Submission User Interface

DSpace System Documentation: Customizing and Configuring Submission User Interface

This page explains various customization and configuration options that are available within DSpace for the Item Submission user interface.

Understanding the Submission Configuration File

The *[dspace]/config/item-submission.xml* contains the submission configurations for *both* the DSpace JSP user interface (JSPUI) or the DSpace XML user interface (XMLUI or Manakin). This configuration file contains detailed documentation within the file itself, which should help you better understand how to best utilize it.

The Structure of *item-submission.xml*

```
<item-submission>
        <!-- Where submission processes are mapped to specific Collections -->
        <submission-map>
        <name-map collection-handle="default" submission-name="traditional" /> ...
        </submission-map>
        <!-- Where "steps" which are used across many submission processes can be defined in a
           single place. They can then be referred to by ID later. -->
        <step-definitions>
        <step id="collection">
        <processing-class>org.dspace.submit.step.SelectCollectionStep</process;/processing-class>
        <workflow-editable>false</workflow-editable>
        </step>
                ...
        </step-definitions>
        <!-- Where actual submission processes are defined and given names. Each <submission-process> has
           many <step> nodes which are in the order that the steps should be in.-->
        <submission-definitions> <submission-process name="traditional">
            ...
        <!-- Step definitions appear here! -->
        </submission-process>
            ...
        </submission-definitions>
        </item-submission>
```

Because this file is in XML format, you should be familiar with XML before editing this file. By default, this file contains the "traditional" Item Submission Process for DSpace, which consists of the following Steps (in this order):

Select Collection -> Initial Questions -> Describe -> Upload -> Verify -> License -> Complete

If you would like to customize the steps used or the ordering of the steps, you can do so within the <*submission-definition*> section of the *item-submission.xml* .

In addition, you may also specify different Submission Processes for different DSpace Collections. This can be done in the <*submission-map*> section. The *item-submission.xml* file itself documents the syntax required to perform these configuration changes.

Defining Steps (<*step*>) within the *item-submission.xml*

This section describes how Steps of the Submission Process are defined within the *item-submission.xml*.

Where to place your <*step*> definitions

<*step*> definitions can appear in one of two places within the *item-submission.xml* configuration file.

1. Within the <*step-definitions*> section
    - This is for globally defined <*step*> definitions (i.e. steps which are used in multiple <*submission-process*> definitions). Steps defined in this section **must** define a unique *id* which can be used to reference this step.
    - For example:

```
<step-definitions>
        <step id="custom-step">
           ...
        </step>
          ...
</step-definitions>
```

    - The above step definition could then be referenced from within a <*submission-process*> as simply <*step id="custom-step"/*>
2. Within a specific <*submission-process*> definition
    - This is for steps which are specific to a single <*submission-process*> definition.
    - For example:

```
<submission-process>
        <step>
           ...
        </step>
</submission-process>
```

The ordering of <*step*> definitions matters!

The ordering of the <*step*> tags within a <*submission-process*> definition directly corresponds to the order in which those steps will appear!

For example, the following defines a Submission Process where the *License* step directly precedes the *Initial Questions* step (more information about the structure of the information under each <step> tag can be found in the section on Structure of the <step> Definition below):

```
<submission-process>
        <!--Step 1 will be to Sign off on the License-->
           <step>
             <heading>submit.progressbar.license</heading>
             <processing-class>org.dspace.submit.step.LicenseStep</processing-classing-class>
             <jspui-binding>org.dspace.app.webui.submit.step.JSPLicenseStep</jspui-binding>
             <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.LicenseStenseStep</xmlui-binding>
             <workflow-editable>false</workflow-editable>
           </step>
        <!--Step 2 will be to Ask Initial Questions-->
        <step>
           <heading>submit.progressbar.initial-questions</heading>
              <processing-class>org.dspace.submit.step.InitialQuestionsStep</process;/processing-class>
              <jspui-binding>org.dspace.app.webui.submit.step.JSPInitialQuestionsSteonsStep</jspui-binding>
              <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.InitialQutialQuestionsStep</xmlui-
binding>
              <workflow-editable>true</workflow-editable>
           </step>
              ...[other steps]...
</submission-process>
```

Structure of the <step> Definition

The same <step> definition is used by both the DSpace JSP user interface (JSPUI) an the DSpace XML user interface (XMLUI or Manakin). Therefore, you will notice each <step> definition contains information specific to each of these two interfaces.

The structure of the <step> Definition is as follows:

```
<step>
        <heading>submit.progressbar.describe</heading>
        <processing-class>org.dspace.submit.step.DescribeStep</processing-classing-class>
        <jspui-binding>org.dspace.app.webui.submit.step.JSPDescribeStep</jspuilt;/jspui-binding>
        <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.DescribeScribeStep</xmlui-binding>
        <workflow-editable>true</workflow-editable>
</step>
```

Each *step* contains the following elements. The required elements are so marked:

- **heading**: Partial I18N key (defined in *Messages.properties* for JSPUI or *messages.xml* for XMLUI) which corresponds to the text that should be displayed in the submission Progress Bar for this step. This partial I18N key is prefixed within either the Messages.properties or messages.xml file, depending on the interface you are using. Therefore, to find the actual key, you will need to search for the partial key with the following prefix:
    - XMLUI: prefix is *xmlui.Submission.* (e.g. "xmlui.Submission.submit.progressbar.describe" for 'Describe' step)
    - JSPUI: prefix is *jsp.* (e.g. "jsp.submit.progressbar.describe" for 'Describe' step)*The 'heading' need not be defined if the step should not appear in the progress bar (e.g. steps which perform automated processing, i.e. non-interactive, should not appear in the progress bar).*
- **processing-class** (Required): Full Java path to the Processing Class for this Step. This Processing Class **must** perform the primary processing of any information gathered in this step, for both the XMLUI and JSPUI. All valid step processing classes must extend the abstract `org.dspace.submit.AbstractProcessingStep` class (or alternatively, extend one of the pre-existing step processing classes in `org.dspace.submit.step.*`)
- **jspui-binding**: Full Java path of the JSPUI "binding" class for this Step. This "binding" class should initialize and call the appropriate JSPs to display the step's user interface. A valid JSPUI "binding" class *must* extend the abstract `org.dspace.app.webui.submit.JSPStep` class. *Thi s property need not be defined if you are using the XMLUI interface, or for steps which only perform automated processing, i.e. non-interactive steps.*
- **xmlui-binding**: Full Java path of the XMLUI "binding" class for this Step. This "binding" class should generate the Manakin XML (DRI document) necessary to generate the step's user interface. A valid XMLUI "binding" class *must* extend the abstract `org.dspace.app.xmlui.submission.AbstractSubmissionStep` class. *This property need not be defined if you are using the JSPUI interface, or for steps which only perform automated processing, i.e. non-interactive steps.*
- **workflow-editable**: Defines whether or not this step can be edited during the *Edit Metadata* process with the DSpace approval/rejection workflow process. Possible values include *true* and *false*. If undefined, defaults to *true* (which means that workflow reviewers would be allowed to edit information gathered during that step).

Reordering/Removing Submission Steps

The removal of existing steps and reordering of existing steps is a relatively easy process!

Reordering steps

1. Locate the <*submission-process*> tag which defines the Submission Process that you are using. If you are unsure which Submission Process you are using, it's likely the one with *name="traditional"*, since this is the traditional DSpace submission process.
2. Reorder the <*step*> tags within that <*submission-process*> tag. Be sure to move the *entire*<*step*> tag (i.e. everything between and including the opening <*step*> and closing <*/step*> tags).
    - *Hint #1:* The <*step*> defining the *Review/Verify* step only allows the user to review information from steps which appear **before** it. So, it's likely you'd want this to appear as one of your last few steps
    - *Hint #2:* If you are using it, the <*step*> defining the *Initial Questions* step should always appear **before** the *Upload* or *Describe* steps since it asks questions which help to set up those later steps.

Removing one or more steps

1. Locate the <*submission-process*> tag which defines the Submission Process that you are using. If you are unsure which Submission Process you are using, it's likely the one with *name="traditional"*, since this is the traditional DSpace submission process.
2. Comment out (i.e. surround with `<!--` and `-->`) the `<step>` tags which you want to remove from that <*submission-process*> tag. Be sure to comment out the *entire {{<step>}}tag (i.e. everything between and including the opening _<step>* and closing *</step>* tags).
    - *Hint #1:* You cannot remove the *Select a Collection* step, as an DSpace Item cannot exist without belonging to a Collection.
    - *Hint #2:* If you decide to remove the <*step*> defining the *Initial Questions* step, you should be aware that this may affect your *Describe* and *Upload* steps! The *Initial Questions* step asks questions which help to initialize these later steps. If you decide to remove the *Initial Questions* step you may wish to create a custom, automated step which will provide default answers for the questions asked!

Assigning a custom Submission Process to a Collection

Assigning a custom submission process to a Collection in DSpace involves working with the *submission-map* section of the *item-submission.xml*. For a review of the structure of the *item-submission.xml* see the section above on Understanding the Submission Configuration File.

Each *name-map* element within *submission-map* associates a collection with the name of a submission definition. Its *collection-handle* attribute is the Handle of the collection. Its *submission-name* attribute is the submission definition name, which must match the *name* attribute of a *submission-process* element (in the *submission-definitions* section of *item-submission.xml*.

For example, the following fragment shows how the collection with handle "12345.6789/42" is assigned the "custom" submission process:

```
<submission-map>
    <name-map collection-handle=" 12345.6789/42" submission-name="
        custom" />
    ...
  </submission-map>

  <submission-definitions>
    <submission-process name="
        custom">
    ...
  </submission-definitions>
```

It's a good idea to keep the definition of the *default* name-map from the example *input-forms.xml* so there is always a default for collections which do not have a custom form set.

Getting A Collection's Handle

You will need the *handle* of a collection in order to assign it a custom form set. To discover the handle, go to the "Communities & Collections" page under "Browse" in the left-hand menu on your DSpace home page. Then, find the link to your collection. It should look something like:

```
http://myhost.my.edu/dspace/handle/12345.6789/42
```

The underlined part of the URL is the handle. It should look familiar to any DSpace administrator. That is what goes in the *collection-handle* attribute of your *name-map* element.

Custom Metadata-entry Pages for Submission

Introduction

This section explains how to customize the Web forms used by submitters and editors to enter and modify the metadata for a new item. These metadata web forms are controlled by the *Describe* step within the Submission Process. However, they are also configurable via their own XML configuration file (*input-forms.xml*).

You can customize the "default" metadata forms used by all collections, and also create alternate sets of metadata forms and assign them to specific collections. In creating custom metadata forms, you can choose:

- The number of metadata-entry pages.
- Which fields appear on each page, and their sequence.
- Labels, prompts, and other text associated with each field.
- List of available choices for each menu-driven field.

NOTE: The cosmetic and ergonomic details of metadata entry fields remain the same as the fixed metadata pages in previous DSpace releases, and can only be altered by modifying the appropriate stylesheet and JSP pages.

All of the custom metadata-entry forms for a DSpace instance are controlled by a single XML file, *input-forms.xml*, in the *config* subdirectory under the DSpace home. DSpace comes with a sample configuration that implements the traditional metadata-entry forms, which also serves as a well-documented example. The rest of this section explains how to create your own sets of custom forms.

Describing Custom Metadata Forms

The description of a set of pages through which submitters enter their metadata is called a *form* (although it is actually a set of forms, in the HTML sense of the term). A form is identified by a unique symbolic *name*. In the XML structure, the *form* is broken down into a series of *pages*: each of these represents a separate Web page for collecting metadata elements.

To set up one of your DSpace collections with customized submission forms, first you make an entry in the *form-map*. This is effectively a table that relates a collection to a form set, by connecting the collection's *Handle* to the form name. Collections are identified by handle because their names are mutable and not necessarily unique, while handles are unique and persistent.

A special map entry, for the collection handle "default", defines the *default* form set. It applies to all collections which are not explicitly mentioned in the map. In the example XML this form set is named *traditional* (for the "traditional" DSpace user interface) but it could be named anything.

The Structure of *input-forms.xml*

The XML configuration file has a single top-level element, *input-forms*, which contains three elements in a specific order. The outline is as follows:

```
<input-forms>

  <--  Map of Collections to Form Sets -->
  <form-map>
    <name-map collection-handle="default" form-name="traditional"
        />
    ...
  </form-map>

  <--  Form Set Definitions -->
  <form-definitions>
    <form name="traditional">
    ...
  </form-definitions>

  <--  Name/Value Pairs used within Multiple Choice Widgets
       -->
  <form-value-pairs>
    <value-pairs value-pairs-name="common_iso_languages"
        dc-term="language_iso">
    ...
  </form-value-pairs>
</input-forms>
```

Adding a Collection Map

Each *name-map* element within *form-map* associates a collection with the name of a form set. Its *collection-handle* attribute is the Handle of the collection, and its *form-name* attribute is the form set name, which must match the *name* attribute of a *form* element.

For example, the following fragment shows how the collection with handle "12345.6789/42" is attached to the "TechRpt" form set:

```
  <form-map>
    <name-map collection-handle=" 12345.6789/42" form-name=" TechRpt"/>
    ...
  </form-map>

  <form-definitions>
    <form name="TechRept">
    ...
  </form-definitions>
```

It's a good idea to keep the definition of the *default* name-map from the example *input-forms.xml* so there is always a default for collections which do not have a custom form set.

Getting A Collection's Handle

You will need the *handle* of a collection in order to assign it a custom form set. To discover the handle, go to the "Communities & Collections" page under "**Browse**" in the left-hand menu on your DSpace home page. Then, find the link to your collection. It should look something like:

```
http://myhost.my.edu/dspace/handle/12345.6789/42
```

The underlined part of the URL is the handle. It should look familiar to any DSpace administrator. That is what goes in the *collection-handle* attribute of your *name-map* element.

Adding a Form Set

You can add a new form set by creating a new *form* element within the *form-definitions* element. It has one attribute, *name*, which as seen above must match the value of the *name-map* for the collections it is to be used for.

Forms and Pages

The content of the *form* is a sequence of *page* elements. Each of these corresponds to a Web page of forms for entering metadata elements, presented in sequence between the initial "Describe" page and the final "Verify" page (which presents a summary of all the metadata collected).

A *form* must contain at least one and at most six pages. They are presented in the order they appear in the XML. Each *page* element must include a *numbe* attribute, that should be its sequence number, e.g.

```
<page number="1">
```

The *page* element, in turn, contains a sequence of *field* elements. Each field defines an interactive dialog where the submitter enters one of the Dublin Core metadata items.

Composition of a Field

Each *field* contains the following elements, in the order indicated. The required sub-elements are so marked:

- **dc-schema** (Required) : Name of metadata schema employed, e.g. *dc* for Dublin Core. This value must match the value of the *schema* element defined in *dublin-core-types.xml*
- **dc-element** (Required) : Name of the Dublin Core element entered in this field, e.g. *contributor*.
- **dc-qualifier**: Qualifier of the Dublin Core element entered in this field, e.g. when the field is *contributor.advisor* the value of this element would be *advisor*. Leaving this out means the input is for an unqualified DC element.
- **repeatable**: Value is *true* when multiple values of this field are allowed, *false* otherwise. When you mark a field repeatable, the UI servlet will add a control to let the user ask for more fields to enter additional values. Intended to be used for arbitrarily-repeating fields such as subject keywords, when it is impossible to know in advance how many input boxes to provide.
- **label** (Required): Text to display as the label of this field, describing what to enter, e.g. "*Your Advisor's Name*".
- **input-type** (Required): Defines the kind of interactive widget to put in the form to collect the Dublin Core value. Content must be one of the following keywords:
    - **onebox** – A single text-entry box.
    - **twobox** – A pair of simple text-entry boxes, used for *repeatable* values such as the DC *subject* item. *Note:* The 'twobox' input type is rendered the same as a 'onebox' in the XML-UI, but both allow for ease of adding multiple values.
    - **textarea** – Large block of text that can be entered on multiple lines, e.g. for an abstract.
    - **name** – Personal name, with separate fields for family name and first name. When saved they are appended in the format 'LastName, FirstName'
    - **date** – Calendar date. When required, demands that at least the year be entered.
    - **series** – Series/Report name and number. Separate fields are provided for series name and series number, but they are appended (with a semicolon between) when saved.
    - **dropdown** – Choose value(s) from a "drop-down" menu list. **Note:** You must also include a value for the *value-pairs-name* attribute to specify a list of menu entries from which to choose. Use this to make a choice from a restricted set of options, such as for the *language* item.
    - **qualdrop_value** – Enter a "qualified value", which includes *both* a qualifier from a drop-down menu and a free-text value. Used to enter items like alternate identifiers and codes for a submitted item, e.g. the DC *identifier* field. **Note:** As for the *dropdown* type, you must include the *value-pairs-name* attribute to specify a menu choice list.
    - **list** – Choose value(s) from a checkbox or radio button list. If the *repeatable* attribute is set to *true*, a list of checkboxes is displayed. If the *repeatable* attribute is set to *false*, a list of radio buttons is displayed. **Note:** You must also include a value for the *value-pairs-name* attribute to specify a list of values from which to choose.
- **hint** (Required): Content is the text that will appear as a "hint", or instructions, next to the input fields. Can be left empty, but it must be present.
- **required**: When this element is included with any content, it marks the field as a required input. If the user tries to leave the page without entering a value for this field, that text is displayed as a warning message. For example, *<required>You must enter a title.</required> Note that leaving the* required element empty will *not* mark a field as required, e.g.:*<required></required>*
- **visibility**: When this optional element is included with a value, it restricts the visibility of the field to the scope defined by that value. If the element is missing or empty, the field is visible in all scopes. Currently supported scopes are:
    - **workflow** : the field will only be visible in the workflow stages of submission. This is good for hiding difficult fields for users, such as subject classifications, thereby easing the use of the submission system.
    - **submit** : the field will only be visible in the initial submission, and not in the workflow stages. In addition, you can decide which type of restriction apply: read-only or full hidden the field (default behaviour) using the *otherwise* attribute of the *visibility* XML element. For example:*<visibility otherwise="readonly">workflow</visibility>* Note that it is considered a configuration error to limit a field's scope while also requiring it - an exception will be generated when this combination is detected.
      Look at the example *input-forms.xml* and experiment with a a trial custom form to learn this specification language thoroughly. It is a very simple way to express the layout of data-entry forms, but the only way to learn all its subtleties is to use it.

For the use of controlled vocabularies see the Configuring Controlled Vocabularies section.

Automatically Elided Fields

You may notice that some fields are automatically skipped when a custom form page is displayed, depending on the kind of item being submitted. This is because the DSpace user-interface engine skips Dublin Core fields which are not needed, according to the initial description of the item. For example, if the user indicates there are no alternate titles on the first "Describe" page (the one with a few checkboxes), the input for the *title.alternative* DC element is automatically elided, *even on custom submission pages.*

When a user initiates a submission, DSpace first displays what we'll call the "initial-questions page". By default, it contains three questions with check-boxes:

1. **The item has more than one title, e.g. a translated title** Controls *title.alternative* field.
2. **The item has been published or publicly distributed before** Controls DC fields:
    - *date.issued*
    - *publisher*
    - *identifier.citation*
3. **The item consists of more than one file** *Does not affect any metadata input fields.*

The answers to the first two questions control whether inputs for certain of the DC metadata fields will displayed, even if they are defined as fields in a custom page. Conversely, if the metadata fields controlled by a checkbox are not mentioned in the custom form, the checkbox is elided from the initial page to avoid confusing or misleading the user.

The two relevant checkbox entries are "The item has more than one title, e.g. a translated title", and "The item has been published or publicly distributed before". The checkbox for multiple titles trigger the display of the field with dc-element equal to 'title' and dc-qualifier equal to 'alternative'. If the controlling collection's form set does not contain this field, then the multiple titles question will not appear on the initial questions page.

Adding Value-Pairs

Finally, your custom form description needs to define the "value pairs" for any fields with input types that refer to them. Do this by adding a *value-pairs* element to the contents of *form-value-pairs*. It has the following required attributes:

- **value-pairs-name** – Name by which an *input-type* refers to this list.
- **dc-term** – Qualified Dublin Core field for which this choice list is selecting a value. Each *value-pairs* element contains a sequence of *pair* sub-elements, each of which in turn contains two elements:
- **displayed-value** – Name shown (on the web page) for the menu entry.
- **stored-value** – Value stored in the DC element when this entry is chosen. Unlike the HTML *select* tag, there is no way to indicate one of the entries should be the default, so the first entry is always the default choice.

Example

Here is a menu of types of common identifiers:

```
<value-pairs value-pairs-name="common_identifiers" dc-term="identifier">
  <pair>
    <displayed-value>Gov't Doc #</displayed-value>
    <stored-value>govdoc</stored-value>
  </pair>
  <pair>
    <displayed-value>URI</displayed-value>
    <stored-value>uri</stored-value>
  </pair>
  <pair>
    <displayed-value>ISBN</displayed-value>
    <stored-value>isbn</stored-value>
  </pair>
</value-pairs>
```

It generates the following HTML, which results in the menu widget below. (Note that there is no way to indicate a default choice in the custom input XML, so it cannot generate the HTML *SELECTED* attribute to mark one of the options as a pre-selected default.)

```
<select name="identifier_qualifier_0">
    <option VALUE="govdoc">Gov't Doc #</option>
    <option VALUE="uri">URI</option>
    <option VALUE="isbn">ISBN</option>
</select>
```

Deploying Your Custom Forms

The DSpace web application only reads your custom form definitions when it starts up, so it is important to remember:

- *You must always restart Tomcat* (or whatever servlet container you are using) for changes made to the *input-forms.xml* file take effect.

Any mistake in the syntax or semantics of the form definitions, such as poorly formed XML or a reference to a nonexistent field name, will cause a fatal error in the DSpace UI. The exception message (at the top of the stack trace in the *dspace.log* file) usually has a concise and helpful explanation of what went wrong. Don't forget to stop and restart the servlet container before testing your fix to a bug.

Configuring the File Upload step

The *Upload* step in the DSpace submission process has two configuration options which can be set with your *[dspace]/config/dspace.cfg* configuration file. They are as follows:

- *upload.max* - The maximum size of a file (in bytes) that can be uploaded from the JSPUI (not applicable for the XMLUI). It defaults to 536870912 bytes (512MB). You may set this to -1 to disable any file size limitation.
    - *Note:* Increasing this value or setting to -1 does **not** guarantee that DSpace will be able to successfully upload larger files via the web, as large uploads depend on many other factors including bandwidth, web server settings, internet connection speed, etc.

- *webui.submit.upload.required* - Whether or not all users are *required* to upload a file when they submit an item to DSpace. It defaults to 'true'. When set to 'false' users will see an option to skip the upload step when they submit a new item.

Creating new Submission Steps

First, a brief warning: *Creating a new Submission Step requires some Java knowledge, and is therefore recommended to be undertaken by a Java programmer whenever possible*

That being said, at a higher level, creating a new Submission Step requires the following (in this relative order):

1. (**Required**) Create a new Step Processing class
    - This class **must** extend the abstract `org.dspace.submit.AbstractProcessingStep` class and implement all methods defined by that abstract class.
    - This class should be built in such a way that it can process the input gathered from *either* the XMLUI or JSPUI interface.
2. (*For steps using JSPUI*) Create the JSPs to display the user interface. Create a new JSPUI "binding" class to initialize and call these JSPs.
    - Your JSPUI "binding" class must extend the abstract class `org.dspace.app.webui.submit.JSPStep` and implement all methods defined there. It's recommended to use one of the classes in `org.dspace.app.webui.submit.step.*` as a reference.
    - Any JSPs created should be loaded by calling the showJSP() method of the `org.dspace.app.webui.submit.JSPStepManager` class
    - If this step gathers information to be reviewed, you must also create a Review JSP which will display a read-only view of all data gathered during this step. The path to this JSP must be returned by your getReviewJSP() method. You will find examples of Review JSPs (named similar to `review-[step].jsp`) in the JSP `submit/` directory.
3. (*For steps using XMLUI*) Create an XMLUI "binding" Step Transformer which will generate the DRI XML which Manakin requires.
    - The Step Transformer must extend and implement all necessary methods within the abstract class `org.dspace.app.xmlui.submission.AbstractSubmissionStep`
    - It is useful to use the existing classes in `org.dspace.app.xmlui.submission.submit.*` as references
4. (**Required**) Add a valid Step Definition to the `item-submission.xml` configuration file.
    - This may also require that you add an I18N (Internationalization) key for this step's *heading*. See the sections on Configuring Multilingual Support for JSPUI or Configuring Multilingual Support for XMLUI for more details.
    - For more information on <*step*> definitions within the `item-submission.xml`, see the section above on Defining Steps (<*step*>) within the `item-submission.xml`.

Creating a Non-Interactive Step

Non-interactive steps are ones that have no user interface and only perform backend processing. You may find a need to create non-interactive steps which perform further processing of previously entered information.

To create a non-interactive step, do the following:

1. Create the required Step Processing class, which extends the abstract `org.dspace.submit.AbstractProcessingStep` class. In this class add any processing which this step will perform.
2. Add your non-interactive step to your *item-submission.xml* at the place where you wish this step to be called during the submission process. For example, if you want it to be called *immediately after* the existing 'Upload File' step, then place its configuration immediately after the configuration for that 'Upload File' step. The configuration should look similar to the following:

```
<step>
   <processing-class>org.dspace.submit.step.MyNonInteractiveStep</processing-class>
   <workflow-editable>false</workflow-editable>
</step>
```

Note: Non-interactive steps will not appear in the Progress Bar! Therefore, your submitters will not even know they are there. However, because they are not visible to your users, you should make sure that your non-interactive step does not take a large amount of time to finish its processing and return control to the next step (otherwise there will be a visible time delay in the user interface).