DSpace 1.8 Configurable Reviewer Workflow

Introduction

Configurable Reviewer Workflows are a contribution forthcoming from @mire and will provide the following features for DSpace Reviewer Workflows

The current DSpace Reviewer Workflow System hardcodes a fixed set of 3 steps, of which the order is determined beforehand, and only the authenticated users (epersons) assigned to each of these workflow steps may participate in the workflow tasks that get assigned. In various use cases, the 3 simple steps for approving, rejecting, and editing the metadata of an submission, executed by the first person who claims the task from a predefined group (pool) of users, have proved to be quite insufficient when institutions have significantly more complex needs for mediating the review of DSpace submissions. To resolve several of the shortcomings, @mire has implemented a ”Configurable DSpace Workflow System”, planned for contribution to DSpace 1.8. This system is a generalization of a solution implemented and deployed in the production environment for certain clients.

The primary focus of the workflow framework is to create a more flexible solution for the administrator to configure, and even to allow an application developer to implement custom steps, which may be configured in the workflow for the collection through a simple configuration file. The concept behind this approach was modeled on the configurable submission system already present in DSpace.

Architectural Design - API

Workflow Manager

Currently, the whole workflow process is managed by the XmlWorkflowManager as was the case for the previous version of the workflow. The workflow manager is responsible for executing the different actions, processing their outcome and activating new ones. The WorkflowManager is being called from the user interface in order to process an action submitted by a user. In case another action follows the one executed by the user, the WorkflowManager will return this action and the user interface will show the new action page.
For the DSpace XMLUI, the submission.js flowscript file is responsible for interacting with the WorkflowManager and will show the user the correct page after submitting an action.

Workflow

The WorkflowManager uses one Workflow object for each collection. In many cases, multiple collections will use the same workflow process and for those, the same Workflow object will be used by the collection manager. A Workflow object represents one workflow process as configured in the workflow.xml and contains a number of workflow steps that the workflow item should go through. The main responsibility of the Workflow object is managing its steps and managing the different roles used in the workflow process. It contains two mappings from identifiers used in the workflow.xml configuration to the actual Java objects used by the workflow framework.

Step

A Step is a child element of the Workflow object and contains a number of actions that will sequentially be executed by one of the workflow roles once the workflow item goes through the step. Each step has the following important properties:

- userSelectionMethod: The user selection method refers to a UserSelectionActionConfig class and is the first action that will be performed in a step. The userSelectionMethod defines how the members of the step role will be assigned to the actions that need to be executed. An example is the task pool system used by the original DSpace workflow.
- actionConfigsMap: The actionconfigsmap contains a mapping from the processing action identifiers used in the workflow.xml to the WorkflowActionConfig objects used by the workflow system. These WorkflowActionConfig objects are the actions that are responsible for the actual execution of the step.
- outcomes: The step also contains a list of alternative outcomes. In case one of the actions in the step returns an outcome different from 0, this list will be used to lookup the next step to activate.
- role: The role object contains the workflow role object that is used to define the epersons or groups that are responsible for the execution of this step.

Workflow Action Config

WorkflowActionConfig objects are the children of a Step object. The actions are the different tasks that can be executed as part of one Step and will therefore be executed by the same role. There are two types of WorkflowActionConfig objects that are being used in the workflow framework:

- WorkflowActionConfig: responsible for the execution of the different tasks in that step.
- UserSelectionActionConfig: responsible for assigning the members of a Step role to the first WorkflowActionConfig of the step. This class subclasses the WorkflowActionConfig class because it adds a few additional functions for assigning users.

The workflow framework uses these WorkflowActionConfig object to separate the user interface classes from the API classes. Therefore, each WorkflowActionConfig object contains an Action object and a property that indicates whether the Action requires interaction with the user interface. The Action object is the API part of the action and is responsible for the actual execution of the action. In case user interface interaction is required, the action will not be automatically executed by will halt until the user interface interaction has been done.

Action

An Action object is responsible for the actual execution of the task for which it is responsible. The Action object is a part of a WorkflowActionConfig object. In order to execute the task for which the Action object is responsible, two methods are available:

- activate: The activate method will perform a number of steps required for the task to be ready for execution. For example, an Action that is responsible for the creation of a task pool, will create the task pool in the activate method.
- execute: The execute method will perform the actual task of the action. For the same example of a task pool feature, the execute action will claim a task for one of the users.

Workflow Factory

The WorkflowFactory is responsible for managing the different workflow objects that are being used by the different collections in DSpace. The WorkflowFactory contains a mapping from collection handles or the "default" options to Workflow objects. Each workflow object represents one workflow process containing a number of different steps. The WorkflowFactory contains all the required functions for loading the API bean classes that are required by the workflow framework. When a workflow, a step or an action is not yet available in the workflow cache, the WorkflowFactory will create the required objects based on the Spring configuration file (config/workflow-actions.xml).

Workflow flow control API

This section shortly explains how an item moves through the workflow starting from the end of the submission process and ending at the archival in DSpace.
When an item enters the workflow after the submission process, the first workflow-step in the collection workflow is selected. The first action of that step, the user selection action, is activated. From there, the workflow behaves as follows:

Case 1: the action requires UI interaction

In case an action requires interaction with the user interface, the workflow process halts after the activation of the action. The activation method will be responsible for preparing the workflow-item for user interface interaction. An example for a pool task activation task is generating a task pool. The user interface interaction will offer the users for which the pool has been generated the ability to claim the action. After interaction with the user interface, the execute method is executed. Based on the result type of this method, the workflow will proceed as explained in the transition section described below.

Case 2: the action does not require UI interaction

In case an action does not require interaction with the user interface, the execute method is executed directly after the activate method. This makes it possible to create automatic actions. An example of such an automatic action might be to calculate an average of reviewer scores and decide based on that average to approve or reject the item. Based on the result type of the execute method, the workflow will proceed as explained in the transition section described below.

Transition between actions and steps

Transitions between different actions and steps depend on the result type of the execute method. The following result types are possible:

- TYPE_PAGE: The TYPE_PAGE result type can occur after interaction with the user interface. This means that the action has multiple pages for the user to go through in order to complete the action. The TYPE_PAGE will keep the user in the same action and this result will be used by the user interface flow to display the user the next page of the action. The user interface side of the workflow will be explained later in this document.
- TYPE_ERROR: This means that a expectable error has occured during the execution of an action. This type can only occur while interacting with the user interface. The user interface side of the workflow will handle the error and notify the user what to do in order to solve the problem. An example of such an error results is a missing field during metadata input.
- TYPE_CANCEL: This type is returned only if the action requires user interface interaction. It means that the user has clicked the cancel button. The user interface workflow control will return the user to the submission overview.
- TYPE_SUBMISSION_PAGE: This restult type also only happens when interacting with the user interface. It means that the user has finished a task in the workflow and will be sent back to the submission overview.
- TYPE_OUTCOME: The TYPE_OUTCOME result type can occur with or withour previous user interface interaction. When a result has the result type TYPE_OUTCOME, more specific information about the actual outcome is available as well. The different outcomes are the following:
    - OUTCOME_COMPLETE: Outcome complete or "0" indicates that the action has ended as it should have ended. The workflow process will in this case:
        - Activate the next action if one is available;
        - Start the next step in case this was the last action of a step and in case no other users are required to complete the step;
        - Do nothing in case this was the last action of a step but other users need to go through the step before it is completed.
    - Another outcome: Other outcomes can be any number larger than "0". These other outcomes correspond to alternative outcomes as defined in the workflow configuration. In case an action returns an alternative outcome, the workflow process will activate the step corresponding to this alternative outcome.

Architectural Design - UI

Workflow XMLUI Factory

The WorkflowXMLUIFactory is responsible for creating the required UI actions that are used by the workflow framework. Currently, the WorkflowXMLUIFactory only has one method that will create an instance of the required class by using the Spring configuration file (config/workflow-actions-xmlui.xml). The caching of the different UI actions is currently being done by the Spring framework itself.

Action Interface

Each object returned by the WorkflowXMLUIFactory is an implementation of the ActionInterface object or more specific, an implementation of AbstractXMLUIAction. These objects are used by the WorkflowTransformer in order to generate the different user interface pages that are used by the workflow framework.

Workflow flow control - UI

The flow control of the workflow framework at the side of the user interface is done in flowscript, more specific in the submission.js file. In case the workflow process requires user interface interaction, tasks will be available for the user in the submission overview page in the DSpace user interface. Once the user starts one of these tasks, the workflow flow control in the flowscript file is used to perform the required tasks. These can be one of the following:

- The user wants to see the full/simple item display page of the workflow item: Nothing happens, the page is reloaded but with another parameter. The transformer responsible for generating the page will add the full/simple item display content instead of the previous one.
- The user wants to edit the item (reuse the submission steps): The existing DSpace submission control is used to handle this request.
- The user clicks one of the other available task buttons: In this case, the API side of the workflow process is called in order to handle the request. The next page for the user will be one of the following, based on the outcome:
    - The same action is returned by the API: This means that an expectable has occurred or another page for that action will be displayed. The responsible AbstractXMLUIAction will generate the required content for the page.
    - Another action is returned by the API: This means that another action in the step is available for the user. The responsible AbstractXMLUIAction will generate the required content for the page.
    - No action is returned: This means the user is done for now and the submission overview page will be displayed.
    - An exception is thrown: An unexpected error has occurred.

Configuration

DSpace.cfg configuration

Currently, there are no workflow configuration options added to the DSpace.cfg configuration file.

Main workflow configuration

The workflow main configuration can be found in the workflow.xml file, located in {dspace.dir}/config. An example of this workflow configuration file can be found bellow.

```
<?xml version="1.0" encoding="UTF-8"?>
<wf-config>
    <workflow-map>
        <!-- collection to workflow mapping -->
        <name-map collection="default" workflow="{workflow.id}"/>
        <name-map collection="123456789/0" workflow="{workflow.id2}"/>
    </workflow-map>

    <workflow start="{start.step.id}" id="{workflow.id}">
        <roles>
         <!-- Roles used in the workflow -->
        </roles>

            <!-- Steps come here-->
            <step id="ExampleStep1" nextStep="ExampleStep2" userSelectionMethod="{UserSelectionActionId}">
            <!-- Step1 config-->
            </step>
        <step id="ExampleStep2" userSelectionMethod="{UserSelectionActionId}">

        </step>
    </workflow>
    <workflow start="{start.step.id2}" id="{workflow.id}">
     <!-- Another workflow configuration-->
    </workflow>
</wf-config>
```

workflow-map

The workflow map contains a mapping between collections in DSpace and a workflow configuration. Similar to the configuration of the submission process, the mapping can be done based on the handle of the collection. The mapping with "default" as the value for the collection mapping, will be used for the collections not occurring in other mapping tags. Each mapping is defined by a "name-map" tag with two attributes:

- collection: can either be a collection handle or "default"
- workflow: the value of this attribute points to one of the workflow configurations defined by the "workflow" tags

workflow

The workflow element is a repeatable XML element and the configuration between two "workflow" tags represents one workflow process. It requires the following 2 attributes:

- id: a unique identifier used for the identification of the workflow and used in the workflow to collection mapping
- start: the identifier of the first step of the workflow, this will be the entry point of this workflow-process. When a new item has been committed to a collection that uses this workflow, the step configured in the "start" attribute will he the first step the item will go through.

roles

Each workflow process has a number of roles defined between the "roles" tags. A role represents one or more DSpace EPersons or Groups and can be used to assign them to one or more steps in the workflow process. One role is represented by one "role" tag and has the following attributes:

- id: a unique identifier (in one workflow process) for the role
- description: optional attribute to describe the role
- scope: optional attrbiute that is used to find our group and must have one of the following values:
    - collection: The collection value specifies that the group will be configured at the level of the collection. This type of groups is the same as the type that existed in the original workflow system.In case no value is specified for the scope attribute, the workflow framework assumes the role is a collection role.
    - repository: The repository scope uses groups that are defined at repository level in DSpace. The name attribute should exactly match the name of a group in DSpace.
    - item: The item scope assumes that a different action in the workflow will assign a number of EPersons or Groups to a specific workflow-item in order to perform a step. These assignees can be different for each workflow item.
- name: The name specified in the name attribute of a role will be used to lookup the in DSpace. The lookup will depend on the scope specified in the "scope" attribute:
    - collection: The workflow framework will look for a group containing the name specified in the name attribute and the ID of the collection for which this role is used.
    - repository: The workflow framework will look for a group with the same name as the name specified in the name attribute
    - item: in case the item scope is selected, the name of the role attribute is not required
- internal: optional attribute which isn't really used at the moment, false by default

```
<roles>
    <role id="{unique.role.id} description="{role.description}" scope="{role.scope}" name="{role.name}"
internal="true/false"/>
</roles>
```

step

The step element represents one step in the workflow process. A step represents a number of actions that must be executed by one specified role. In case no role attribute is specified, the workflow framework assumes that the DSpace system is responsible for the execution of the step and that no user interface will be available for each of the actions in this step. The step element has the following attributes in order to further configure it:

- id: The id attribute specifies a unique identifier for the step, this id will be used when configuring other steps in order to point to this step. This identifier can also be used when configuring the start step of the workflow item.
- nextStep: This attribute specifies the step that will follow once this step has been completed under normal circumstances. If this attribute is not set, the workflow framework will assume that this step is an endpoint of the workflow process and will archive the item in DSpace once the step has been completed.
- userSelectionMethod: This attribute defines the UserSelectionAction that will be used to determine how to attache users to this step for a workflow-item. The value of this attribute must refer to the identifier of an action bean in the workflow-actions.xml. Examples of the user attachment to a step are the currently used system of a task pool or as an alternative directly assigning a user to a task.
- role: optional attribute that must point to the id attribute of a role element specified for the workflow. This role will be used to define the epersons and groups used by the userSelectionMethod.
- RequiredUsers

```
<step id="{step.id}" nextStep="{next.step.id}" userSelectionMethod="{user.selection.bean.id}" role="{role.id}" >
<!-- optional alternate outcomes, depending on the outcome of the actions you can alter the next step here -->
<alternativeOutcome>
     <step status="{integer}">{alternate.step.id}</step>
</alternativeOutcome>
<action id="{action.bean.id}"/>
<action id="{action.bean.id.1}"/>
</step>
```

Each step contains a number of actions that the workflow item will go through. In case the action has a user interface, the users responsible for the exectution of this step will have to execute these actions before the workflow item can proceed to the next action or the end of the step.

There is also an optional subsection that can be defined for a step part called "alternativeOutcome". This can be used to define outcomes for the step that differ from the one specified in the nextStep attribute. Each action returns an integer depending on the result of the action. The default value is "0" and will make the workflow item proceed to the next action or to the end of the step.
In case an action returns a different outcome than the default "0", the alternative outcomes will be used to lookup the next step. The alternativeOutcome element contains a number of steps, each having a status attribute. This status attribute defines the return value of an action. The value of the element will be used to lookup the next step the workflow item will go through in case an action returns that specified status.

Workflow actions configuration

API configuration

The workflow actions configuration is located in the {dspace.dir}/config/spring/api directory and is named "workflow-actions.xml". This configuration file describes the different Action Java classes that are used by the workflow framework. Because the workflow framework uses Spring framework for loading these action classes, this configuration file contains Spring configuration.

This file contains the beans for the actions and user selection methods referred to in the workflow.xml. In order for the workflow framework to work properly, each of the required actions must be part of this configuration.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans
/spring-beans-2.0.xsd
                        http://www.springframework.org/schema/util http://www.springframework.org/schema/util
/spring-util-2.0.xsd">

    <!-- At the top are our bean class identifiers --->
    <bean id="{action.api.id}" class="{class.path}" scope="prototype"/>
    <bean id="{action.api.id.2}" class="{class.path}" scope="prototype"/>

 <!-- Below the class identifiers come the declarations for out actions/userSelectionMethods -->

 <!-- Use class workflowActionConfig for an action -->
 <bean id="{action.id}" class="oorg.dspace.xmlworkflow.state.actions.WorkflowActionConfig" scope="prototype">
     <constructor-arg type="java.lang.String" value="{action.id}"/>

     <property name="processingAction" ref="{action.api.id}"/>
     <property name="requiresUI" value="{true/false}"/>
 </bean>

 <!-- Use class UserSelectionActionConfig for a user selection method -->
 <!--User selection actions-->
 <bean id="{action.api.id.2}" class="org.dspace.xmlworkflow.state.actions.UserSelectionActionConfig" scope="
prototype">
     <constructor-arg type="java.lang.String" value="{action.api.id.2}"/>

     <property name="processingAction" ref="{user.selection.bean.id}"/>
     <property name="requiresUI" value="{true/false}"/>
 </bean>
</beans>
```

Two types of actions are configured in this Spring configuration file:

- User selection action: This type of action is always the first action of a step and is responsible for the user selection process of that step. In case a step has no role attached, no user will be selected and the NoUserSelectionAction is used.
- Processing action: This type of action is used for the actual processing of a step. Processing actions contain the logic required to execute the required operations in each step. Multiple processing actions can be defined in one step. These user and the workflow item will go through these actions in the order they are specified in the workflow configuration unless an alternative outcome is returned by one of them.

User Selection Action

Each user selection action that is used in the workflow config refers to a bean definition in this workflow-actions.xml configuration. In order to create a new user selection action bean, the following XML code is used:

```
<bean id="{action.api.id.2}" class="org.dspace.xmlworkflow.state.actions.UserSelectionActionConfig" scope="
prototype">
     <constructor-arg type="java.lang.String" value="{action.api.id.2}"/>

     <property name="processingAction" ref="{user.selection.bean.id}"/>
     <property name="requiresUI" value="{true/false}"/>
 </bean>
```

This bean defines a new UserSelectionActionConfig and the following child tags:

- constructor-arg: This is a constructor argument containing the ID the task. This is the same as the id attribute of the bean and is used by the workflow config to refer to this action.
- property processingAction: This tag refers the the ID of the API bean, responsible for the implementation of the API side of this action. This bean should also be configured in this XML.
- property requiresUI: In case this property is true, the workflow framework will expect a user interface for the action. Otherwise the framework will automatically execute the action and proceed to the next one.

Processing Action

Processing actions are configured similar to the user selection actions. The only difference is that these processing action beans are implementations of the WorkflowActionConfig class instead of the UserSelectionActionConfig class.

User Interface configuration

The configuration file for the workflow user interface actions is located in the {dspace.dir}/config/xmlui and is named "workflow-actions-xmlui.xml". BEach bean defined here has an id which is the action identifier and the class is a classpath which links to the xmlui class responsible for generating the User Interface side of the workflow action. Each of the class defined here must extend the //org.dspace.app.xmlui.aspect.submission.workflow. AbstractXMLUIAction// class, this class contains some basic settings for an action and has a method called //addWorkflowItemInformation()// which will render the given item with a show full link so you don't have to write the same code in each of your actions if you want to display the item. The id attribute used for the beans in the configuration must correspond to the id used in the workflow configuration. In case an action requires a User Interface class, the workflow framework will look for a UI class in this configuration file.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans
/spring-beans-2.0.xsd
                        http://www.springframework.org/schema/util http://www.springframework.org/schema/util
/spring-util-2.0.xsd">

    <bean id="{action.id}" class="{classpath}" scope="prototype"/>
    <bean id="{action.id.2}" class="{classpath}" scope="prototype"/>
</beans>
```

Authorizations

Currently, the authorizations are always granted and revoked based on the tasks that are available for certain users and groups. The types of authorization policies that is granted for each of these is always the same:

- READ
- WRITE
- ADD
- DELETE

Database

Database changes - Original text

The changes made to the database can always be found in the {dspace.src.dir}/etc directory in the file //collection-workflow-changes.sql//.
The workflow uses a separate metadata schema named //workflow// the fields this schema contains can be found in the {dspace.dir}/registries directory and in the file //workflow-types.xml//.

The tables //collectionrole//, //workflowitemrole// where created and are used by the workflow system for the roles (the roles have been described above).
The //tasklistitem// table was altered to contain the step and the action identifiers so that the workflow system knows where in the workflow the item is.
The table //taskowner// was created so that multiple user can claim one item (this was not possible in the old workflow, there the owning user was a field in the workflowitem table).

Database additions

There are a number of important database changes in order to support the new reviewer workflow framework. The following tables have been added:

collectionrole

The collectionrole table stores the the groups that are assigned to one of the collection scope roles. A new entry in this table is created when an administrator creates a new collection role group in the Edit Collection interface. The table has the following columns:

- collectionrole_id (INTEGER): Unique ID for a row in the collectionrole table
- role_id: The (TEXT): Stores the name of this role as specified in the XML configuration document

- collection_id: (INTEGER, REFERENCES collection(collection_id)): The ID of the collection for which the role is defined.
- group_id: (INTEGER, REFERENCES epersongroup(eperson_group_id)): The ID of the group that contains the members of this role.

workflowitemrole

This table stores the the group or eperson that has been assigned to one of the workflow items to perform one of the steps in the workflow. This table stores the item scope workflow roles. For one item, multiple groups or epersons can be configured so multiple rows in this table can correspond to the same item, step and workflow. The selection of the members of these item roles is performed by another action earlier in the workflow. This table contains the following columns:

- workflowitemrole_id (INTEGER): Unique ID for a row in the workflowitemrole table.
- role_id (TEXT): Stores the name of this role as specified in the XML configuration document.
- workflow_item_id (INTEGER, REFERENCES workflowitem(workflow_id)): The workflow item for which the role has been configured.
- eperson_id (INTEGER, REFERENCES eperson(eperson_id)): The eperson that has been configured for this workflow item role. This value can be empty in case a group has been configured.
- group_id (INTEGER REFERENCES epersongroup(eperson_group_id)): The group that has been configured for this workflow item role. This value can be empty in case an eperson has been configured.

taskowner

This table contains all the tasks that belong to an eperson. Entries in this table are created when a user claims a certain task from the task pool or can be automatically created when automated action assigns the user to a task. The taskowner table contains the following columns:

- taskowner_id (INTEGER): Unique ID for a row in the taskowner table.
- workflow_item_id (INTEGER, REFERENCES workflowitem(workflow_id)): The ID of the workflow item for which an eperson has a task assigned /claimed.
- action_id (TEXT): The ID of the action the eperson has claimed or has been assigned to.
- step_id (TEXT): The ID of the step that contains the action.
- workflow_id (TEXT): The ID of the workflow process that contains the step.
- owner_id (INTEGER, REFERENCES eperson(eperson_id)): The ID of the eperson who owns the task.

Database changes

The following tables have been added to the DSpace database. All tables are prefixed with 'xmlwf_' to avoid any confusion with the existing workflow related database tables:

xmlwf_workflowitem

The xmlwf_workflowitem table contains the different workflowitems in the workflow. This table has the following columns:

- workflowitem_id: The identifier of the workflowitem and primary key of this table
- item_id: The identifier of the DSpace item to which this workflowitem refers.
- collection_id: The collection to which this workflowitem is submitted.
- multiple_titles: Specifies whether the submission has multiple titles (important for submission steps)
- published_before: Specifies whether the submission has been published before (important for submission steps)
- multiple_files: Specifies whether the submission has multiple files attached (important for submission steps)

xmlwf_collectionrole

The xmlwf_collectionrole table represents a workflow role for one collection. This type of role is the same as the roles that existed in the original workflow meaning that for each collection a separate group is defined to described the role. The xmlwf_collectionrole table has the following columns:

- collectionrol_id: The identifier of the collectionrole and the primaty key of this table
- role_id: The identifier/name used by the workflow configuration to refer to the collectionrole
- collection_id: The collection identifier for which this collectionrole has been defined
- group_id: The group identifier of the group that defines the collection role

xmlwf_workflowitemrole

The xmlwf_workflowitemrole table represents roles that are defined at the level of an item. These roles are temporary roles and only exist during the execution of the workflow for that specific item. Once the item is archived, the workflowitemrole is deleted. Multiple rows can exist for one workflowitem with e.g. one row containing a group and a few containing epersons. All these rows together make up the workflowitemrole The xmlwf_workflowitemrole table has the following columns:

- workflowitemrole_id: The identifier of the workflowitemrole and the primaty key of this table
- role_id: The identifier/name used by the workflow configuration to refer to the workflowitemrole
- workflowitem_id: The xmlwf_workflowitem identifier for which this workflowitemrole has been defined
- group_id: The group identifier of the group that defines the workflowitemrole role
- eperson_id: The eperson identifier of the eperson that defines the workflowitemrole role

xmlwf_pooltask

The xmlwf_pooltask table represents the different task pools that exist for a workflowitem. These task pools can be available at the beginning of a step and contain all the users that are allowed to claim a task in this step. Multiple rows can exist for one task pool containing multiple groups and epersons. The xmlwf_pooltask table has the following columns:

- pooltask_id: The identifier of the pooltask and the primaty key of this table
- workflowitem_id: The identifier of the workflowitem for which this task pool exists
- workflow_id: The identifier of the workflow configuration used for this workflowitem
- step_id: The identifier of the step for which this task pool was created
- action_id: The identifier of the action that needs to be displayed/executed when the user selects the task from the task pool
- eperson_id: The identifier of an eperson that is part of the task pool
- group_id: The identifier of a group that is part of the task pool

xmlwf_claimtask

The xmlwf_claimtask table represents a task that has been claimed by a user. Claimed tasks can be assigned to users or can be the result of a claim from the task pool. Because a step can contain multiple actions, the claimed task defines the action at which the user has arrived in a particular step. This makes it possible to stop working halfway the step and continue later. The xmlwf_claimtask table contains the following columns:

- claimtask_id: The identifier of the claimtask and the primary key of this table
- workflowitem_id: The identifier of the workflowitem for which this task exists
- workflow_id: The id of the workflow configuration that was used for this workflowitem
- step_id: The step that is currenlty processing the workflowitem
- action_id: The action that should be executed by the owner of this claimtask
- owner_id: References the eperson that is responsible for the execution of this task

xmlwf_in_progress_user

The xmlwf_in_progess_user table keeps track of the different users that are performing a certain step. This table is used because some steps might require multiple users to perform the step before the workflowitem can proceed. The xmlwf_in_progress_user table contains the following columns:

- in_progress_user_id: The identifier of the in progress user and the primary key of this table
- workflowitem_id: The identifier of the workflowitem for which the user is performing or has performed the step.
- user_id: The identifier of the eperson that is performing or has performe the task
- finished: Keeps track of the fact that the user has finished the step or is still in progress of the execution

Backwards compatibility

Aspects and configuration files

In order to be backwards compatible with previous versions of DSpace, a number of changes have been made to the existing submission/workflow aspect. The submission aspect has been split up into muliple aspects: one submission aspect for the submission process, one workflow aspect containing the code for the original workflow and one xmlworkflow aspect containing the code for the new XML configurable workflow framework. In order to enable one of the two aspects, either the workflow or xmlworkflow aspect should be enabled in the dspace-install-dir/config/xmlui.xconf configuration file. Besides that, a workflow configuration file has been created that specifies the workflow that will be used in the back-end of the DSpace code. It is important that the option selected in this configuration file matches the aspect that was enabled. The workflow configuration file is available in dspace-install-dir/config /modules/workflow.cfg. This configuration file has been added because it is important that a CLI import process uses the correct workflow and this should not depend on the UI configuration. The workflow.cfg configration file contains the following propertu:

- workflow.framework: possible values are 'originalworkflow' for the original workflow or xmlworkflow for the new XML configurable workflow framework.

Workflowitem conversion/migration scripts

Depending on the workflow that is used by a DSpace installation, different scripts can be used when migration to the new workflow.

SQL based migration

SQL based migration can be used when the out of the box original workflowframework is used by your DSpace installation. This means that your DSpace installation uses the workflow steps and roles that are available out of the box. The migration script will migrate the policies, roles, tasks and workflowitems from the original workflow to the new workflowframework. The following SQL scripts are available depending on the database that is used by the DSpace installation:

- dspace-install-dir/etc/oracle/xmlworkflow/workflow_migration.sql
- dspace-install-dir/etc/postgres/xmlworkflow/workflow_migration.sql

Java based migration

In case your DSpace installation uses a customized version of the workflow, the migration script might not work properly and a different approach is recommended. Therefore, an additional Java based script has been created that restarts the workflow for all the workflowitems that exist in the original workflow framework. The script will take all the existing workflowitems and place them in the first step of the XML configurable workflow framework thereby taking into account the XML configuration that exists at that time for the collection to which the item has been submitted. This script can also be used to restart the workflow for workflowitems in the original workflow but not to restart the workflow for items in the XML configurable workflow.

To execute the script, run the following CLI command: dspace(.bat) dsrun org.dspace.xmlworkflow.migration.RestartWorkflow -e admin@myrespository.org

The following arguments can be specified when running the script:

- -e: specifies the username of an adminstrator user
- -n: if sending submissions through the workflow, send notification emails

- -p: the provenance description to be added to the item
- -h: help

Additional workflow steps/actions and features

Work in progress...

Workflow overview features

A new features has been added to the XML based workflow that resembles the features available in the JSPUI of DSpace that allows administrators to abort workflowitems. The feature added to the XMLUI allows administrators to look at the status of the different workflowitems and look for workflowitems based on the collection to which they have been submitted. Besides that, the administrator has the ability to permanently delete the workflowitem or send the item back to the submitter.

Issues

Curation System

The DSpace 1.7 version of the curation system integration into the original DSpace workflow only exists in the WorkflowManager.advance() method. Before advancing to the next workflow step or archiving the Item, a check is performed to see whether any curation tasks need to be executed/scheduled. The problem is that this check is based on the hardcoded workflow steps that exist in the original workflow. These hardcoded checks are done in the CurationManager and will need to be changed.

Existing issues

- What happens with collection roles after config changes
- What with workflowitems after config changes
- What with undefined outcomes
- Config checker
- Configurable authorizations?