

Submission User Interface

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

- 1 [Default Submission Process](#)
 - 1.1 [Optional Steps](#)
- 2 [Understanding the Submission Configuration File](#)
 - 2.1 [The Structure of item-submission.xml](#)
 - 2.2 [Defining Steps \(<step>\) within the item-submission.xml](#)
 - 2.2.1 [Where to place your <step> definitions](#)
 - 2.2.2 [The ordering of <step> definitions matters!](#)
 - 2.2.3 [Structure of the <step> Definition](#)
- 3 [Reordering/Removing/Adding Submission Steps](#)
- 4 [Assigning a custom Submission Process to a Collection](#)
 - 4.1 [Getting A Collection's Handle](#)
- 5 [Custom Metadata-entry Pages for Submission](#)
 - 5.1 [Introduction](#)
 - 5.2 [Describing Custom Metadata Forms](#)
 - 5.3 [The Structure of input-forms.xml](#)
 - 5.3.1 [Adding a Collection Map](#)
 - 5.3.1.1 [Getting A Collection's Handle](#)
 - 5.3.2 [Adding a Form Set](#)
 - 5.3.2.1 [Forms and Pages](#)
 - 5.3.2.2 [Composition of a Field](#)
 - 5.3.2.3 [Item type Based Metadata Collection](#)
 - 5.3.2.4 [Automatically Omitted Fields](#)
 - 5.3.3 [Configuring Controlled Vocabularies](#)
 - 5.3.4 [Adding Value-Pairs](#)
 - 5.3.4.1 [Example](#)
 - 5.4 [Deploying Your Custom Forms](#)
- 6 [Configuring the File Upload step](#)
- 7 [Creating new Submission Steps](#)
 - 7.1 [Creating a Non-Interactive Step](#)
- 8 [Configuring StartSubmissionLookupStep](#)
 - 8.1 [About the Biblio-Transformation-Engine](#)
 - 8.2 [StartSubmissionLookupStep in action!](#)
 - 8.3 [SubmissionLookup service configuration file](#)

Default Submission Process

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

1. "Select Collection" step: If not already selected, the user must select a collection to deposit the Item into.
2. "Describe" step: This is where the user may enter descriptive metadata about the Item. This step may consist of one or more pages of metadata entry. By default, there are two pages of metadata-entry. For information on modifying the metadata entry pages, please see [Custom Metadata-entry Pages for Submission](#) section below.
3. "Upload" step: This is where the user may upload one or more files to associate with the Item. For more information on file upload, also see [Configuring the File Upload step](#) below.
4. "Review" step: This is where the user may review all previous information entered, and correct anything as needed.
5. "License" step: 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 Admin UI.
6. "Complete" step: The deposit is now completed. The 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](#).

DSpace 4.0 has removed the "Initial Questions" step by default

Prior to DSpace 4.0, the "Initial Questions" step preceded all "Describe" steps. However, it was removed by default in DSpace 4.0.

You may still choose to re-enable the "Initial Questions" step, as needed. However, please note the warning below about the auto-assigning of Dates in the "Initial Questions" step.


Optional Steps

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

- "Access" step: 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" step: 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.
- "Start Submission Lookup" step: This step allows the user to search or load metadata from an external service (arXiv online, bibtex file, etc.) and prefill the submission form. For more information on enabling and using it, please see the section on [Configuring StartSubmissionLookupStep](#) below.
- "Initial Questions" step: This step asks users a simple set of "initial questions" which help to determine which metadata fields are displayed in the "Describe" step (see above). These initial questions include:
 - *Multiple Titles*: The item has more than one title, e.g. a translated title (If selected, then users will be asked for an alternative title in the Describe step)
 - *Published Before*: The item has been published or publicly distributed before (If selected, then users will be asked for a publication date and publisher in the Describe step).

Initial Questions will auto-assign a publication date when "Published Before" is unselected

 Please note, if you enable Initial Questions, and your users do NOT select "Published Before" option, then DSpace will auto-assign a publication date (dc.date.issued) to that particular Item.

It may be entirely accurate for some types of content (e.g. for gray literature or even theses/dissertations) to auto-assign this publication date. As such, you may wish to still enable "Initial Questions" if your repository is mainly for previously unpublished content. You may also choose to only enable it for specific Collections – see [Assigning a custom Submission Process to a Collection](#) section below.

However, if the Item actually was published in some other location, this will result in an incorrect publication date being reported by DSpace. This tendency for an incorrect publication date has been reported by Google Scholar to DSpace developers (see: [DS-1481](#)), which is why the "Initial Questions" are now disabled by default (see [DS-1655](#)).

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 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</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 -> 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>
```

-
2. Within a specific <submission-process> definition
 - The above step definition could then be referenced from within a <submission-process> as simply <step id="custom-step"/>
 - 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-class>
    <jspui-binding>org.dspace.app.webui.submit.step.JSPLicenseStep</jspui-binding>
    <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.LicenseStep</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</processing-class>
    <jspui-binding>org.dspace.app.webui.submit.step.JSPInitialQuestionsStep</jspui-binding>
    <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.InitialQuestionsStep</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) and 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-class>
  <jspui-binding>org.dspace.app.webui.submit.step.JSPDescribeStep</jspui-binding>
  <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.DescribeStep</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. *This 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/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 (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!

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 (i.e. everything between and including the opening `<step>` and closing `</step>` tags).

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-process>
</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>
    ...
  </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>

```

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 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.

Item type Based Metadata Collection

This feature is available for use with the XMLUI since DSpace 3.0 and with JSPUI since 3.1. 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. 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>
```

Automatically Omitted 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 omitted, *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 checkboxes:

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 omitted 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.

Configuring Controlled Vocabularies

DSpace now 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 organised 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 correspondent DC metadata fields. Edit the file `[dspace]/config/input-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.

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 *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).

Configuring StartSubmissionLookupStep

StartSubmissionLookupStep is a new submission step, available since DSpace 4.0 contributed by [CINECA](#), that extends the basic SelectCollectionStep allowing the user to search or load metadata from an external service (arxiv online, bibtex file, etc.) and prefill the submission form. Thanks to the [EKT](#) work s it is underpinned by the Biblio Transformation Engine (<https://github.com/EKT/Biblio-Transformation-Engine>) framework.

To enable the StartSubmissionLookupStep you only need to change the configuration of the `id="collection"` step to match the following

item-submission.xml excerpt

```
<step id="collection">
  <heading></heading> <!--can specify heading, if you want it to appear in Progress Bar-->
  <processing-class>org.dspace.submit.step.StartSubmissionLookupStep</processing-class>
  <jspui-binding>org.dspace.app.webui.submit.step.JSPStartSubmissionLookupStep</jspui-binding>
  <xmlui-binding>org.dspace.app.xmlui.aspect.submission.submit.SelectCollectionStep</xmlui-binding>
  <workflow-editable>false</workflow-editable>
</step>
```

UI compatibility



The new step is available **only for JSP UI**. Nonetheless, if you run both UIs and want the JSP UI benefit of the new step you can configure it as processing class also for XML as it degrades gracefully to the standard SelectCollectionStep logic

About the Biblio-Transformation-Engine

The BTE is a Java framework developed by the Hellenic National Documentation Centre ([EKT](#)) and consists of programmatic APIs for filtering and modifying records that are retrieved from various types of data sources (eg. databases, files, legacy data sources) as well as for outputting them in appropriate standards formats (eg. database files, txt, xml, Excel). The framework includes independent abstract modules that are executed separately, offering in many cases alternative choices to the user depending of the input data set, the transformation workflow that needs to be executed and the output format that needs to be generated.

The basic idea behind the BTE is a standard workflow that consists of three steps, a data loading step, a processing step (record filtering and modification) and an output generation. A data loader provides the system with a set of Records, the processing step is responsible for filtering or modifying these records and the output generator outputs them in the appropriate format.

The standard BTE version offers several predefined Data Loaders as well as Output Generators for basic bibliographic formats. However, Spring Dependency Injection can be utilized to load custom data loaders, filters, modifiers and output generators.

StartSubmissionLookupStep in action!

When StartSubmissionLookupStep is enabled, the user comes up with the following screen when a new submission is initiated:

The screenshot shows a web interface titled "New submission: get data from bibliographic external service". At the top, a yellow banner states "No collection selected". Below this is a search form with two tabs: "Search Form" and "Results". The "Search Form" tab is active, showing a section titled "Search for identifier". This section contains instructions: "Fill in publication identifiers (DOI is preferable) and then press 'Search'. A list of all matching publications will be shown to you to select in order to proceed with the submission process." There are three input fields with corresponding logos: "PubMed ID:" with a PubMed logo, "DOI (Digital Object Identifier) :" with PubMed, CrossRef, and arXiv logos, and "arXiv ID:" with an arXiv logo. Below the input fields are two buttons: "Upload a file" and "Default mode Submission". At the bottom left, there is a link "Go to DSpace Home".

There are four accordion tabs (default configuration hides the third tab):

1) Search for identifier: In this tab, the user can search for an identifier in the supported online services (currently, [arXiv](#), [PubMed](#), [CrossRef](#) and [CiNii](#) are supported). The publication results are presented in the tab "Results" in which the user can select the publication to proceed with. This means that a new submission form will be initiated with the form fields prefilled with metadata from the selected publication.

Currently, there are four identifiers that are supported (DOI, PubMed ID, arXiv ID and NAID (CiNii ID)). But these can be extended - refer to the following paragraph regarding the SubmissionLookup service configuration file.

User can fill in any of the four identifiers. DOI is preferable. Keep in mind that the service can integrate results for the same publication from the three different providers so filling any of the four identifiers will pretty much do the work. If identifiers for different publications are provided, the service will return a list of publications which will be shown to user to select. The selected publication will make it to the submission form in which some fields will be pre-filled with the publication metadata. The mapping from the input metadata (from arXiv or Pubmed or CrossRef or CiNii) to the DSpace metadata schema (and thus, the submission form) is configured in the Spring XML file that is discussed later on - you can see a table at the very end of this chapter.

Through the same file, a user can also extend the providers that the SubmissionLookup service can search publication from.

2) Upload a file: In this tab, the user can upload a file, select the type (bibtex, csv, etc.), see the publications in the "Results" tab and then either select one to proceed with the submission or make all of them "Workspace Items" that can be found in the "Unfinished Submissions" section in the "My DSpace" page.

New submission: get data from bibliographic external service

No collection selected

Search Form Results

Search for identifier

Upload a file

Select a file to upload and its type from the drop-down menu. If "Preview Mode" is enabled, the list of the publications in the file will be shown to you to select the one for submission. If it is disabled, all publications will be imported in your MyDSpace page as "Unfinished Submissions" while the first one will go through the submission process.

Select data type: Select...

File: Επιλογή αρχείου Δεν έχει επιλεγεί κανένα αρχείο

☐ Preview mode

Collection: Test Collection

Search Exit

Default mode Submission

Go to

The "preview mode" in the figure above has the following functionality:

"ON": The list of the publications in the uploaded file will be shown to the user to select the one for the submission. The selected publication's metadata will pre-fill the submission form's fields according to configuration in the Spring XML configuration file.

"OFF": All the publications of the uploaded file will be imported in the user's MyDSpace page as "Unfinished Submissions" while the first one will go through the submission process.

(Regarding the pubmed, crossref and arxiv file upload, you can find the attached file named "sample-files.zip" that contains samples of these three file types)

3) Free search: In this tab, the user can freely search for Title, Author and Year in the four supported providers (PubMed, CrossRef, Arxiv and CiNii). By default, the four providers are configured to be disabled for free search but you can enable it via the configuration file. Thus, initially this accordion tab is not shown to the user except for a data loader is declared as a "search provider" - refer to the following paragraphs.

New submission: get data from bibliographic external service

No collection selected

Search Form Results

Free search

PubMed

Insert base info about publication: either title or author/year is required.
If you know any unique identifier about publication like DOI, Pubmed, or arXiv you can switch on the identifier search mode.

Title:

Year:

Authors/Publishers :

Search for identifier

Upload a file

Default mode Submission

The process is the same as in the previous cases. A result of publications is presented to the user to select the one to proceed with the submission.

4) Default mode submission: In this tab, the user can proceed to the default manual submission. The SubmissionLookup service will not run and the submission form will be empty for the user to start filling it.

SubmissionLookup service configuration file

The StartSubmissionLookupStep relies on business logic provided by the SubmissionLookup service that can be heavily extended and customized and is built on top of the BTE.

The basic idea behind BTE is that the system holds the metadata in an internal format using a specific key for each metadata field. DataLoaders load the record using the aforementioned keys, while the output generator needs to map these keys to DSpace metadata fields.

The BTE configuration file is located in path: [dspace]/config/spring/api/bte.xml and it's a Spring XML configuration file that consists of Java beans. (If these terms are unknown to you, please refer to Spring Dependency Injection web site for more information.)

The service is broken down into two phases. In the first phase, the imported publications' metadata are converted to an intermediate format while in the second phase, the intermediate format is converted to DSpace metadata schema

Explanation of beans:

```
<bean id="org.dspace.submit.lookup.SubmissionLookupService" />
```

This is the top level bean that describes the service of the SubmissionLookup. It accepts three properties:

a) phase1TransformationEngine: the phase 1 BTE transformation engine.

b) phase2TransformationEngine: the phase 2 BTE transformation engine

c) detailFields: A list of the keys that the user wants to display in the detailed form of a publication. That is, when the results are shown, user can see the details of each one. In the detailed form, some fields appear. These fields are configured by this property. Refer to the table at the very end of this chapter to see the available values. This property is disabled by default while the list that is shown commented out is the default list for the detailed form.

```
<bean id="phase1TransformationEngine" />
```

The transformation engine for the first phase of the service (from external service to intermediate format)

It accepts three properties:

a) dataLoader : The data loader that will be used for the loading of the data

b) workflow : This property refers to the bean that describes the processing steps of the BTE. If no processing steps are listed there all records loaded by the data loader will pass to the output generator, unfiltered and unmodified.

c) outputGenerator : The output generator to be used.

Normally, you do not need to touch any of these three properties. You can edit the reference beans instead.

```
<bean id="multipleDataLoader" />
```

This bean declares the data loader to be used to load publications from. It has one property "dataloadersMap", a map that declares key-value pairs, that is a unique key and the corresponding data loader to be used. Here is the point where a new data loader can be added, in case the ones that are already supported do not meet your needs.

A new data loader class must be created based on the following:

a) Either extend the abstract class **gr.ekt.bte.core.dataloader.FileDataLoader**

in such a case, your data loader key will appear in the drop down menu of data types in the "*Upload a file*" accordion tab

b) Or, extend the abstract class **or g.dspace.submit.lookup.SubmissionLookupDataLoader**

in such a case, your data loader key will appear as a provider in the "*Search for identifier*" accordion tab

```
<bean id="bibTeXDataLoader" />
<bean id="csvDataLoader" />
<bean id="tsvDataLoader" />
<bean id="risDataLoader" />
<bean id="endnoteDataLoader" />
<bean id="pubmedFileDataLoader" />
<bean id="arXivFileDataLoader" />
<bean id="crossRefFileDataLoader" />
<bean id="ciniiFileDataLoader" />
<bean id="pubmedOnlineDataLoader" />
<bean id="arXivOnlineDataLoader" />
<bean id="crossRefOnlineDataLoader" />
<bean id="ciniiOnlineDataLoader" />
```

These beans are the actual data loaders that are used by the service. They are either "FileDataLoaders" or "SubmissionLookupDataLoaders" as mentioned previously.

The data loaders have the following properties:

a) fieldMap : it is a map that specifies the mapping between the keys that hold the metadata in the input format and the ones that we want to have internal in the BTE. At the end of this article there is a table that summarises the fields that are used from the three online services (pubmed, arXiv and crossRef) - which are the ones that the submission lookup step is capable of reading from the online services - and the keys used internally in the BTE.

Some loaders have more properties:

CSV and **TSV** (which is actually a CSV loader if you look carefully the class value of the bean) loaders have some more properties:

a) skipLines: A number that specifies the first line of the file that loader will start reading data. For example, if you have a csv file that the first row contains the column names, and the second row is empty, the the value of this property must be 2 so as the loader starts reading from row 2 (starting from 0 row). The default value for this property is 0.

b) separator: A value to specify the separator between the values in the same row in order to make the columns. For example, in a TSV data loader this value is "\u0009" which is the "Tab" character. The default value is "," and that is why the CSV data loader doesn't need to specify this property.

c) quoteChar: This property specifies the quote character used in the CSV file. The default value is the double quote character (").

pubmedOnlineDataLoader, **crossRefOnlineDataLoader**, **arXivOnlineDataLoader** and **ciniiOnlineDataLoader** also support another property:

a) searchProvider: if is set to true, the dataloader supports free search by title, author or year. If at least one of these data loaders is declared as a search provider, the accordion tab "Free search" is appeared. Otherwise, it stays hidden.

crossRefOnlineDataLoader and **ciniiOnlineDataLoader** also have two more properties:

a) apiKey/appld respectively: Both these services need to acquire (for free) an API key in order to access their online services. For CrossRef, visit: <http://www.crossref.org/requestaccount/> and for CiNii visit: <https://portaltools.nii.ac.jp/developer/en/>

b) maxResults: the maximum results that these services will reply with to your search. By default, this property is commented out while the default value is 10 for both services.

(Regarding the file dataloaders, you can find the attached file named "sample-files.zip" that contains samples of all the file types that the corresponding data loaders can handle)

```
<bean id="phase1LinearWorkflow" />
```

This bean specifies the processing steps to be applied to the records metadata before they proceed to the output generator of the transformation engine. Currently, three steps are supported, but you can add yours as well.

```
<bean id="mapConverter_arxivSubject" />
<bean id="mapConverter_pubstatusPubmed" />
<bean id="removeLastDot" />
```

These beans are the processing steps that are supported by the 1st phase of transformation engine. The two first map an incoming value to another one specified in a properties file. The last one is responsible to remove the last dot from the incoming value.

All of them have the property "**fieldKeys**" which is a list of keys where the step will be applied.

In the case you need to create your own filters and modifiers follow the instructions below:

To create a new filter, you need to extend the following BTE abstract class:

```
gr.ekt.bte.core.AbstractFilter
```

You will need to implement the following method:


```
public abstract boolean isIncluded ( Record record )
```

Return false if the specified record needs to be filtered, otherwise return true.

To create a new modifier, you need to extend the following BTE abstract class:

```
gr.ekt.bte.core.AbstractModifier
```

You will need to implement the following method:

```
public abstract Record modify ( Record record )
```

within you can make any changes you like in the record. You can use the Record methods to get the values for a specific key and load new ones (For the later, you need to make the Record mutable)

After you create your own filters or modifiers you need to add them in the Spring XML configuration file as in the following example:

```
<bean id="customfilter" class="org.mypackage.MyFilter" />

<bean id="phase1LinearWorkflow" class="gr.ekt.bte.core.LinearWorkflow">
  <property name="process">
    <list>
      ... <old filters and modifiers>...
      <ref bean="customfilter" />
    </list>
  </property>
</bean>
```

```
<bean id="phase2TransformationEngine" />
```

The transformation engine for the second phase of the service (from the intermediate format to DSpace metadata schema)

Normally, you do not need to touch any of these three properties. You can edit the reference beans instead.

```
<bean id="phase2linearWorkflow" />
```

This bean specifies the processing steps to be applied to the records metadata before they proceed to the output generator of the transformation engine. Currently, two steps are supported, but you can add yours as well.

```
<bean id="fieldMergeModifier" />
<bean id="valueConcatenationModifier" />
<bean id="languageCodeModifier" />
```

These beans are the processing steps that are supported by the 2nd phase of transformation engine. The first merges the values of multiple keys to a new key. The second one concatenates the values of a specific key to a unique value. The third one translated the three-letters language code to two-letters one (ie: eng to en)

```
<bean id="org.dspace.submit.lookup.DSpaceWorkspaceItemOutputGenerator" />
```

This bean declares the output generator to be used which is, in this case, a DSpaceWorkspaceItem generator. It accepts two properties:

a) outputMap: A map from the intermediate keys to the DSpace metadata schema fields. The table below displays the default output mapping. As you can see, some fields, while they are read from the input source, are not output in DSpace since there are no default metadata schema fields to host them. However, if you create the corresponding metadata field registry, you can come back in this configuration to add a map between the input field key and the DSpace metadata field.

b) extraMetadataToKeep: A list of DSpace metadata schema fields to keep in the output

The following table presents the available keys from the online services, the keys that BTE uses in phase1 and the final output map to DSpace metadata fields.

Arxiv	PubMed	CrossRef	CiNii	BTE Key (phase 1)	Extra Keys created by BTE (phase 2)	DSpace Metadata Field	Appears in Detail Form
title	articleTitle	articleTitle	title	title		dc.title	yes
published	pubDate	year	issued	issued		dc.date.issued	yes
id				url			
summary	abstractText		description	abstract		dc.description.abstract	yes
comment				note			
pdfUrl				fulltextUrl			
doi	doi	doi		doi		dc.identifier	yes
journalRef	journalTitle	journalTitle	journal	journal		dc.source	yes
author	author	authors	authors	authors		dc.contributor.author	yes
authorWithAffiliation				authorsWithAffiliation			
primaryCategory				arxivCategory		dc.subject	yes
category				arxivCategory		dc.subject	
	pubmedID			pubmedID			
	publicationStatus			publicationStatus			
	pubModel						
	printISSN	printISSN	issn	jissn		dc.identifier.issn	yes
	electronicISSN	electronicISSN		jeissn			
	journalVolume	volume	volume	volume			
	journalIssue	issue	issue	issue			
	language		language	language		dc.language.iso	yes
	publicationType	doiType		subtype		dc.type	yes
	primaryKeyword		subjects	keywords	allkeywords	dc.subject	yes
	secondaryKeyword			keywords	allkeywords	dc.subject	yes
	primaryMeshHeading			mesh	allkeywords	dc.subject	yes
	secondaryMeshHeading			mesh	allkeywords	dc.subject	yes
	startPage	firstPage	spage	firstpage			
	endPage	lastPage	epage	lastpage			
		printISBN		pisbn		dc.identifier.isbn	yes
		electronicISBN		eisbn			
		editionNumber		editionnumber			
		seriesTitle		seriestitle			
		volumeTitle		volumetitle			
		publicationType					
		editors		editors		dc.contributor.editor	yes
		translators		translators		dc.contributor.other	yes
		chairs		chairs		dc.contributor.other	yes
			naid	naid			
			ncid	ncid			
			publisher	publisher		dc.publisher	yes

I can see more beans in the configuration file that are not explained above. Why is this?



The configuration file hosts options for two services. [BatchImport service](#) and SubmissionLookup service. Thus, some beans that are not used in the first service, are not mentioned in this documentation. However, since both services are based on the BTE, some beans are used by both services.