Submission User Interface

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

DSpace Submission Configuration changed in v7.x

The name and structure of the Submission configuration files changed in 7.x. The DSpace 6.x (and below) "item-submission.xml" and "input-forms.xml" configuration files are no longer supported. In 7.x and above, the format of the "item-submission.xml" file has been updated, and the older "input-forms.xml" has been replaced by a new "submission-forms.xml".

You can choose to either start fresh with the new v7 configuration files (see documentation below) and/or use the "./dspace submission-forms-migrate" script to migrate your old configurations into new ones. See the Upgrading DSpace guide (step on "Update your DSpace Configurations") for more information on using the migration script.

Default Submission Process

The DSpace Submission process consists of a series of "steps", where each "step" corresponds to one or "sections" in the Submission UI. By default, the DSpace Submission process includes the following steps/sections, in this order:

1. **"Select Collection"** (id="collection"), appears as dropdown: If not already selected, the user must select a collection to deposit the Item into. As of DSpace 7, you also can change the Collection you are submitting into at any time. However, be aware that there may be some metadata lost if the Collection you switch two uses a *different* submission form & you already began entering metadata in the current submission.
2. **"Describe" sections** (id="traditionalpageone" and "traditionalpagetwo"): This is where the user may enter descriptive metadata about the Item. This step may consist of one or more sections of metadata entry. By default, there are two sections of metadata-entry . For information on modifying the metadata entry pages, please see Custom Metadata-entry Pages for Submission section below.
3. **"Upload" section** (id="upload"): This is where the user may upload one or more files to associate with the Item. As of DSpace 7, you can also drag and drop files anywhere on the page to trigger an upload. For more information on file upload, also see Configuring the File Upload step below.
4. **"License" section** (id="license"): This is where the user **must** agree to the repository distribution license in order to complete the deposit. This repository distribution license is defined in the `[dspace]/config/default.license` file. It can also be customized per-collection from the Collection Edit UI.
5. **"Deposit" button**: Once all required fields/sections are completed, the "Deposit" button becomes enabled. After clicking it, the new Item will either become immediately available or undergo a workflow approval process (depending on the Collection policies). For more information on the workflow approval process see Configurable Workflow

To modify or reorganize these submission steps, just modify the `[dspace]/config/item-submission.xml` file. Please see the section below on Reordering/Removing/Adding Submission Steps.

You can also choose to have different submission processes for different DSpace Collections. For more details, please see the section below on Assigning a custom Submission Process to a Collection.

Optional Steps

DSpace also ships with several optional steps which you may choose to enable if you wish. In no particular order:

- **"Item Access" (or Embargo) section** (id="itemAccessConditions"): *Only available in 7.2 or above.* This step allows the user to (optionally) modify access rights or set an embargo during the deposit of an Item. For more information on this step, and Embargo options in general, please see the Embargo documentation.
- **"CC License" section** (id="cclicense"): This step allows the user to (optionally) assign a Creative Commons license to a particular Item. Please see the Configuring Creative Commons License section of the Configuration documentation for more details.
- **"Extraction" section** (id="extractionstep"): This step will automatically attempt to extract metadata from uploaded files. Currently it only supports bibliographic formats documented in Importing Items via basic bibliographic formats (Endnote, BibTex, RIS, TSV, CSV) and online services (OAI, arXiv, PubMed, CrossRef, CiNii). Any extracted metadata is immediately populated in the submission form (without notifying the user).
  - By default it is disabled, as it populates metadata automatically (without notifying the user). This means it can sometimes result in duplicative metadata in the submission form.
  - The behavior of this step can be more fully configured via the 'config/spring/api/step-processing-listener.xml' configuration
  - NOTE: this action is also only triggered when a request is performed (e.g. when a file is uploaded or the submission form is saved). You can configure the Angular UI to autosave based on a timer in order to force this action to be done more regularly.
- Various Configurable Entities related steps: These steps are "Describe" steps that are specific to different Entity types. They provide a list of metadata fields of specific interest to those Entities.

To enable any of these optional submission steps, just uncomment the step definition within the `[dspace]/config/item-submission.xml` file. Please see the section below on Reordering/Removing/Adding Submission Steps.

You can also choose to enable certain steps only for specific DSpace Collections. For more details, please see the section below on Assigning a custom Submission Process to a Collection.

Understanding the Submission Configuration Files

The `[dspace]/config/item-submission.xml` contains the submission configurations for the DSpace UI. 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*

The structure of this file changed slightly in DSpace 7

As of DSpace 7, the following structural changes were made to item-submission.xml:

- Step definitions under <step-definitions> now use the <step-definition> tag (previously, in 6.x, the tag was named <step>)
- Every step definition now needs to be defined under <step-definitions> (previously, in 6.x, you could also define them in <submission-process>), and have a unique ID
- Each <step-definition> now only represents a single "section" of the Submission UI. (previously, in 6.x, some steps like Describe represented multiple pages)
- An attribute "mandatory=[true|false]" was added to the <step> element. When true, that section is always displayed to the user. When false, it's not displayed by default, but instead must be activated explicitly by the user by choosing to add the section in the Submission UI.
- The old <workflow-editable> element has been replaced with a <scope> element which defines when/how this <step> should be displayed.

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 -> Describe (two steps) -> Upload -> 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-definition*>

The <*step-definition*> always appear within the <*step-definitions*> section of the *item-submission.xml* configuration file.

- This section allows all <*step*> definitions to be defined globally (i.e. so they may be 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-definition 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"/*>

The ordering of <*step*> tags matter!

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 *Describe* 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 id="license"/>

    <!--Step 2 & 3 will be to ask for metadata-->
    <step id="traditionalpageone"/>
    <step id="traditionalpagetwo"/>

        ...[other steps]...
</submission-process>
```

Structure of the <step-definition> tag

The structure of the <step-definition> tag is as follows:

```
<step-definition id="traditionalpageone" mandatory="true">
    <heading>submit.progressbar.describe.stepone</heading>
    <processing-class>org.dspace.app.rest.submit.step.DescribeStep</processing-class>
    <type>submission-form</type>
    <!-- <scope visibility="hidden" visibilityOutside="hidden">submission</scope> -->
</step-definition>
```

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

- **mandatory** *(attribute)*: [true|false] When true, the step's section is displayed by default to all users in the UI. When false, the step is not displayed and must be activated explicitly by the user by selecting it in the UI or supplying data of interest to the section.
- **heading**: Partial I18N key (defined in the UI's language packs) which corresponds to the text that should be displayed in section header for this step. This partial I18N key is prefixed with "submission.sections.". Therefore, the full i18n key is "submission.sections.[heading]" in the User Interface's language packs (e.g. en.json5 for English)
- **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. 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.*`)
- **type (Required)**: The type of step defined. Most steps are of type "submission-form", which means they directly map to a `<form>` defined in the s`ubmission-forms.xml` configuration file. In this situation, the `<step-definition>` "id" attribute MUST map to a `<form>` "name" attribute defined in submission-forms.xml. Any value is allowed, and only "submission-form" has a special meaning at this time.
- **scope:** Optionally, allows you to limit the "scope" of this particular step, and define whether the step is visible outside that scope. Valid scope values include "submission" (limited to the submission form) and "workflow" (limited to workflow approval process).
    - "visibility" attribute defines the visibility of the step while **within** the given scope. Can be set to "read-only" (in this scope you can see this step but not edit it), or "hidden" (in this scope you cannot see this step).
    - "visibilityOutside" attribute defines the visibility of the step while **outside** the given scope. Can be set to "read-only" (in other scopes you can see this step but not edit it), or "hidden" (in other scopes you cannot see this step).

Reordering/Removing/Adding 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.

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.*
    - *Hint:* You cannot remove the *"collection"* step, as an DSpace Item cannot exist without belonging to a Collection.

Adding one or more optional 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. Uncomment (i.e. remove the `<! --` and `-->`) the `<step>` tag(s) which you want to add to that `<submission-process>` tag. Be sure to uncomment the *entire* `<step>` *tag.*

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.

There are two ways to configure this mapping:

1. The traditional way is to use the "collection-handle" attribute to map a submission form to it's Collection. 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*.
    a. 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>
```

2. As of 7.6, another option is to use the "collection-entity-type" attribute to map *all Collections* which use that Entity Type (requires Configurable Entities) to a specific submission definition name (via the *submission-name* attribute, similar to above).
    a. For example, the following fragment shows how to map all Collections which use the out-of-the-box Entity Types to a submission definition of the same name:

```
<submission-map>
    ...
    <name-map collection-entity-type="Publication" submission-name="Publication"/>
    <name-map collection-entity-type="Person" submission-name="Person"/>
    <name-map collection-entity-type="Project" submission-name="Project"/>
    <name-map collection-entity-type="OrgUnit" submission-name="OrgUnit"/>
    <name-map collection-entity-type="Journal" submission-name="Journal"/>
    <name-map collection-entity-type="JournalVolume" submission-name="JournalVolume"/>
    <name-map collection-entity-type="JournalIssue" submission-name="JournalIssue"/>
    ...
<submission-map>
```

It's a good idea to keep the definition of the *default* name-map, 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 Community or Collection in the DSpace UI. Look for the "Permanent URI" listed near the top of the page. It should look something like:

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

The handle is everything after "handle/" (in the above example it is "12345.6789/42"). It should look familiar to any DSpace administrator. That is what goes in the *collection-handle* attribute of your *name-map* element.

Assigning a default Submission Process per Entity Type

Alternatively to a collection's Handle, Entities Types can be used as an attribute. With these configurations you will enable default submission forms per Entity type. You don't have to specify every collection's handle to use for a particular submission form if you intend to use entities.
In order to do it so, instead of `collection-handle` attribute you need to use `collection-entity-type`. The possible values for this attribute are the ones that you use or that you specified in `relationship-types.xml` file (please check the documentation for more information). In order the submission process to be assigned to an entity type, you need to previously have associated an Entity Type to a Collection (please check: Configurable Entities#3.ConfigureCollectionsforeachEntitytype).

As an example, for every time you need to insert a new person in a Person's collection. You just need to specify the submission form to be used, like: `submission-name="customPerson"` in the example and also the entity type that is associated, like `collection-entity-type="Person"`.

```
<submission-map>
    <name-map collection-entity-type="Person" submission-name="customPerson" />
    ...
</submission-map>

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

If a collection `collection-handle="12345.6789/42"` configuration will prevail over this configuration. Meaning that if a `collection-entity-type` is defined and a collection-handle is also defined and if a collection handle overlaps in both configurations, then, the submission to be considered it will be the one that is defined by `collection-handle` (it will prevail the one with more granularity).

Custom Metadata-entry Steps 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 `[dspace]/config/submission-forms.xml`.

In this configuration you can create alternate metadata forms, which can then be mapped to a "submission-form" step in the "item-submission.xml" (see above).

In creating custom metadata forms, you can choose:

- Which fields appear on each form, and their sequence. (Keep in mind, each "form" represents to a "step" or section)
- Labels, prompts, and other text associated with each field.
- Ability to display smaller fields side-by-side in a single "row"
- List of available choices for each menu-driven field.

All of the custom metadata-entry forms for a DSpace instance are controlled by a single XML file, `submission-forms.xml`, in the *config* subdirectory under the DSpace home, `[dspace]/config/submission-forms.xml`. DSpace comes with a number of sample forms which implement the traditional metadata-entry forms, and also serves as a well-documented example. Some default forms include:

- "bitstream-metadata" - This is a special form which defines the metadata fields available for every uploaded bitstream (file)
- "traditionalpageone" - A sample form which is used by the first "Describe" step defined in item-submission.xml
- "traditionalpagetwo" - A sample form which is used by the second "Describe" step defined in item-submission.xml
- A number of sample forms for various out-of-the-box Configurable Entities. These forms all have a corresponding <step> defined in item-submission.xml. In conjunction to those <step> definitions, these forms may be used to submit new Entities of specific types. Usually this is done by mapping that Entity-specific submission-process (in item-submission.xml) to a Collection which is used for new submissions of that Entity.

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 fields through which submitters enter their metadata is called a *form* (in the UI, each "form" is displayed in a separate collapsible section). A form is identified by a unique symbolic *name*. In the XML structure, the *form* is broken down into *rows of fields*. This allows you to place smaller fields side-by-side in a single, horizontal row, or alternatively decide to display one field per row.

The Structure of *submission-forms.xml*

The name & structure of this file changed slightly in DSpace 7

As of DSpace 7, the following structural changes were made to this configuration:

- input-forms.xml (v6) was renamed to submission-forms.xml
- <form-map> top-level element was removed. All Collection mappings are now in item-submission.xml
- <page> element under <form> was removed. As described below, <form> element now represent a single section of the submission process.
- <row> element under <form> was added. As described below, multiple fields can now be displayed in one horizontal row.
- A new form named "bitstream-metadata" was introduced to allow you to configure which metadata is requested for a bitstream during submission.

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

```
<input-forms>

    <--  Form Set Definitions -->
    <form-definitions>
        <form name="traditionalpageone">
            ...
        </form>
        ...
    </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">
            ...
        </value-pairs>
        ...
    </form-value-pairs>
</input-forms>
```

Using a form in a submission process for a Collection

Keep in mind, the "submission-forms.xml" only defines *forms* and *value-pairs* (used for specific fields like selectboxes). To enable a form requires also updating the "item-submission.xml" configuration to use that form (see also above):

1. In "item-submission.xml", a <step-definition> of type "submission-form" must be created, with an "id" matching the *name* of the *form* (see above for more details on step-definition)
2. In "item-submission.xml", a <submission-process> must be created/updated to use that newly defined "step".
3. Finally, also in "item-submission.xml", a Collection must be setup to use that submission process in the <submission-map> section.

So, if you modify submission-forms.xml, you may need to double check your changes will be used in your item-submission.xml.

Adding a Form

You can add a new form by creating a new *form* element within the *form-definitions* element. It has one attribute, *name*, which as described above must match the "id" of a <step-definition> in "item-submission.xml".

Forms and Pages

The content of the *form* is a sequence of *row* elements. Each of these corresponds to a single, horizontal row, containing metadata input fields. The rows are presented in sequence, with the first row displayed at the top of the form. A form is displayed as a section (or step) within the submission process.

A *form* may contain any number of rows. A row generally only contains one or two input fields (including more than one input field may require the "style" setting, see below). 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.
- **language**: If set to *true* a drop down menu will be shown, containing languages. The selected language will be used as language tag of the metadata field. A compulsory argument *value-pairs-name* must be given containing the name of the value pair that contains all the languages: e. g. `<language value-pairs-name="common_iso_languages">true</language>`.
- **repeatable**: Value is *true* when multiple values of this field are allowed, *false* otherwise. When you mark a field repeatable, the UI will add an "Add more" control to the field, allowing the user to 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. Repeatable fields also support reordering of values.
- **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 (i.e. a normal input textbox)
    - **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'. *(By default, this input type is unused. Author fields now use the "onebox" type to support different types of names.)*
    - **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.
        - Requires that you 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.
        - Similar to the *dropdown* type, *requires* that you include the *value-pairs-name* attribute to specify a menu choice list.
        - Because the "qualdrop_value" dynamically sets the *qualifier* (based on the drop-down menu), the <dc-qualifier> field *MUST* be empty. The <dc-qualifier> element cannot be used with this field type.
    - **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. *(By default, this input type is unused.)*
        - Requires that you include a value for the *value-pairs-name* attribute to specify a list of values from which to choose.
    - **tag** - A free-text field which allows you to add multiple labels/tags as values. An example is the "Subject Keywords" field.
        - Note: A tag field MUST be marked as <repeatable>true</repeatable>.
- **hint** (Required): Content is the text that will appear as a "hint", or instructions, below the input field. Can be left empty, but the tag must be present.
- **required**: When this element is included with any content, it marks the field as a required input. If the user saves the form 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>*
- **vocabulary:** When specified, this field uses a controlled vocabulary defined in `[dspace]/config/controlled-vocabularies/[name].xml`. This setting may be used to provide auto-complete functionality, for example in the "Subject Keywords" field (which uses the "tag" input type). See also the "Configuring Controlled Vocabularies" section below.
- **regex:** When specified, this field will be validated against the Regular Expression, and only successfully validating values will be saved. An example is commented out in the default "Author" field. If the validation fails, the following error message will be shown by default: *"This input is restricted by the current pattern: {{ pattern }}."*. This can be customized, by adding an entry to the internalization files with the key `error.validation.pattern.schema_element_qualifier` and the schema, element and qualifier of the field. For example: *"error.validation.pattern.dc_identifier": "The identifier can only consist of numbers"*. For instructions on how to add custom entries see: Customize UI labels using Internationalization (i18n) files
- **style:** When specified, this provides a CSS style recommendation to the UI for how to style that field. This is primarily used when displaying multiple fields per row, so that you can tell the UI how many columns each field should use in that row. Keep in mind, these styles should follow the Bootstrap Grid System, where the number of columns adds up to 12. An example can be see in the default "Date of Issue" and "Publisher" fields, which are configured to use 4 (col-sm-4) and 8 (col-sm-8) columns respectively.
- **visibility:** the submission scope for which the field should be visible. Values allowed are *submission* or *workflow.* When one of the two options is given the filed will be visible only for the scope provided and it will be hidden otherwise.
- **readonly:** this option can be used only together with the *visibility* element, and it means the field should be a read-only input instead of being hidden out of the scope provided by the *visibility* element. The value allowed is *readonly*, e.g.: *<readonly>readonly</readonly>*

Visibility configuration examples

A field configured to be visible only with *submission* scope, while is hidden with *workflow* scope

```
<field>
    <dc-schema>dc</dc-schema>
    <dc-element>title</dc-element>
    <dc-qualifier>alternative</dc-qualifier>
    <repeatable>true</repeatable>
    <label>Other Titles</label>
    <input-type>onebox</input-type>
    <hint>If the item has any alternative titles, please enter them here.</hint>
    <required></required>
    <visibility>submission</visibility>
</field>
```

A field configured to be visible only with *workflow* scope, while is read-only with *submission* scope

```
<field>
    <dc-schema>dc</dc-schema>
    <dc-element>title</dc-element>
    <dc-qualifier>alternative</dc-qualifier>
    <repeatable>true</repeatable>
    <label>Other Titles</label>
    <input-type>onebox</input-type>
    <hint>If the item has any alternative titles, please enter them here.</hint>
    <required></required>
```

```
    <readonly>readonly</readonly>
    <visibility>workflow</visibility>
</field>
```

Item type Based Metadata Collection

Available in 7.3 and later

A field can be made visible depending on the value of *dc.type*. A new field element, <type-bind>, has been introduced to facilitate this. The <type-bind> takes a comma separated list of publication types. If the field is missing or empty, it will always be visible. In this example the field will only be visible if a value of "thesis" or "ebook" has been entered into *dc.type* on an earlier page:

```
<field>
    <dc-schema>dc</dc-schema>
    <dc-element>identifier</dc-element>
    <dc-qualifier>isbn</dc-qualifier>
    <label>ISBN</label>
    <type-bind>thesis,ebook</type-bind>
</field>
```

A field may be configured multiple times in the submission configuration with different values in type-bind. This is useful if a field is required for one type but not another, or should display a different label and hint message depending on the publication type:

```
<field>
    <dc-schema>dc</dc-schema>
    <dc-element>identifier</dc-element>
    <dc-qualifier>isbn</dc-qualifier>
    <label>ISBN</label>
    <type-bind>book,ebook</type-bind>
        <required>You must enter an ISBN for this book</required>
</field>

<field>
    <dc-schema>dc</dc-schema>
    <dc-element>identifier</dc-element>
    <dc-qualifier>isbn</dc-qualifier>
    <label>ISBN of Parent Publication</label>
    <type-bind>thesis,book chapter,letter</type-bind>
        <hint>Enter the ISBN of the book in which this was published</hint>
</field>
```

If a field is *required* but is bound to a type that does not match the submitted publication, the *required* value will be ignored.

Note: When the submitter changes the Type field, other fields (usually just below it) dynamically appear. There's a brief demo of this feature in the 2022-07-13 - DSpace 7 Q&A webinar at time 19:05. The submission process is one page, but it has collapsible sections, each of which corresponds to one of the old "pages".

Configuring Controlled Vocabularies

DSpace supports controlled vocabularies to confine the set of keywords that users can use while describing items. The need for a limited set of keywords is important since it eliminates the ambiguity of a free description system, consequently simplifying the task of finding specific items of information. The controlled vocabulary allows the user to choose from a defined set of keywords organized in an tree (taxonomy) and then use these keywords to describe items while they are being submitted.

The taxonomies are described in XML following this (very simple) structure:

```
<node id="acmccs98" label="ACMCCS98">
    <isComposedBy>
        <node id="A." label="General Literature">
            <isComposedBy>
                <node id="A.0" label="GENERAL"/>
                <node id="A.1" label="INTRODUCTORY AND SURVEY"/>
                ...
            </isComposedBy>
        </node>
        ...
    </isComposedBy>
</node>
```

You are free to use any application you want to create your controlled vocabularies. A simple text editor should be enough for small projects. Bigger projects will require more complex tools. You may use Protegé to create your taxonomies, save them as OWL and then use a XML Stylesheet (XSLT) to transform your documents to the appropriate format. Future enhancements to this add-on should make it compatible with standard schemas such as OWL or RDF.

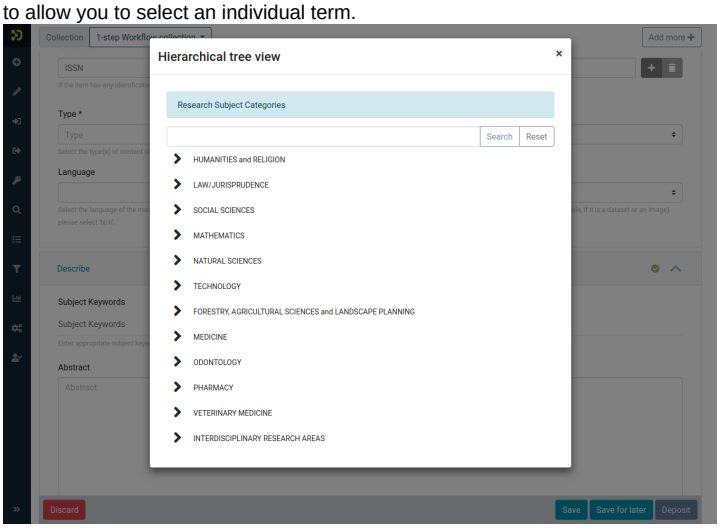
New vocabularies should be placed in `[dspace]/config/controlled-vocabularies/` and must be according to the structure described.

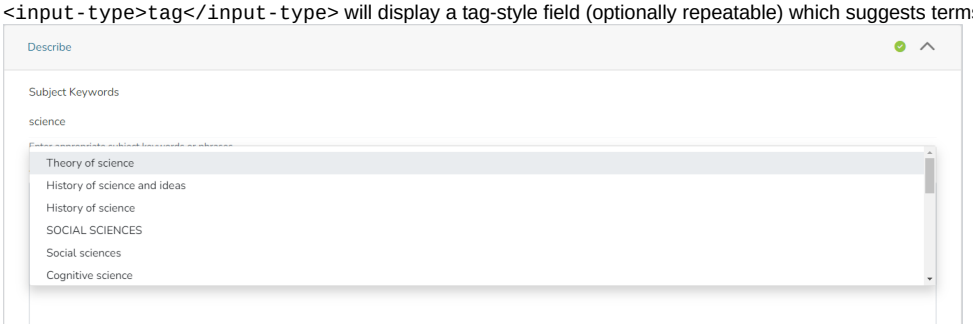
Vocabularies need to be associated with the corresponding metadata fields. Edit the file `[dspace]/config/submission-forms.xml` and place a `"vocabulary"` tag under the `"field"` element that you want to control. Set value of the `"vocabulary"` element to the name of the file that contains the vocabulary, leaving out the extension (the add-on will only load files with extension "*.xml"). For example:

```
<field>
    <dc-schema>dc</dc-schema>
    <dc-element>subject</dc-element>
    <dc-qualifier></dc-qualifier>
    <repeatable>true</repeatable>
    <label>Subject Keywords</label>
    <input-type>onebox</input-type>
    <hint>Enter appropriate subject keywords or phrases below.</hint>
    <required></required>
    <vocabulary>srsc</vocabulary>
</field>
```

The vocabulary element has an optional boolean attribute closed that can be used to force input only with the Javascript of controlled-vocabulary add-on. The default behaviour (i.e. without this attribute) is as set closed="false". This allow the user also to enter the value in free way.

Controlled vocabularies have two main display types in the submission form:

1. `<input-type>onebox</input-type>` will display a onebox style field (optionally repeatable) which pops up the entire hierarchical vocabulary to allow you to select an individual term.


2. `<input-type>tag</input-type>` will display a tag-style field (optionally repeatable) which suggests terms within the vocabulary as you type.


The following vocabularies are currently available by default:

- **nsi** - *nsi.xml* - The Norwegian Science Index
- **srsc** - *srsc.xml* - Swedish Research Subject Categories

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** – 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 *submission-forms.xml* and/or *item-submission.xml* to 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, may result in errors in the DSpace REST API & 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

Basic Settings

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

- *spring.servlet.multipart.max-file-size (default=512MB)* - Spring Boot's maximum allowable file upload size. For DSpace, we default it to 512MB (in application.properties). But, you may wish to override the default value in your local.cfg. Example values include "512MB", "1GB", or even "-1" (to allow unlimited). See Spring's documentation on this setting: https://spring.io/guides/gs/uploading-files/#_tuning_file_upload_limits
    - NOTE: Increasing this value significantly does NOT guarantee that DSpace will be able to successfully upload files of a very large size via the web. Large uploads depend on many other factors including bandwidth, web server settings, internet connection speed, etc. Therefore, for very large files, you may need to consider importing via command-line tools or similar.
- *spring.servlet.multipart.max-request-size (default=512MB)* - Spring Boot's maximum allowable upload *request* size (i.e. the maximum total upload size for all files in a multi-file upload). For DSpace, we default it to 512MB (in application.properties). But, you may wish to override the default value in your local.cfg. Example values include "512MB", "1GB", or even "-1" (to allow unlimited). See Spring's documentation on this setting: https://spring.io/guides/gs/uploading-files/#_tuning_file_upload_limits

    - NOTE: Increasing this value significantly does NOT guarantee that DSpace will be able to successfully upload files of a very large size via the web. Large uploads depend on many other factors including bandwidth, web server settings, internet connection speed, etc. Therefore, for very large files, you may need to consider importing via command-line tools or similar.
- *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.

Modifying metadata form presented for Bitstreams

After uploading a file (bitstream) in the Submission UI, you can optionally edit that bitstream's metadata. The form displayed on that edit screen is built by the "bitstream-metadata" form defined in submission-forms.xml. You can modify that form to change the fields captured for a Bitstream. *However, the "dc.title" field is REQUIRED in order to store the name of the file.*

```
<form-definitions>
    <!-- Form used for entering in Bitstream/File metadata after uploading a file -->
    <form name="bitstream-metadata">
        ...
    </form>
</form-definitions>
```

Modifying access conditions (embargo, etc.) presented for Bitstreams

After uploading a file (bitstream) in the Submission UI, you can optionally edit that bitstream's access conditions. This allows you to embargo a bitstream, lease it, or limit it to Administrators only.

These access conditions are defined in a new Spring Bean configuration file `[dspace]/config/spring/api/access-conditions.xml`

- The "uploadConfigurationService" bean maps an existing "UploadConfiguration" bean (default is "uploadConfigurationDefault") to a specific step /section name used in item-submission.xml.

  ```
  <!-- This default configuration says the <step-definition id="upload"> defined in item-submission.xml
  uses "uploadConfigurationDefault" -->
  <bean id="uploadConfigurationService" class="org.dspace.submit.model.UploadConfigurationService">
      <property name="map">
          <map>
              <entry key="upload" value-ref="uploadConfigurationDefault" />
          </map>
      </property>
  </bean>
  ```

- One or more UploadConfiguration beans may exist, providing different options for different upload sections. An "UploadConfiguration" consists of several properties:

    - **name** (Required): The unique name of this upload configuration
    - **configurationService** (Required through 7.3): reference to the DSpace ConfigurationService (should always be "org.dspace.services.ConfigurationService"). Starting in 7.4 this is no longer required and should not be set.
    - **metadata** (Required): The metadata "form" to use for this upload configuration. The value specified here MUST correspond to a <form> defined in your submission-forms.xml. In the below example, the "bitstream-metadata" form is used by the "uploadConfigurationDefault" bean...meaning that form will be used to capture metadata about the uploaded bitstream.
    - **options** (Required, but can be empty): list of all "AccessConditionOption" beans to enable. This list will be shown to the user to let them select which access restrictions to place on each bitstream. NOTE: *To disable the ability to select bitstream access restrictions, comment out all <ref> tags to create an empty list of options.*
    - **maxSize:** Optionally, you can specify a maximum size of file accepted by this UploadConfiguration. If unspecified, default is to use the maximum file upload limits specified in Spring Boot (see "Basic Settings" above)
    - **required**: Optionally, you can specify if a file upload is required for this UploadConfiguration. If true, upload is required and users cannot complete a submission without uploading at least one file. If false, no upload is required to complete the submission. If unspecified, default is to use "webui.submit.upload.required" configuration in dspace.cfg/local.cfg, which defaults to "true" (file upload required).

      ```
      <bean id="uploadConfigurationDefault" class="org.dspace.submit.model.UploadConfiguration">
          <property name="name" value="upload"></property>
              <property name="configurationService" ref="org.dspace.services.ConfigurationService"/>
              <property name="metadata" value="bitstream-metadata" />
          <property name="options">
              <!-- This is the list of access options which will be displayed on the "bitstream-
      metadata" form -->
              <!-- If no <ref> tags appear in this list, then access restrictions will not be allowed on
      bitstreams -->
              <list>
                  <ref bean="openAccess"/>
                  <ref bean="lease"/>
                  <ref bean="embargoed" />
      ```

```
            <ref bean="administrator"/>
        </list>
    </property>
</bean>
```

- Any number of "AccessConditionOption" beans may be added for applying different types of access permissions to uploaded files (based on which one the user selects). These beans are easy to add/update, and just require the following
    - **id** (Required): Each defined bean MUST have a unique "id" and have "class=org.dspace.submit.model.AccessConditionOption".
    - **groupName**: Optionally, define a specific DSpace Group which this Access Condition relates to. This group will be saved to the ResourcePolicy when this access condition is applied.
    - **name**: Give a unique name for this Access Condition. This name is stored in the ResourcePolicy "name" when this access condition is applied.
    - **hasStartDate**: If the access condition is time-based, you can decide whether a start date is required. (true = required start date, false = disabled/not required). This start date will be saved to the ResourcePolicy when this access condition is applied.
    - **startDateLimit**: If the access condition is time-based, you can optionally set an start date limit (e.g. +36MONTHS). This field is used to set an upper limit to the start date based on the current date. In other words, a value of "+36MONTHS" means that users cannot set a start date which is more than 3 years from today. This setting's value uses Solr's Date Math Syntax, and is always based on today (NOW).
    - **hasEndDate**: If the access condition is time-based, you can enable/disable whether an end date is required. (true = required end date, false = disabled/not required). This end date will be saved to the ResourcePolicy when this access condition is applied.
    - **endDateLimit**: If the access condition is time-based, you can optionally set an end date limit (e.g. +6MONTHS). This field is used to set an upper limit to the start date based on the current date. In other words, a value of "+6MONTHS" means that users cannot set an end date which is more than 6 months from today. This setting's value use Solr's Date Math Syntax, and is always based on today (NOW).

```
<!-- Example access option named "embargo", which lets users specify a future date
    (not more than 3 years from now) when this file will be available to Anonymous users -->
<bean id="embargoed" class="org.dspace.submit.model.AccessConditionOption">
        <property name="groupName" value="Anonymous"/>
        <property name="name" value="embargo"/>
        <property name="hasStartDate" value="true"/>
        <property name="startDateLimit" value="+36MONTHS"/>
        <property name="hasEndDate" value="false"/>
</bean>
```

- NOTE: It's possible to **test** the Date math syntax via command-line to see what the value would look like starting from today:

```
./dspace dsrun org.dspace.util.DateMathParser +999YEARS
Applied +999YEARS to implicit current time:  Wed Jun 25 19:42:48 UTC 3023
```

- By default, DSpace comes with these out-of-the-box Access Conditions (which you can customize/change based on local requirements)
    - "administrator" - access restricts the bitstream to the Administrator group immediately (after submission completes)
    - "openAccess" - makes the bitstream immediately accessible to Anonymous group (after submission completes)
    - "embargoed" - embargoes the bitstream for a period of time (maximum of 3 years, as defined in startDateLimit default setting), after which it becomes anonymously accessible. See also Embargo for discussion of how embargoes work in DSpace.
    - "lease" - makes the bitstream anonymously accessible immediately (after submission completes), but that access *expires* after a period of time (maximum of 6 months, as defined in endDateLimit default setting). After that date it is no longer accessible (except to Administrators)

Configuring the Item Access Conditions step

Enabling the step

By default, the "Item Access Conditions" step is disabled. To enable it, simply update your `item-submission.xml` to include this tag in your <submission-process>:

```
<submission-process name="traditional">
   ...

   <!-- This step enables embargoes and other access restrictions at the Item level -->
   <step id="itemAccessConditions"/>
</submission-process>
```

After making this update, you will need to restart your backend (REST API) for the changes to take effect.

Modifying access conditions (embargo, etc.) presented for Items

The "Item Access Conditions" step uses a similar access condition configuration as the "Upload" step as described in the Modifying access conditions (embargo, etc.) presented for Bitstreams documentation above.

All available Item access conditions are defined in a new Spring Bean configuration file `[dspace]/config/spring/api/access-conditions.xml`

- One or more "AcccessConditionConfiguration" beans may exist, providing different options for different submission forms (only one should be in use in a form at a time). By default an "accessConditionConfigurationDefault" bean is defined. An "AccessConditionConfiguration" consists of several properties:
    - **name** (Required): The unique name of this configuration. It must match the "id" of the step defined in your `item-submission.xml`
    - **canChangeDiscoverable:** Whether this configuration allows users to change the discoverability of an Item. A "discoverable" item is one that is findable through all search/browse interfaces, provided that you have access to see that Item. A "non-discoverable" item is one that will never be findable through search/browse (except by Administrators)... instead a direct link is necessary to view the Item. See also DSpace Item State Definitions. When "canChangeDiscoverable" is "true", the user can modify discoverability in this submission section. When set to "false", the user cannot modify this setting and all submitted Items will be "discoverable".
    - **options** (Required): list of all "AccessConditionOption" beans to enable for this Item access conditions step. This list will be shown to the user to let them select which access restrictions to place on this Item.
- This step uses the same "AccessConditionOption" beans as the "Upload" step, as described in the Modifying access conditions (embargo, etc.) presented for Bitstreams documentation above. You can choose to enable the same options for both Items and Bitstreams, or provide different options for each.
- By default, DSpace comes with these out-of-the-box Access Conditions (which you can customize/change based on local requirements)
    - "administrator" - access restricts the bitstream to the Administrator group immediately (after submission completes)
    - "openAccess" - makes the bitstream immediately accessible to Anonymous group (after submission completes)
    - "embargoed" - embargoes the bitstream for a period of time (maximum of 3 years, as defined in startDateLimit default setting), after which it becomes anonymously accessible. See also Embargo for discussion of how embargoes work in DSpace.
    - "lease" - makes the bitstream anonymously accessible immediately (after submission completes), but that access *expires* after a period of time (maximum of 6 months, as defined in endDateLimit default setting). After that date it is no longer accessible (except to Administrators)

Examples of selecting Item and Bitstream access conditions

What happens when a User selects different access conditions for an Item (via the "Item Access Conditions" step) and its files (via the "Upload" step)? Generally speaking, both

Generally speaking, both access restrictions will be applied. Here's some examples:

- If a user selects "openAccess" in the "Item Access Conditions" step AND "embargo" in the "Upload" step for one Bitstream
    - Then, the Item's metadata will be publicly visible, but that single Bitstream will be embargoed.
- If a user selects "openAccess" in the "Item Access Conditions" step AND "administrator" in the "Upload" step for one Bitstream
    - Then, the Item's metadata will be publicly visible, but that single Bitstream will only be visible to Administrators
- If a user selects "administrator" in the "Item Access Conditions" step AND nothing in the "Upload" step.
    - Then, the Item's metadata and all Bitstreams will only be accessible to administrators.
- If a user selects "embargo" in the "Item Access Conditions" step AND nothing in the "Upload" step.
    - Then, the Item's metadata and all Bitstreams will be embargoed. Nothing will be visible in the system until the embargo data passes.
- If a user selects "embargo" in the "Item Access Conditions" step AND "openAccess" in the "Upload" step for one Bitstream.
    - Then, the Item's metadata will be embargoed (making it impossible to find the Item unless you are an Administrator). HOWEVER, the bitstream will be publicly accessible immediately (but only via a direct link, as it won't be searchable in the system until the embargo date passes).
- (To test other scenarios, submit a test Item with those permissions applied. Then, edit that Item, visit the "Status" tab, and click "Authorizations" to see what access restrictions were applied to the Item and its bitstreams.)

Configuring the Sherpa Romeo step

Enabling the Sherpa Romeo step

By default, the "Sherpa RoMEO Policy" step is disabled. To enable it, simply update your `item-submission.xml` to include this tag in your <submission-process>:

```
<submission-process name="traditional">
   ...
   <!-- This step shows when appropriate publisher policies retrieved from SHERPA/RoMEO -->
   <step id="sherpaPolicies"/>
</submission-process>
```

you must also obtain your `sherpa.romeo.apikey` registering you client application here https://v2.sherpa.ac.uk/api/ and put them in the `local.cfg`

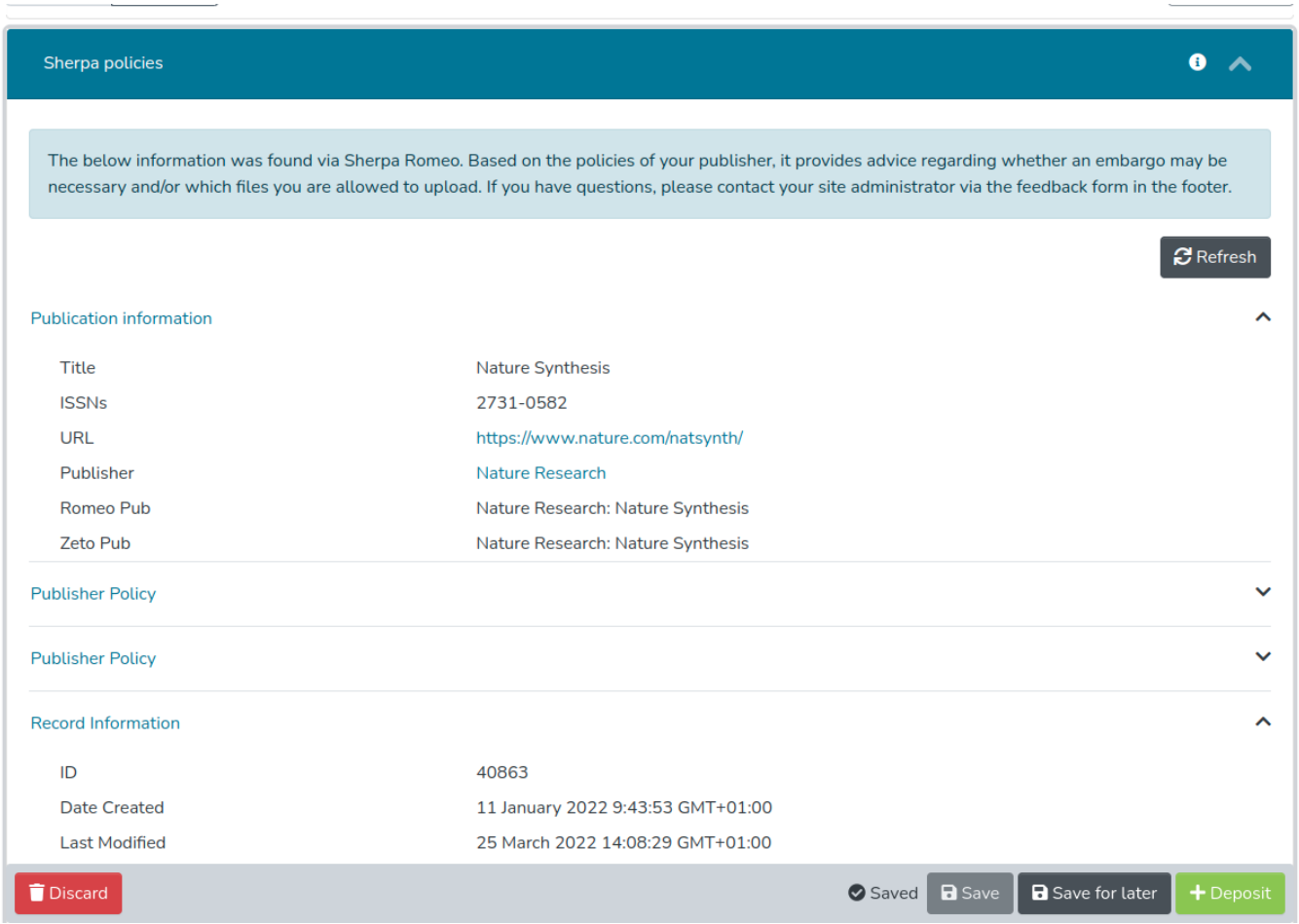
```
sherpa.romeo.apikey = <YOUR-API-KEY>
```

The step needs to extract the ISSN of the Journal where the publication has been submitted/published to query the Sherpa/RoMEO database in order to visualize the publisher policies, this is done by an implementation of the `org.dspace.app.sherpa.submit.ISSNItemExtractor` interface configured in the `org.dspace.app.sherpa.submit.SHERPASubmitConfigurationService`

The configuration is provided by Spring `config/spring/api/sherpa.xml`

```
<bean class="org.dspace.app.sherpa.submit.SHERPASubmitConfigurationService"
        id="org.dspace.app.sherpa.submit.SHERPASubmitConfigurationService">
        <property name="issnItemExtractors">
                <list>
                        <bean class="org.dspace.app.sherpa.submit.MetadataValueISSNExtractor">
                                <property name="metadataList">
                                        <list>
                                                <value>dc.identifier.issn</value>
                                        </list>
                                </property>
                        </bean>
                        <!-- Uncomment this bean if you have SHERPARoMEOJournalTitle enabled
                        <bean class="org.dspace.app.sherpa.submit.MetadataAuthorityISSNExtractor">
                                <property name="metadataList">
                                        <list>
                                                <value>dc.title.alternative</value>
                                        </list>
                                </property>
                        </bean>  -->
                </list>
        </property>
</bean>
```

out-of-box implementations able to extract the ISSN from the metadata value or authority are provided.



Configuring the "Identifiers" step

By default, the "Identifiers" step is disabled. To enable it, update your `item-submission.xml` to include this tag in your <submission-process>:

```
<submission-process name="traditional">

   ...

   <!-- This step shows identifiers already registered for this in-progress item

   <step id="identifiers"/>

   ...

</submission-process>
```

It is recommended to display this step above most others so that the submitter can clearly see any identifiers that will be created while completing their submission.

You must also enable registration of identifiers for workspace and workflow items in `dspace/config/modules/identifiers.cfg` or `local.cfg` (thi s is disabled by default):

```
identifiers.submission.register = true
```

While editing this configuration, pay attention to the filter configuration - logical item filters can be referenced here to apply some conditions as to whether an item qualifies for a DOI or not (eg. based on metadata entered, the type of work, or so on).

Any identifiers registered for the current submission or workflow item will be displayed in a read-only section. If no identifiers are registered, a placeholder “no identifiers” message will be displayed.

If DOI registration is configured for logical item filtering, the DOI will be minted (in a 'pending' state) or deleted as appropriate whenever the in-progress item is saved, depending on whether it passes the filter test.

See `dspace/config/modules/identifiers.cfg`

Creating new Submission Steps Programmatically.

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.*

In most scenarios, this is NOT necessary, as it's much easier to configure a custom Submission Step using DescribeStep or similar.

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

1. 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 the UI
2. 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 to the UI*
    - For more information on *<step-definition> tags* within the `item-submission.xml`, see the section above on Defining Steps (*<step>*) within the `item-submission.xml`.
3. For the UI, you will need to..
    - Add a new section type to SectionsType enum matching to the type of step you are creating
    - Create a new Component for this new SectionsType, annotated with "@renderSectionFor()".... see existing section components under src/app/submission/sections for examples.
    - *(Other steps may be necessary... this process has not been fully documented at this time.)*