Configuring Normalization Rules for Primo VE

Normalization rules define transformations that are applied to external source records (DC or generic XML) to create the Primo VE expanded Dublin Core records for discovery. You can create your own normalization rules and apply them to externally loaded records.

Note
For information on how to create normalization rules for MARC21 records from external sources, see Working with Normalization Rules.

Using the Expanded DC Schema

The Primo VE Dublin Core expanded schema includes additional discovery fields that you can populate as needed using normalization rules. The following fields are supported by the expanded DC schema:

- **discovery.resourceType** – These fields hold the resource types that display in Primo, which may be different from the values in the dc:type and dcterms:type fields that are loaded from the external records. For more information, see Mapping Resource Types from External Data Sources.

- **discovery.local1 thru discovery.local50** – These fields hold additional information that you can use to display additional information from the source and also to use as a search index. For more information, see Mapping to Local Discovery Fields.

Mapping Resource Types from External Data Sources

Because the values for Dublin Core resource types can be anything and may not match the supported codes used in Primo VE, it may be necessary to map the resource type from the source record's dc:type or dcterms:type field to either a supported resource type code in Primo VE (see Supported Resource Types in Primo VE) or a local resource type defined in Primo VE (see Configuring Local Resource Types for Primo VE).

You can use any of the following methods to map the resource types to Primo VE:

- Create normalization rules that replace the value of the relevant dc:type or dcterms:type field.

- Create normalization rules that map a resource type value to the discovery:resourceType field and leave the original DC field as it is.

- Add resource type mappings to the **Dublin Core Type to Discovery Type Mapping** mapping table. For more information, see Using the Mapping Table to Map Resource Types.

Primo VE determines a record's resource type based on the existence of the following fields in order:

1. discovery.resourceType field
2. dcterms:type field

3. dc:type field

If processing continues to items 2 and 3, the system compares the value of the item with the mappings in the **Dublin Core Type to Discovery Type Mapping** mapping table. If a match is found, the matching value in the mapping table is used and stored in the discovery:resourceType field. If no match is found and the mapping table has a default value, the default value will be used. Otherwise, the system will use the value of the DC field as is.

**Examples**

For more information about creating normalization rules, see [Creating Normalization Rules](#).

The following example maps the Proceedings resource type in dc:type to a supported code in Primo VE.

```
rule "from dc:type A to discovery:type B"
when
"dc"."type" equals "Proceedings"
then
set "conference_proceedings" in "discovery"."resourceType"
end
```

The following example maps the LibGuides resource type in dcterms:type to a new local resource type that you must define in Primo VE.

**Note**

If the local resource type had been defined as **LibGuides** in Primo VE, the following normalization rule would not be necessary, and the system would just use the resource type that was mapped to dcterms:type in Primo VE.

```
rule "from dcterms:type A to discovery:type B"
when
"dcterms"."type" equals "LibGuides"
then
set "library_guides" in "discovery"."resourceType"
end
```
**Supported Resource Types in Primo VE**

The table below lists the resource type codes that are supported by Primo VE. You will need to map your source’s values to one of the supported codes or a local resource type (if needed). For more information about creating local resource types, see [Configuring Local Resource Types for Primo VE](#).

<table>
<thead>
<tr>
<th>Resource Type Code</th>
<th>Display Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>databases</td>
<td>Database</td>
</tr>
<tr>
<td>audios</td>
<td>Audio</td>
</tr>
<tr>
<td>newspapers</td>
<td>Newspaper</td>
</tr>
<tr>
<td>manuscripts</td>
<td>Manuscript</td>
</tr>
<tr>
<td>conference_proceedings</td>
<td>Conference Proceeding</td>
</tr>
<tr>
<td>dissertations</td>
<td>Dissertation</td>
</tr>
<tr>
<td>kits</td>
<td>Kit</td>
</tr>
<tr>
<td>other</td>
<td>Other</td>
</tr>
<tr>
<td>archival_materials</td>
<td>Archival Material</td>
</tr>
<tr>
<td>realia</td>
<td>Realia</td>
</tr>
<tr>
<td>books</td>
<td>Book</td>
</tr>
<tr>
<td>book_chapters</td>
<td>Book Chapter</td>
</tr>
<tr>
<td>collections</td>
<td>Collection</td>
</tr>
<tr>
<td>Resource Type Code</td>
<td>Display Name</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>legal_documents</td>
<td>Legal Document</td>
</tr>
<tr>
<td>patents</td>
<td>Patent</td>
</tr>
<tr>
<td>reference_entries</td>
<td>Reference Entry</td>
</tr>
<tr>
<td>research_datasets</td>
<td>Research Dataset</td>
</tr>
<tr>
<td>reviews</td>
<td>Review</td>
</tr>
<tr>
<td>statistical_data_sets</td>
<td>Statistical Data Set</td>
</tr>
<tr>
<td>technical_reports</td>
<td>Technical Report</td>
</tr>
<tr>
<td>journals</td>
<td>Journal</td>
</tr>
<tr>
<td>newspaper_articles</td>
<td>Newspaper Article</td>
</tr>
<tr>
<td>articles</td>
<td>Article</td>
</tr>
<tr>
<td>text_resources</td>
<td>Text Resource</td>
</tr>
<tr>
<td>government_documents</td>
<td>Government Document</td>
</tr>
<tr>
<td>images</td>
<td>Image</td>
</tr>
<tr>
<td>maps</td>
<td>Map</td>
</tr>
<tr>
<td>videos</td>
<td>Video</td>
</tr>
<tr>
<td>scores</td>
<td>Score</td>
</tr>
</tbody>
</table>
Mapping to Local Discovery Fields

The Dublin Core expanded schema for Primo VE allows you to map information to the following fields, which are populated during the loading of records: `discovery.local1 - discovery.local50`. After the records have been loaded, Primo VE can use the local fields to display additional information in a record’s brief and full displays and also to use them as search fields. Local fields are particularly useful to map non-standard DC fields that are not loaded by default during the import to Primo VE.

When using a local discovery field, you must also configure its similarly named local display field. For example, if you have created a normalization rule for `discovery.local2`, you must also define the local display field `Local_field_02`, making sure that you select the Use the parallel Local Field 01/50 from the Dublin Core record check box. For more information, see Configuring Local Display and Search Fields for Primo VE.

Examples

For more information about creating normalization rules, see Creating Normalization Rules.

The following example maps the value in dc:relation to local field 1 in Primo VE.

```
rule "move relation to local field 1"

when

exist "dc"."relation"

then

move "dc"."relation" to "discovery"."local1"

end
```

The following example sets the value of local field 2 to Internal Research for every record processed.

```
rule "set local field 2 for internal research"

when

True

then

set "Internal Research" in "discovery"."local2"
```
Creating Normalization Rules

The MD Editor allows you to create normalization rules, which are used to customize the transformation of external records (either Dublin Core or generic XML) into Primo VE’s extended Dublin Core format for discovery. Once the rules are created, you can add them as needed to normalization rule processes, which can then be assigned to import profiles. For more information, see Creating a Normalization Rule Process.

Note

For information on how to create normalization rules for MARC21 external data sources, see Working with Normalization Rules.

To add a normalization rule file:

1. Open the MD Editor (Configuration Menu > Discovery > Loading External Data Sources > Normalization Rules for External Data Sources).

   ![Open Rules Tab]

2. Select one of the following folders the folder under the Rules tab to view the relevant normalization rule files: Normalization Rules (DC) or Normalization Rules (XML). You can use them as the basis for your new rules. It is also a good practice to see whether a rule has already been created.
3. Create a new rule by selecting File > New > Normalization Rules (DC) or Normalization Rules (XML), which depends on the format of the Discovery import files that you are receiving from the external source.

4. Specify the following properties and then click Save:

   ◦ **Name** - Specify an identifiable name for the rule file, which may contain related types of rules.

   ◦ **Description** - Enter text to describe the contents of your rule file.

   ◦ **Subfolder** - Select one of the following subfolder locations: **Private** (for your use only and cannot be included in a normalization process) or **Shared** (public rule file).

   ◦ **Enabled** - Select this check box to allow the rule file to be applied to normalization rule task lists. (By default, the normalization rule file is disabled.)

5. Add your rules to the editor pane. For more information, see **Normalization Rule Syntax**.
6. Click **Preview**. The rule or rules in the file are applied to the specified records and the outcome is displayed.

7. Click **Save** under the editor pane.

---

### Normalization Rule Syntax

Rule files contain one or more rules, which contain one or more conditions followed by one or more actions. The following syntax is used for each rule in the file:

```
rule "<rule name>"
    when
        <condition 1>
        <condition 2>
        ...
    then
        <action 1>
        <action 2> [if <condition>]
        ...
end
```

### Conditions

A rule can contain one or more conditions that apply to specific fields and values in the source record. If all conditions are met, the rule is applied to the record. Otherwise, the rule is not applied and the record is left unchanged. Each condition must include one of the following keywords:

- **true** – Unconditionally applies the rule to all records.
- **exist** – Applies the rule when a specific field exists.
- **not exist** – Applies the rule when a specific field does not exist.
- **equals** – Applies the rule when the entire contents of the specified field matches.
- **not equals** – Applies the rule when the entire contents of the specified field does not match.
The following table provides an example for each type of condition.

### DC and XML Conditions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General Conditions:</strong></td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td><strong>DC Conditions:</strong></td>
<td></td>
</tr>
<tr>
<td>exist &quot;{namespace}&quot;.&quot;{field}&quot;</td>
<td>exist &quot;dc&quot;.&quot;description&quot;</td>
</tr>
<tr>
<td>exist &quot;{namespace}&quot;.&quot;{field}&quot;(&quot;{xsiType}&quot;)</td>
<td>exist &quot;dc&quot;.&quot;identifier&quot;(&quot;dcterms:ISBN&quot;)</td>
</tr>
<tr>
<td>&quot;{namespace}&quot;.&quot;{field}&quot; equals &quot;{value}&quot;</td>
<td>&quot;dc&quot;.&quot;identifier&quot; equals &quot;sport&quot;</td>
</tr>
<tr>
<td>&quot;{namespace}&quot;.&quot;{field}&quot;(&quot;{xsiType}&quot;) equals &quot;{value}&quot;</td>
<td>&quot;dc&quot;.&quot;identifier&quot;(&quot;dcterms:ISBN&quot;) equals &quot;sport&quot;</td>
</tr>
<tr>
<td>&quot;{namespace}&quot;.&quot;{field}&quot; not equals &quot;{value}&quot;</td>
<td>&quot;dc&quot;.&quot;identifier&quot; not equals &quot;sport&quot;</td>
</tr>
<tr>
<td>&quot;{namespace}&quot;.&quot;{field}&quot;(&quot;{xsiType}&quot;) not equals &quot;{value}&quot;</td>
<td>&quot;dc&quot;.&quot;identifier&quot;(&quot;dcterms:ISBN&quot;) not equals &quot;sport&quot;</td>
</tr>
<tr>
<td><strong>XML Conditions:</strong></td>
<td></td>
</tr>
<tr>
<td>&quot;xpathExpression&quot; equals &quot;value&quot;</td>
<td>&quot;ino:object/viaRecord/recordId&quot; equals &quot;test&quot;</td>
</tr>
<tr>
<td>&quot;xpathExpression&quot; not equals &quot;value&quot;</td>
<td>&quot;ino:object/viaRecord/recordId&quot; not equals &quot;test&quot;</td>
</tr>
<tr>
<td>exist &quot;xpathExpression&quot;</td>
<td>exist &quot;ino:object/viaRecord/recordId&quot;</td>
</tr>
<tr>
<td>not exist &quot;xpathExpression&quot;</td>
<td>not exist &quot;ino:object/viaRecord/recordId&quot;</td>
</tr>
</tbody>
</table>

### Actions

You can apply actions to single fields in a record based on conditions specified in the rule. If all of the conditions are met, the actions are applied to the record in the order in which they are listed. Each action must include one of the following keywords:
- **copy** – Copies the value from a field to another field.
- **move** – Moves the value from a field to another field.
- **remove** – Removes the value from a field.
- **set** – Sets the value of a field directly.

In addition, you can append the **if equals <value>** condition to an above action to apply rule changes for specific matching values only.

The following table provides an example for each type of action.

### DC and XML Actions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DC Actions:</strong></td>
<td></td>
</tr>
<tr>
<td>copy &quot;{namespace}&quot;.&quot;{field}&quot;(&quot;{xsiType}&quot;) to &quot;{newTagNameSpace}&quot;.&quot;{newTagField}&quot;(&quot;{newXsiType}&quot;)</td>
<td>copy &quot;dc&quot;.&quot;identifier&quot;(&quot;dcterms:ISBN&quot;) to &quot;dc&quot;.&quot;title&quot;(&quot;dcterms:ISBN&quot;)</td>
</tr>
</tbody>
</table>
| copy "{namespace}"."{field}"("{xsiType}") to "{newTagNameSpace}"."{newTagField}" | copy "dc"."identifier"("dcterms:ISBN") to "dc"."title"
<p>| copy &quot;{namespace}&quot;.&quot;{field}&quot; to &quot;{newTagNameSpace}&quot;.&quot;{newTagField}&quot;(&quot;{newXsiType}&quot;) | copy &quot;dc&quot;.&quot;identifier&quot; to &quot;dc&quot;.&quot;title&quot;(&quot;dcterms:ISBN&quot;) |
| copy &quot;{namespace}&quot;.&quot;{field}&quot; to &quot;{newTagNameSpace}&quot;.&quot;{newTagField}&quot; | copy &quot;dc&quot;.&quot;identifier&quot; to &quot;dc&quot;.&quot;title&quot; |
| copy &quot;{namespace}&quot;.&quot;{field}&quot;(&quot;{xsiType}&quot;) if equals &quot;{value}&quot; to &quot;{newTagNameSpace}&quot;.&quot;{newTagField}&quot;(&quot;{newXsiType}&quot;) | copy &quot;dc&quot;.&quot;identifier&quot;(&quot;dcterms:ISBN&quot;) if equals &quot;sport&quot; to &quot;dc&quot;.&quot;title&quot;(&quot;dcterms:ISBN&quot;) |
| copy &quot;{namespace}&quot;.&quot;{field}&quot;(&quot;{xsiType}&quot;) if equals &quot;{value}&quot; to &quot;{newTagNameSpace}&quot;.&quot;{newTagField}&quot; | copy &quot;dc&quot;.&quot;identifier&quot;(&quot;dcterms:ISBN&quot;) if equals &quot;sport&quot; to &quot;dc&quot;.&quot;title&quot; |
| copy &quot;{namespace}&quot;.&quot;{field}&quot; if equals &quot;{value}&quot; to &quot;{newTagNameSpace}&quot;.&quot;{newTagField}&quot;(&quot;{newXsiType}&quot;) | copy &quot;dc&quot;.&quot;identifier&quot; if equals &quot;sport&quot; to &quot;dc&quot;.&quot;title&quot;(&quot;dcterms:ISBN&quot;) |
| copy &quot;{namespace}&quot;.&quot;{field}&quot; if equals &quot;{value}&quot; to &quot;{newTagNameSpace}&quot;.&quot;{newTagField}&quot; | copy &quot;dc&quot;.&quot;identifier&quot; ) if equals &quot;sport&quot; to &quot;dc&quot;.&quot;title&quot; |
| move &quot;{namespace}&quot;.&quot;{field}&quot;(&quot;{xsiType}&quot;) to &quot;{newTagNameSpace}&quot;.&quot;{newTagField}&quot;(&quot;{newXsiType}&quot;) | move &quot;dc&quot;.&quot;identifier&quot;(&quot;dcterms:ISBN&quot;) to &quot;dc&quot;.&quot;title&quot;(&quot;dcterms:ISBN&quot;) |</p>
<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>move</strong> &quot;{namespace}&quot;.&quot;{field}&quot;(&quot;xsiType&quot;) to &quot;{newTagNameSpace}&quot;.&quot;{newTagField}&quot;</td>
<td>move &quot;dc&quot;.&quot;identifier&quot;(&quot;dcterms:ISBN&quot;) to &quot;dc&quot;.&quot;title&quot;</td>
</tr>
<tr>
<td><strong>move</strong> &quot;{namespace}&quot;.&quot;{field}&quot; to &quot;{newTagNameSpace}&quot;.&quot;{newTagField}&quot;((&quot;newXsiType&quot;))</td>
<td>move &quot;dc&quot;.&quot;identifier&quot; to &quot;dc&quot;.&quot;title&quot;(&quot;dcterms:ISBN&quot;)</td>
</tr>
<tr>
<td><strong>move</strong> &quot;{namespace}&quot;.&quot;{field}&quot; to &quot;{newTagNameSpace}&quot;.&quot;{newTagField}&quot;</td>
<td>move &quot;dc&quot;.&quot;identifier&quot; to &quot;dc&quot;.&quot;title&quot;</td>
</tr>
<tr>
<td><strong>move</strong> &quot;{namespace}&quot;.&quot;{field}&quot;(&quot;xsiType&quot;) to &quot;{newTagNameSpace}&quot;.&quot;{newTagField}&quot;(&quot;newXsiType&quot;)</td>
<td>move &quot;dc&quot;.&quot;identifier&quot;(&quot;dcterms:ISBN&quot;) if equals &quot;sport&quot; to &quot;dc&quot;.&quot;title&quot;(&quot;dcterms:ISBN&quot;)</td>
</tr>
<tr>
<td><strong>move</strong> &quot;{namespace}&quot;.&quot;{field}&quot;(&quot;xsiType&quot;) if equals &quot;{value}&quot; to &quot;{newTagNameSpace}&quot;.&quot;{newTagField}&quot;</td>
<td>move &quot;dc&quot;.&quot;identifier&quot;(&quot;dcterms:ISBN&quot;) if equals &quot;sport&quot; to &quot;dc&quot;.&quot;title&quot;</td>
</tr>
<tr>
<td><strong>move</strong> &quot;{namespace}&quot;.&quot;{field}&quot; to &quot;{newTagNameSpace}&quot;.&quot;{newTagField}&quot;((&quot;newXsiType&quot;))</td>
<td>move &quot;dc&quot;.&quot;identifier&quot; if equals &quot;sport&quot; to &quot;dc&quot;.&quot;title&quot;(&quot;dcterms:ISBN&quot;)</td>
</tr>
<tr>
<td><strong>move</strong> &quot;{namespace}&quot;.&quot;{field}&quot; if equals &quot;{value}&quot; to &quot;{newTagNameSpace}&quot;.&quot;{newTagField}&quot;</td>
<td>move &quot;dc&quot;.&quot;identifier&quot; if equals &quot;sport&quot; to &quot;dc&quot;.&quot;title&quot;</td>
</tr>
<tr>
<td><strong>remove</strong> &quot;{namespace}&quot;.&quot;{field}&quot; if equals &quot;{value}&quot;</td>
<td>remove &quot;dc&quot;.&quot;identifier&quot; if equals &quot;sport&quot;</td>
</tr>
<tr>
<td><strong>remove</strong> &quot;{namespace}&quot;.&quot;{field}&quot;(&quot;xsiType&quot;) if equals &quot;{value}&quot;</td>
<td>remove &quot;dc&quot;.&quot;identifier&quot;(&quot;dcterms:ISBN&quot;) if equals &quot;sport&quot;</td>
</tr>
<tr>
<td><strong>remove</strong> &quot;{namespace}&quot;.&quot;{field}&quot;(&quot;xsiType&quot;)</td>
<td>remove &quot;dc&quot;.&quot;identifier&quot;(&quot;dcterms:ISBN&quot;)</td>
</tr>
<tr>
<td><strong>remove</strong> &quot;{namespace}&quot;.&quot;{field}&quot;</td>
<td>remove &quot;dc&quot;.&quot;identifier&quot;</td>
</tr>
<tr>
<td><strong>set</strong> &quot;{value}&quot; in &quot;{namespace}&quot;.&quot;{field}&quot;(&quot;xsiType&quot;)</td>
<td>set &quot;sport&quot; in &quot;dc&quot;.&quot;identifier&quot;(&quot;dcterms:ISBN&quot;)</td>
</tr>
<tr>
<td><strong>set</strong> &quot;{value}&quot; in &quot;{namespace}&quot;.&quot;{field}&quot;</td>
<td>set &quot;sport&quot; in &quot;dc&quot;.&quot;identifier&quot;</td>
</tr>
</tbody>
</table>

**XML Actions:** (For more information, see [W3C XPATH Standards](https://www.w3.org/TR/xpath/).)
<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
</table>
| copy "concat ((xpathExpression), ", (xpathExpression))" to "{namespace}"."{field}" | copy "concat(//person[@pers_role='pau']/name/first/text(), ", //person[@pers_role='pau']/name/last/text())" to "dc"."creator"
| copy "xpathExpression" to "newTagNameSpace"."newTagField"("xsiType") | copy "ino:object/viaRecord/isbn" to "dcterms"."title"("dcterms:ISBN") |
| copy "xpathExpression" to "newTagNameSpace"."newTagField" | copy "ino:object/viaRecord/isbn" to "dcterms"."title"
| set "value" in "newTagNameSpace"."newTagField"("xsiType") | set "whatever" in "dcterms"."title"("dcterms:ISBN")
| set "value" in "newTagNameSpace"."newTagField" | set "whatever" in "discovery"."local1"

**DC Examples**

rule "move description"
when
  exist "dc"."description"
then
  move "dc"."description" to "dcterms"."description"
end

rule "move description"
when
  exist "dc"."identifier"("dcterms:ISBN")
then
end

rule "move description"
when
  "dc"."identifier"("dcterms:ISBN") equals "sport1111"
then

Note

The `concat` keyword is an XPATH function, which is one of many XPATH functions that can be used within the XML actions.

rule "dc identifier equals sport"
  when
    "dc"."identifier"("dcterms:ISBN") not equals "sport"
  then
    move "dc"."title" to "dcterms"."description"
end

rule "from dc:type A to discovery:type C"
  when
    "dc"."type" equals "Photograph"
  then
    set "images" in "discovery"."resourceType"
end

rule "from dc:type A to discovery:type D"
  when
    "dc"."type" equals "Poster"
  then
    set "images" in "discovery"."resourceType"
end

XML Examples

rule "Copy identifier"
  when
    exist "ino:object/viaRecord/recordId"
  then
    copy "ino:object/viaRecord/recordId" to "dc"."identifier"
end

rule "copy first title"
  when
    true
  then
    copy "(/title/textElement)[1]" to "dc"."title"
end

rule "copy all titles"
when
  true
then
  copy "(/*title/textElement)" to "dcterms"."title"
end

rule "creator"
  when
    exist "ino:object/viaRecord/work/creator/nameElement"
  then
    copy "ino:object/viaRecord/work/creator/nameElement" to "dc"."creator"
end

rule "equals english"
  when
    "ino:object/viaRecord/work/lang/textElement" equals "eng"
  then
    set "English" in "dc"."language"
end

rule "set Constant"
  when
    true
  then
    set "test subject" in "dc"."subject"
end