Different Publishers Per Path Workflow

- Summary
- Prerequisites
- Getting your module ready
    - Create a new module
    - Module dependencies
- Changing Publication task
- Acknowledges

Summary

- This tutorial shows how to make the workflow send the publication request to different groups of users depending on the content path, using the workflow that comes by default on Magnolia 5.3+.

Prerequisites

- Magnolia EE 5.3+ (workflow is an enterprise feature)
- Eclipse (This tutorial uses Eclipse Indigo). You are free to work with an IDE of your choice. However, for this particular implementation we recommend working with Eclipse to ensure that that implementation is successful. See Working with Eclipse and Git .
- Maven: http://maven.apache.org/
- Git:http://git.magnolia-cms.com/gitweb/

Getting your module ready

Create a new module

Set up your module with Maven and use archetype: `magnolia-module-archetype (An archetype to create basic Magnolia modules)`

Module creation input example

```
$ mvn archetype:generate -DarchetypeCatalog=https://nexus.magnolia-cms.com/content/groups/public
[...]
Confirm properties configuration:
groupId: org.mydomain.workflow
artifactId: mycompany-module-myproject-workflow
version: 1.0-SNAPSHOT
package: org.mydomain.workflow
magnolia-version: 5.3.7
module-class-name: MyProjectWorkflowModule
module-name: myproject-workflow
Y:
```

Import the module into your IDE

? Unknown Attachment

Module dependencies

Ensure that the pom file contains all of the necessary dependencies. As workflow is part of the Enterprise Edition, un-comment the repository at the end of the file.

mycompany-module-myproject-workflow pom

```
[...]
<dependency>
    <groupId>info.magnolia.workflow</groupId>
    <artifactId>magnolia-module-workflow-jbpm</artifactId>
</dependency>
[...]
```

In this case I'm letting the parent pom manage the versions. For Magnolia 5.3.8 we are using `magnolia-module-workflow-jbpm-5.4.5`.

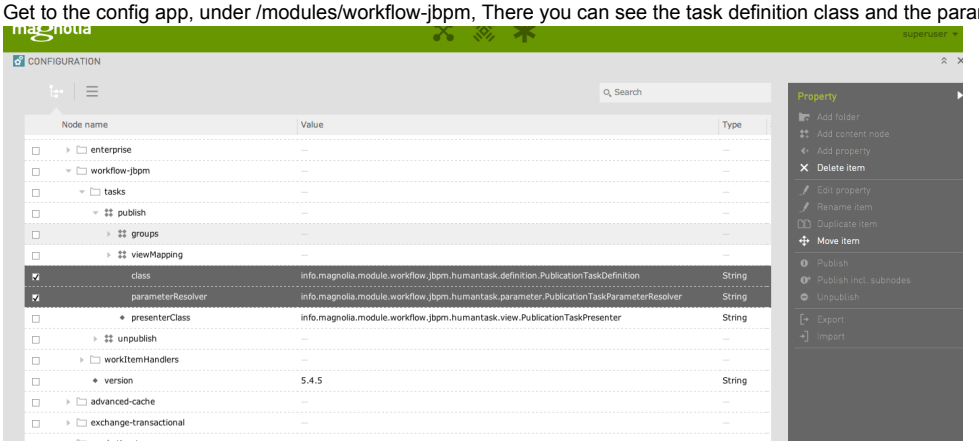
Update your module descriptor located under `src/main/resources/META-INF/magnolia/myproject-workflow.xml`.

myproject-workflow.xml

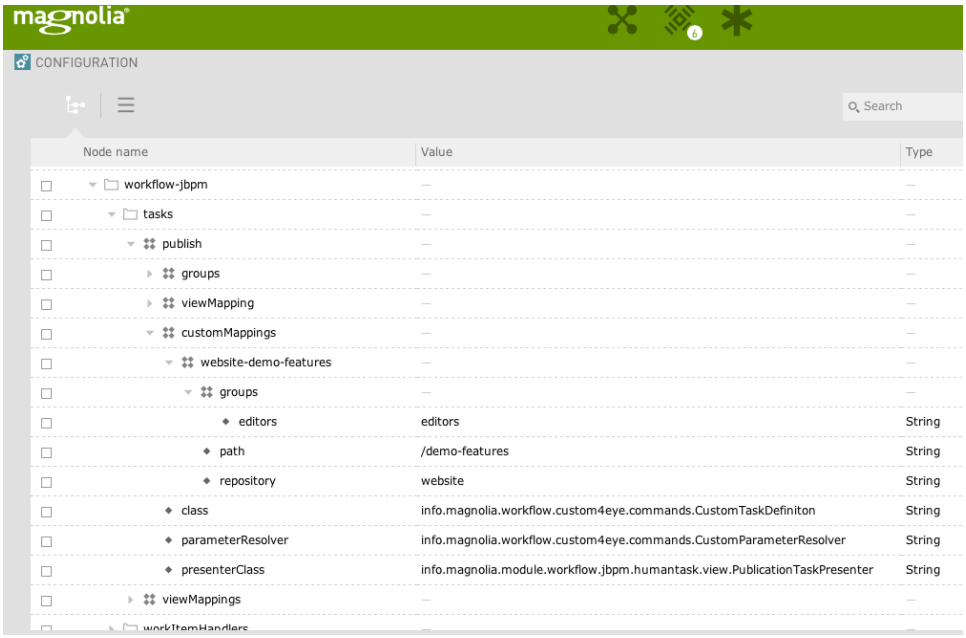
```
[...]
<dependency>
    <name>workflow-jbpm</name>
    <version>5.4/*</version>
</dependency>
[...]
```

Changing Publication task

Our default workflow uses a publication task to send the name of the publishers groups to the running workflow, we will need to extend this task to add the new groups.

1. Locate the publishing task.
   Get to the config app, under /modules/workflow-jbpm, There you can see the task definition class and the parameter resolver class.
   

2. Before you create your own TaskDefinition class and TaskParameterResolver class, we need to decide how will we set the new parameters in the configuration, in this tutorial we will create a mapping with path, groupids and repository, this way you can create as many mappings as you need. Once we have this clear, we need to read the new parameters from the configuration into our class.



3. Create custom TaskDefinition extending from the default one and provide its implementation.

```
public interface CustomTaskDefiniton extends PublicationTaskDefinition {

        Collection<CustomMapping> getCustomMappings();

}
```

```
public class CustomTaskDefinitionImpl extends ConfiguredPublicationTaskDefinition implements
CustomTaskDefiniton {

        private Collection<CustomMapping> customMappings = new LinkedList<CustomMapping>();

        public Collection<CustomMapping> getCustomMappings() {

                return customMappings;
        }

        public void setCustomMappings(Collection<CustomMapping> mappings) {

                this.customMappings = mappings;
        }

        /**
     * Used to define the mappings.
     */
    public static class CustomMapping {

            private String repository;
                private String path;
        private List<String> groups;
        private boolean enabled = true;

        public String getRepository() {
                        return repository;
                }

                public void setRepository(String repository) {
                        this.repository = repository;
                }

        public String getPath() {
            return this.path;
        }

        public void setPath(String path) {
            this.path = path;
        }

        public List<String> getGroups() {
            return groups;
        }

        public void setGroups(List<String> groups) {
            this.groups = groups;

        }

        public boolean isEnabled() {
            return this.enabled;
        }

        public void setEnabled(boolean enabled) {
            this.enabled = enabled;
        }
    }
}
```

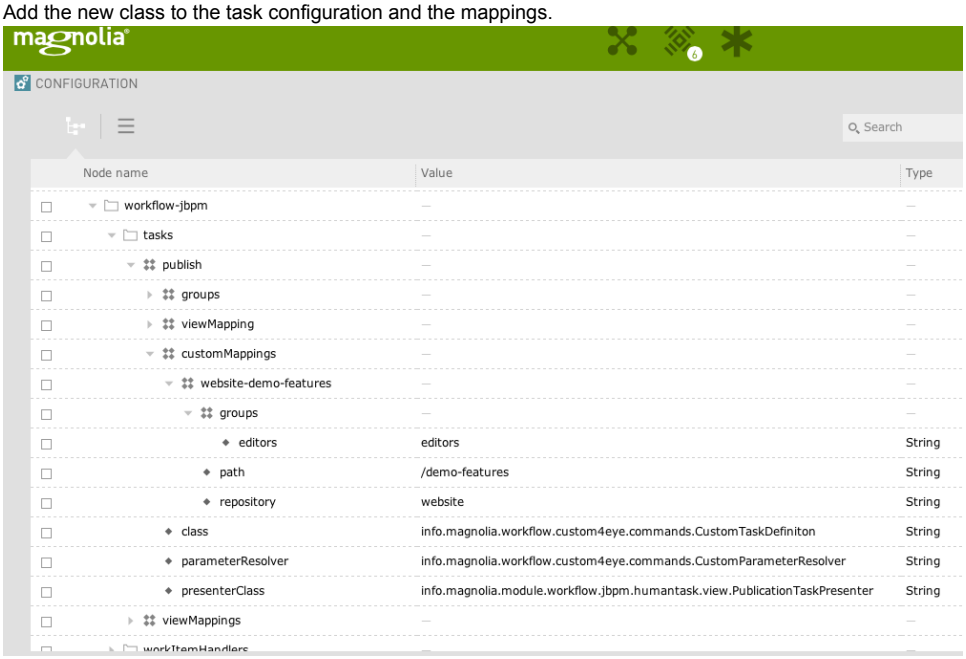
4. Create custom ParameterResolver extending from the default one. The method to override is the setTaskParameters.

```
public class CustomParameterResolver extends PublicationTaskParameterResolver {

        @Inject
        public CustomParameterResolver(CustomTaskDefiniton definition) {
                super(definition);
        }

        @Override
        public void setTaskParameters(HumanTask task, WorkItem workItem) {

                super.setTaskParameters(task, workItem);
                Map<String, Object> content = task.getContent();

                if (content != null) {
                        String path = (String) content.get(Context.ATTRIBUTE_PATH);
                        String repository =        (String) content.get(Context.ATTRIBUTE_REPOSITORY);
                        task.setGroupIds(getGroups(repository, path));
                }
        }

        protected List<String> getGroups(String repository, String path) {

        for (Iterator<CustomMapping> iter = ((CustomTaskDefiniton) getDefinition()).getCustomMappings().
iterator(); iter.hasNext();) {
            CustomMapping mapping = iter.next();

            if (mapping.isEnabled() && path != null && path.startsWith(mapping.getPath())
                            && repository != null && repository.startsWith(mapping.getRepository())) {

                return mapping.getGroups();
            }
        }
                return getDefinition().getGroups();
    }
}
```

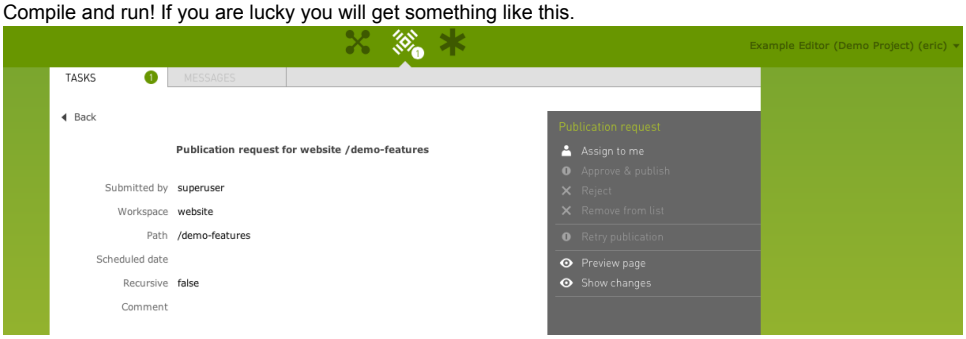
5. Now add the new definition to the module properties file.

```
[...]
<components>
    <id>main</id>
    <type-mapping>
      <type>info.magnolia.workflow.custom4eye.commands.CustomTaskDefiniton</type>
      <implementation>info.magnolia.workflow.custom4eye.commands.CustomTaskDefinitionImpl<
/implementation>
    </type-mapping>
</components>
[...]
```

6. Add the new class to the task configuration and the mappings.



7. Compile and run! If you are lucky you will get something like this.



Acknowledges

Thanks to Rich and Espen for ideas and formattings 😂.