. That means that the first and second UI scenarios are taken care of for us. We could create our own renderer, but for this example, we'll use the out-of-the-box code. Looking ahead to implementation, we'll be using an action to implement our business logic. One of the nice side-effects of that decision is that actions have a configuration framework we can leverage. So for this example, we won't have to write any code to address the third UI scenario. That leaves us needing to implement only the business logic.

Implementing the Action

Alfresco doesn't have out-of-the-box code to handle the type of logic we outlined in the previous section. One way to implement it is to write a custom action and then call the action from a workflow.

The action will check the document it is running against for "replaces" associations and move any documents it replaces to a folder specified when the action is configured. In this example, the folder will be set to the "Archived" folder.

Implementing the logic in an action makes a lot of sense because actions are configurable components that can be re-used across the enterprise. If another department needs to move replaced documents they can use the custom action without writing code.

How to do it

At its most basic, an action consists of an Action Executer class and its associated bean declaration. In our example, we need a UI to set the destination folder so we'll also need a bean handler, a JSP page, and a resource bundle.

Before starting, though, let's think about similar code that might already exist within Alfresco that we could leverage for our new action. Alfresco is open source--it would be a shame to ignore that valuable resource. Plus, following the same patterns to implement our customization will make it easier for someone to support and makes it easier to share our code with others or even to contribute the code back to the Alfresco community project.

We're doing a move, so the Move action is a good place to start. In fact, the only difference between the Move action and ours is that the node being moved isn't the current node--it's the node on the target end of a "replaces" association.

Action executer

Alfresco's executer class for the Move action is called org.alfresco.repo.action.executer.MoveActionExecuter. You can find it in the Alfresco Repository project. Let's copy it into our own project and call it MoveReplacedActionExecuter. (We'll call our action "Move Replaced" because it moves replaced documents). The executeImpl method is what we're looking for. That's where the move logic is handled. It looks like this:

```
 public void executeImpl(Action ruleAction, NodeRef actionedUponNodeRef) {
      if (this.nodeService.exists(actionedUponNodeRef) == true) {
          NodeRef destinationParent =
(NodeRef)ruleAction.getParameterValue(PARAM_DESTINATION_FOLDER);
          QName destinationAssocTypeQName =
(Qname)ruleAction.getParameterValue(PARAM_ASSOC_TYPE_QNAME);
          QName destinationAssocQName =
(Qname)ruleAction.getParameterValue(PARAM_ASSOC_QNAME);
          this.nodeService.moveNode( actionedUponNodeRef, destinationParent,
destinationAssocTypeQName, destinationAssocQName);
      }
 }
```

The code simply grabs some parameter values and then calls the NodeService to do the move. All we need to do is modify it to find the nodes related to the current node by a "replaces" association, and then for each result, set up and perform a move.

```
 public void executeImpl(Action ruleAction, NodeRef actionedUponNodeRef) {
      // get the replaces associations for this node
      List assocRefs = nodeService.getTargetAssocs(actionedUponNodeRef,
((QNamePattern) QName.createQName(NamespaceService.CONTENT_MODEL_1_0_URI,
"replaces")) );
      // if there are none, return
      if (assocRefs.isEmpty()) {
          // no work to do, return
```

```
        return;
    } else {
        for (AssociationRef assocNode : assocRefs) {
            // create a noderef for the replaces association NodeRef assocRef =
assocNode.getTargetRef();
            // if the node exists
            if (this.nodeService.exists(assocRef) == true) {
                NodeRef destinationParent =
(NodeRef)ruleAction.getParameterValue(PARAM_DESTINATION_FOLDER);
                QName destinationAssocTypeQName =
(Qname)ruleAction.getParameterValue(PARAM_ASSOC_TYPE_QNAME);
                String currentNameString = (String)
this.nodeService.getProperty(assocRef,
QName.createQName(NamespaceService.CONTENT_MODEL_1_0_URI, "name"));
                this.nodeService.moveNode( assocRef, destinationParent,
destinationAssocTypeQName,QName.createQName(NamespaceService.CONTENT_MODEL_1_0_URI
, currentNameString));
            }
        } // next assocNode
    } // end if isEmpty
}
```

Note that in Alfresco's "moveNode" call, they use destinationAssocQName as the qname of the newly-moved node. That variable resolves to the string "move". If you ever move more than one document into a folder using that action, you'll get integrity violations. This seems like a bug to me. My version of the "moveNode" call uses the current name of the object being moved as the new name, which seems like a better way to go.

The only other change we need to make is to change the value of the constant NAME from "move" to "move-replaced". Following the same pattern Alfresco used, the executer NAME constant needs to match the JSP name which is referenced by the bean handler class.

Bean handler

Alfresco's bean handler class for the Move action is called org.alfresco.web.bean.actions.handlers.MoveHandler and it resides in the Alfresco Web Client project. The bean handler class handles the "view" for the action configuration. Because both the "Move" action and our "Move Replaced" action both have a destination folder and no other configurable properties, only minor modifications are needed.

First, we'll be using our own JSP for the presentation so we need to change the getJSPPath() method from this:

```
public String getJSPPath() {
       return getJSPPath(MoveActionExecuter.NAME);
}
```

to something like this:

```
 public final static String CUSTOM_ACTION_JSP = "/jsp/extension/actions/" +
MoveReplacedActionExecuter.NAME + ".jsp";

 public String getJSPPath() {
     return CUSTOM_ACTION_JSP;
 }
```

All we're doing here is telling the bean where to find our customized JSP.

Next, if you've ever configured an action before you know that the user interface includes a short summary of what the action is going to do. The generateSummary() method is responsible for that message. The actual text will be in a resource bundle, so we just need to change the property name from "action_move" to "action_move_replaced".

JSP page

The JSP page that handles the move configuration for our class is identical to the out-of-the-box move. But we do want to have a different title, so we'll need our own page. And, because the page is in a different folder structure than the Alfresco JSP, the existing JSP includes will need to be updated, because they use relative links.

Copy "/source/web/jsp/actions/move.jsp" from the Alfresco Web Client project into our custom project. I use "web/jsp/extension" for all customized Alfresco JSPs by convention, so the full path to the JSP in the custom project is, "web/jsp/extension/actions/move-replaced.jsp". The customized JSPs will be deployed to the same "web/jsp" directory as the out-of-the-box JSPs so any existing relative links that need to point to out-of-the-box JSPs need an extra "../" prepended to them. In this case there are three includes that need to be updated.

To point to our custom title property in our resource bundle, change the titleId attribute of the r:page tag from "title_action_move" to "title_action_move_replaced".

That's it for the Java and JSP code. Now it is time to tie it all together with the appropriate XML configuration and properties files.

Bean declaration

First, we need to declare the action executer class and point to the resource bundle. Create a file in alfresco/extension called move-replaced-action-context.xml with the following:

```
 <beans>
     <!-- Move Replaced Action Bean -->
     <bean id="move-replaced"
```

```
class="com.optaros.alfresco.repo.action.executer.MoveReplacedActionExecuter"
parent="action-executer">
         <property name="nodeService">
             <ref bean="NodeService" />
         </property>
     </bean>
     <bean id="extension.actionResourceBundles" parent="actionResourceBundles">
         <property name="resourceBundles">
             <list>
                 <value>alfresco.extension.move-replaced-action-messages</value>
             </list>
         </property>
     </bean>
 </beans>
```

Resource bundle

Next, we need to create a resource bundle. Create a file in alfresco/extension called move-replaced-action-messages.properties. Note that this file name and path matches the resource bundle declaration from the previous step.

```
 ##
 ## Move Replaced Action I18N file
 ##
 # Action title and description
 # See action-config.properties
 move-replaced.title=Move replaced document to space
 move-replaced.description=This will move the target node of a replaces
association to a specified space.
```

Webclient.properties and web-client-config-custom.xml

The summary text and the JSP title get pulled from alfresco/extension/webclient.properties. Add the following entries to the file:

```
 action_move_replaced=Move replaced to ''{0}''

 title_action_move_replaced=Move Replaced Action
```

Finally, we need to hook in to the Alfresco action wizard UI so that when someone configures a workflow, our new action is listed as a valid choice. Edit the alfresco/extension/web-client-config-custom.xml file and add:

```
        <config evaluator="string-compare" condition="Action Wizards">
                <!--  add custom action handler for "Move Replaced" action -->
                <action-handlers>
                        <handler name="move-replaced"
class="com.optaros.alfresco.web.bean.actions.handlers.MoveReplacedHandler" />
```

```
                </action-handlers>
        </config>
```

Deploy, test, and enjoy

The files we've written in this example can be deployed over the top of an existing Alfresco deployment with the exception of the webclient.properties and web-client-config-custom.xml files, which may already exist. It can be a bit confusing to determine what goes where. In this case, all XML and properties files go in the Alfresco extension directory. All Java code goes in a JAR file that should be copied to Alfresco's WEB-INF/lib directory. If you have problems, refer to the Alfresco wiki. Links appear at the end of this article.

To set up a test, you'll need to create content using a type that has the replaceable aspect. Probably the easiest thing to do is to simply run an action on an existing document to add the replaceable aspect. In 1.3 Enterprise, "replaceable" is not listed as an aspect that can be added through an action. If your release has the same problem, add the following to the "Action Wizards" section of the web-client-config.xml file:

```
 <!-- add missing aspects to list of aspects in add/remove aspect action -->
<aspects>
     <aspect name="replaceable" />
 </aspects>
```

After the aspect has been applied, you should see a "replaces" section in the document's properties. Edit the properties and select a document to replace. This establishes the "replaces" association.

Now add a rule to the space in which "Published" documents reside. The rule should run our new "Move Replaced" action. Configure the action to move documents to the "Archived" space. To test the logic, paste the document with the "replaces" association into the "Published" space. The document it is replacing should get automatically moved to the "Archived" folder.

Conclusion

This article has shown how actions and workflows can be used within Alfresco to implement flexible and reusable business logic by making light customizations to the out-of-the-box web client. Hopefully it has sparked some ideas about how you could use custom actions in your next Alfresco implementation.

Where to find more information

- The Alfresco SDK comes with a Custom Action example.
- For deployment help, see the Client Configuration Guide and Packaging and Deploying Extensions in the Alfresco wiki.

- For general development help, see the Developer Guide.
- For help customizing the data dictionary, see the Data Dictionary wiki page.

About the Author

Jeff Potts is a Principle Architect at Optaros, a leading Open Source and Next Generation Internet consultancy. Jeff has nearly fifteen years of experience implementing content management, collaboration, and other knowledge management technologies for a variety of Fortune 500 companies. Jeff lives in Dallas, Texas with his wife and two kids. Read more at ecmarchitect.com.