Repository Tools 2

Defining Crosswalks Guide

Version 1.2 , created 6 January 2017

Updated 28 April 2016

Contents

Introduction

This document describes how to define crosswalks to copy item data into and out of Symplectic Elements for a Repository Tools 2 integration. This document is designed primarily for technical staff to enable them to create and amend crosswalk map files.

For an overview of Repository Tools 2 and the crosswalking process see:

- Repository Tools 2 - A Functional Overview
- Repository Tools 2 - Repository as Data Source
- Repository Tools 2 - Enabling deposit via Elements

A crosswalk defines how an item's data values are set when the item is copied from one system into another, e.g. when harvesting an item from a DSpace repository into Elements or when depositing a publication from Elements into a repository. Elements uses a crosswalk map file (often referred to simply as a 'crosswalk map' or a 'crosswalk') to define this. This is a structured XML file that conforms to a published schema. (We should make these schema files available on the support site & reference them here.)

Sample crosswalk map files are provided with Elements, however all Elements users will want to adjust these files to ensure they reflect the data structures and processes used in their organisation.

This document describes how the crosswalk maps are defined, at a detailed level.

Components of a crosswalk map file

All crosswalk map files share a set of common parts, with some components that depend on whether the map describes an inbound crosswalk (i.e. harvesting of data into Elements) or an outbound crosswalk (i.e. depositing data from Elements).

All crosswalk maps *must* contain:

1. One or more definitions of how individual fields should be mapped from the source system into the destination system (<xwalk:field-map> elements within <xwalk:field-maps>). Multiple definitions (<xwalk:field-map> elements) may be required in order to support different item types.

All crosswalk maps *may* contain:

2. Parameters to adjust some aspects of the crosswalk behaviour (<xwalk:parameters> element). Parameters are specific to individual repositories and crosswalk directions.
3. Elements metadata to inform the crosswalk (<xwalk:elements-metadata> element). Elements will automatically provide this data in normal operation.
4. Value maps to provide text lookup, replacement and filtering capabilities (<xwalk:value-maps> element).

An inbound crosswalk map *must* contain:

5. A selector to define which object type and field map should be used for an inbound item (<xwalkin:object-type-selector> element).

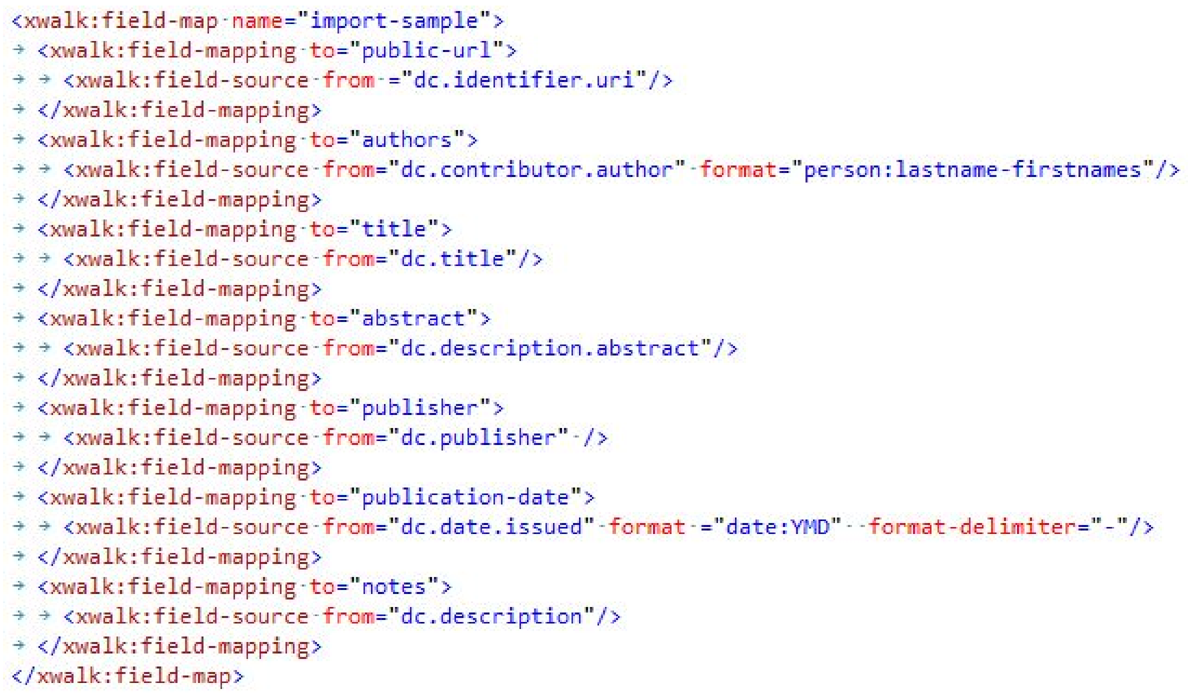
An outbound crosswalk map *must* contain:

6. A selector to define which field map should be used for an outbound item (<xwalkout:field-map-selector> element).
7. A selector to define which collection should be used for an outbound item (<xwalkout:collection-selector> element).

The <xwalk:field-map> Element

This element defines how a set of fields should be mapped from a source system to a destination system.

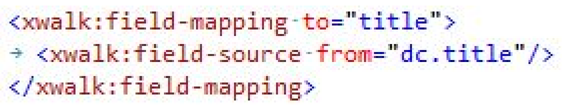
Multiple <xwalk:field-map> elements may exist within a <xwalk:field-maps> element. Each has a name attribute that is used to identify it. A simple field map is shown below.



```
<xwalk:field-map name="import-sample">
	<xwalk:field-mapping to="public-url">
		<xwalk:field-source from ="dc.identifier.uri"/>
	</xwalk:field-mapping>
	<xwalk:field-mapping to="authors">
		<xwalk:field-source from="dc.contributor.author" format="person:lastname-firstnames"/>
	</xwalk:field-mapping>
	<xwalk:field-mapping to="title">
		<xwalk:field-source from="dc.title"/>
	</xwalk:field-mapping>
	<xwalk:field-mapping to="abstract">
		<xwalk:field-source from="dc.description.abstract"/>
	</xwalk:field-mapping>
	<xwalk:field-mapping to="publisher">
		<xwalk:field-source from="dc.publisher" />
	</xwalk:field-mapping>
	<xwalk:field-mapping to="publication-date">
		<xwalk:field-source from="dc.date.issued" format ="date:YMD"  format-delimiter="-"/>
	</xwalk:field-mapping>
	<xwalk:field-mapping to="notes">
		<xwalk:field-source from="dc.description"/>
	</xwalk:field-mapping>
</xwalk:field-map>
```

A simple field mapping

Each destination field is defined using a <xwalk:field-mapping> element, with the to attribute specifying the identifier of the destination field. At least one <xwalk:field-source> element is used to define the source data, with the from attribute specifying the identifier of the source field.



```
<xwalk:field-mapping to="title">
	<xwalk:field-source from="dc.title"/>
</xwalk:field-mapping>
```

The sample above shows a simple mapping - the Elements ‘title’ field is to be taken from the ‘dc.title’ field in the source system.

Field Identifiers

Identifiers are used to distinguish individual fields in both source and destination systems.

Elements Field Identifiers

Elements uses field identifiers that can be found on the Module Admin, Underlying Fields page and when viewing object type definitions. These are typically meaningful words, e.g. title, abstract, authors etc. Any Elements custom fields can also be referenced - identifiers for these begin c-xxx.

When making a deposit from Elements, a limited number of ‘special’ identifiers can also be used to reference other Elements fields:

- *object.type*: the Elements object type (journal-article, book etc)
- *object.id*: the Elements internal object id
- *object.reporting-date-1, object.reporting-date-2*: the object reporting dates
- *object.labels, publication.labels*: the list of all labels for the publication .

Repository Field Identifiers

Each type of repository defines its own set of identifiers. For example DSpace uses Dublin Core values - dc.title, dc.contributor.authors etc.

Text field mapping

Text fields are the simplest to map as they involve the manipulation of a simple string value. In addition to the simple format above, several additional attributes can be used to specify how a string value should be transformed.

Splitting values using a delimiter

A text value can be split into multiple values using the split-using-delimiter attribute.

```
<xwalk:field-mapping to ="keywords">
    <xwalk:field-source from="dc.keyword.list" split-using-delimiter =";"/>
</xwalk:field-mapping>
```

The above example will split a list of keywords into separate values using the semicolon character. (The Elements keywords field is a list of string values.)

Concatenating multi-value fields

Some fields contain multiple values, but need to be stored as a single string value in the destination field. The concatenate-with-separator attribute can be used to achieve this.

```
<xwalk:field-mapping to ="c-classifications">
    <xwalk:field-source from="dc.subject.classification" concatenate-with-separator=", "/>
</xwalk:field-mapping>
```

The above example concatenates all the dc.subject.classification values, separating them with ‘, ‘ and puts the result into the c-classifications field..

Performing text look-ups and amendments

There are times when a string value needs to be looked up or amended. Crosswalk files enable the use of value-maps to lookup a value, or substrings within a value, and replace the incoming string with another. This can be used to convert between differing sets of options in source and destination systems, remove or replace substrings or interpret free-format values into options.

```
<xwalk:field-mapping to ="c-is-published">
    <xwalk:field-source from="dc.ispublished" value-map="boolean-text-lookup"/>
</xwalk:field-mapping>
```

For full details see the section on the <xwalk:value-maps> Element'.

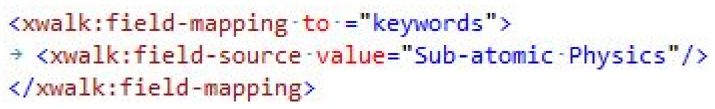
Adding Prefixes and Suffixes

A prefix or suffix can be specified as part of a <xwalk:field-source> element, using the prefix and suffix attributes.

```
<xwalk:field-mapping to ="c-funder-info">
    <xwalk:field-source from="dc.grant.organisation" prefix="This research was supported by " suffix="."/>
</xwalk:field-mapping>
```

Constant (or hard-coded) values

You can specify a constant value for an item using the value attribute (with no from attribute):

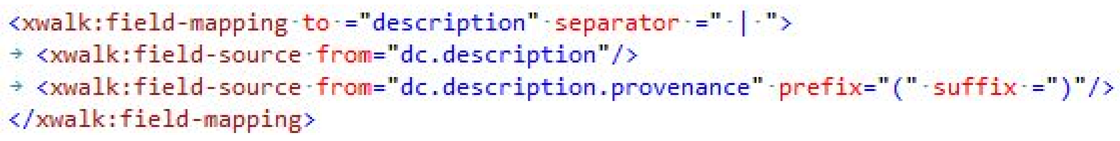


```
<xwalk:field-mapping to ="keywords">
	<xwalk:field-source value="Sub-atomic Physics"/>
</xwalk:field-mapping>
```

The above mapping will always set the keywords field to ‘Sub-atomic Physics’.

Multiple source field values

It is possible to specify multiple source fields for a field mapping. In this situation the values of each field-source are concatenated together, using the separator attribute of the <xwalk:field-mapping> element:



```
<xwalk:field-mapping to ="description" separator =" | ">
	<xwalk:field-source from="dc.description"/>
	<xwalk:field-source from="dc.description.provenance" prefix="(" suffix =")"/>
</xwalk:field-mapping>
```

Other scalar data type mapping

The following scalar data types operate in a similar manner to text values:

- Boolean
- Integer
- Decimal
- URL
- ISBN-10
- ISBN-13
- ISSN
- DOI
- Choice
- List of strings.

Compound data types

Compound data types (such as dates, person lists etc) usually need more information than scalar data types in order to be accurately interpreted (inbound crosswalks) or to build data values (outbound crosswalks). This means that additional information may be needed when defining field mappings.

Formats

When interpreting (and building) compound data strings we need to know the format of the string - we specify this using the format attribute, and sometimes also with a format-separator attribute. For example, consider a string ‘2015-09-03’ that is to be interpreted as a date. This would be interpreted differently if the format was YMD than if the format was YDM, and would not be interpreted at all if we expected a ‘/’ as the separator instead of the ’-’.

Data parts

On other occasions we may not have a single string, but we may have access to the parts of a compound value - perhaps we have separate values for the year, month and day of a date. So long as we know what part of the date each item represents we can accurately interpret the date. We can specify the parts of a compound data type using the data-part attribute.

Date field mapping

Elements date fields hold date information. These fields require a year component, and may include month and day components.

Formats for date fields

The following date format attribute values are supported:

- date:YYYY-MM-DD (this is the default format)
- date:DMY
- date:MDY
- date:YMD (equivalent to date:YYYY-MM-DD, the default format)
- date:YM
- date:MY
- date:Y.

Dates also make use of the format-delimiter attribute. This defaults to '-'.

Data parts for date fields

The following date data-part attribute values are supported:

- date:day
- date:month
- date:year.

Interpreting date strings

If a date string uses the default format, mapping is performed without using any additional attributes:

```
<xwalk:field-mapping to="publication-date">
    <xwalk:field-source from="dc.date.issued"/>
</xwalk:field-mapping>
```

This is equivalent to:

```
<xwalk:field-mapping to="publication-date">
    <xwalk:field-source from="dc.date.issued" format ="date:YMD"  format-delimiter="-"/>
</xwalk:field-mapping>
```

In the above mapping we have explicitly specified format and format-delimiter attributes.

Interpreting data parts

The following field-mapping would interpret the date as 17 February 2016:

```
<xwalk:field-mapping to ="publication-date">
    <xwalk:field-source data-part="date:year" value="2016"/>
    <xwalk:field-source data-part="date:month" value="2"/>
    <xwalk:field-source data-part="date:day" value="17"/>
</xwalk:field-mapping>
```

Money field mapping

Elements money fields hold financial amount information, including a amount value and a currency.

Formats for money fields

The following money format attribute values are supported:

- money:usd (this is the default format)
- money:gbp
- money:aud
- money:nzd
- money:cad
- money:eur
- money:jpy
- money:sgd.

Data parts for money fields

The following money data-part attribute values are supported:

- money:amount
- money:currency.

Interpreting money strings

A value can be mapped to the the default money format (USD) as shown below:

```
<xwalk:field-mapping to ="amount">
    <xwalk:field-source from ="dc.amount"/>
</xwalk:field-mapping>
```

And the mapping below specifies that the amount is in GBP:

```
<xwalk:field-mapping to ="amount">
    <xwalk:field-source from ="dc.amount" format="money:gbp"/>
</xwalk:field-mapping>
```

Interpreting data parts

If the amount was USD 34.00 the following field-mapping would set the destination field to a text string of ‘34.00[USD]’:

```
<xwalk:field-mapping to ="dc-money-custom-format">
    <xwalk:field-source from ="amount" data-part ="money:amount"/>
    <xwalk:field-source from ="amount" data-part ="money:currency" prefix="[" suffix="]"/>
</xwalk:field-mapping>
```

Person list field mapping

Elements uses person-list fields to store lists of people (authors, editors etc).

Formats for person list fields

The following person format attribute values are supported:

- person:lastname-initials (this is the default format)
- person:initials-lastname
- person:lastname-firstnames
- person:firstnames-lastname.

Data parts for person list fields

The following person data-part attribute values are supported:

- person:lastname
- person:initials
- person:firstnames.

Interpreting person list strings

A simple field mapping is shown below:

```
<xwalk:field-mapping to ="authors">
	<xwalk:field-source from="dc.contributor.author"/>
</xwalk:field-mapping>
```

This is equivalent to:

```
<xwalk:field-mapping to ="authors">
	<xwalk:field-source from="dc.contributor.author"  format="person:lastname-initials"/>
</xwalk:field-mapping>
```

Note the dc.contributor.author field above may have multiple occurrences (one for each author) and will be interpreted as multiple people within the destination authors field. If we were interpreting a single string which contained all the authors, each separated by a semi-colon, we would use the mapping below:

```
<xwalk:field-mapping to ="authors">
	<xwalk:field-source from="dc.contributor.authors" split-using-delimiter=";"/>
</xwalk:field-mapping>
```

Interpreting person list parts

The following field-mapping would interpret the person as Smith, DK:

```
<xwalk:field-mapping to ="authors">
	<xwalk:field-source data-part="person:lastname" value="Smith"/>
	<xwalk:field-source data-part="person:initials" value="DK"/>
</xwalk:field-mapping>
```

Address list field mapping

Elements uses address-list fields to store lists of addresses - in most cases the lists consist of a single address.

Formats for address list fields

The following address format attribute values are supported:

- address:full-multi-line (this is the default format)
- address:full-single-line
- address:name
- address:organisation
- address:suborganisation
- address:streetaddress
- address:city
- address:state
- address:zipcode
- address:country
- address:type
- address:iso-country-code.

Data parts for address list fields

The following address data-part attribute values are supported:

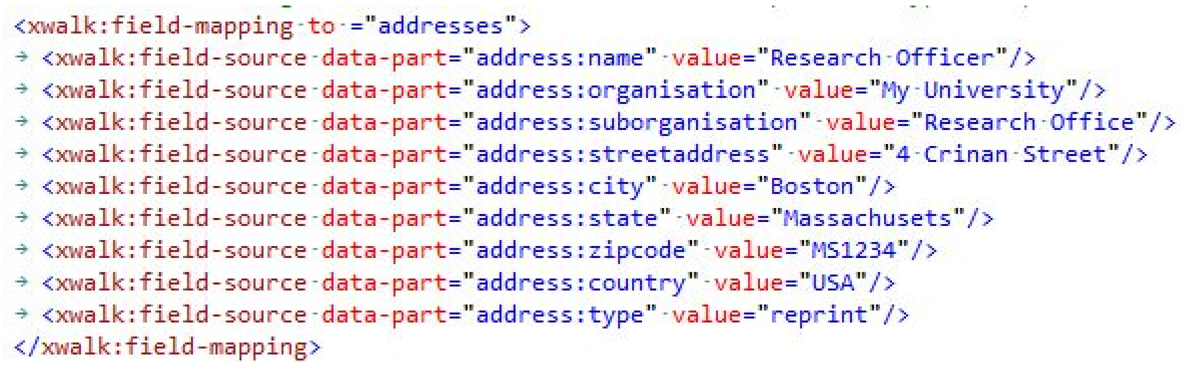
- address:name
- address:organisation
- address:suborganisation
- address:streetaddress
- address:city
- address:state
- address:zipcode
- address:country
- address:type.

Interpreting address list strings

With an inbound crosswalk address-list fields must be defined using data-part attributes - interpreting a single string as a formatted address or addresses is not supported.

Interpreting address list parts

The following field-mapping would specify a hard-coded address:



```
<xwalk:field-mapping to ="addresses">
	<xwalk:field-source data-part="address:name" value="Research Officer"/>
	<xwalk:field-source data-part="address:organisation" value="My University"/>
	<xwalk:field-source data-part="address:suborganisation" value="Research Office"/>
	<xwalk:field-source data-part="address:streetaddress" value="4 Crinan Street"/>
	<xwalk:field-source data-part="address:city" value="Boston"/>
	<xwalk:field-source data-part="address:state" value="Massachusets"/>
	<xwalk:field-source data-part="address:zipcode" value="MS1234"/>
	<xwalk:field-source data-part="address:country" value="USA"/>
	<xwalk:field-source data-part="address:type" value="reprint"/>
</xwalk:field-mapping>
```

Pagination field mapping

Elements uses pagination fields to track the pages of a publication.

Formats for pagination fields

The following pagination format attribute values are supported:

- pagination:start-end (this is the default format)
- pagination:begin-page
- pagination:end-page
- pagination:page-count.

By default the format-delimiter attribute is ‘-’, but may be overridden.

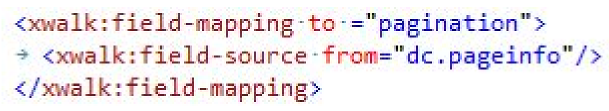
Data parts for pagination fields

The following pagination data-part attribute values are supported:

- pagination:begin-page
- pagination:end-page
- pagination:page-count.

Interpreting pagination strings

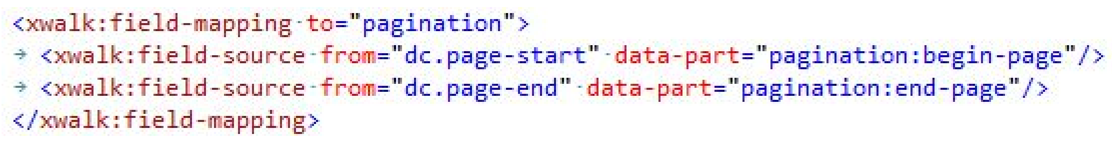
Using the default pagination format and delimiter, we can define a field mapping as shown below:



```
<xwalk:field-mapping to ="pagination">
	<xwalk:field-source from="dc.pageinfo"/>
</xwalk:field-mapping>
```

Interpreting data parts

The following field-mapping would interpret the begin and end page parts specified:



```
<xwalk:field-mapping to="pagination">
	<xwalk:field-source from="dc.page-start" data-part="pagination:begin-page"/>
	<xwalk:field-source from="dc.page-end" data-part="pagination:end-page"/>
</xwalk:field-mapping>
```

External identifier field mapping

Elements uses external identifier fields to track identifiers of a publication. Each external identifier is defined using an identifier string and a data value identifying the type of identifier.

Formats for external identifier fields

The following external identifier format attribute values are supported:

- identifier:arxiv
- identifier:arxiv-author-id"/>
- identifier:cinii-article-id"/>
- identifier:crossref-article-id"/>
- identifier:dais"/>
- identifier:dimensions-grant-id"/>
- identifier:email-address"/>
- identifier:epmc-article-id"/>
- identifier:figshare-for-institutions-user-account-id"/>
- identifier:fundref-id"/>
- identifier:google-books-id"/>
- identifier:isidoc"/>
- identifier:local-scival-expert-id"/>
- identifier:mla-record-id"/>
- identifier:nihms"/>
- identifier:orcid"/>
- identifier:pmc"/>
- identifier:pubmed"/>
- identifier:repec-article-id"/>
- identifier:researcherid"/>
- identifier:scopus-author-id"/>
- identifier:ssrn-article-id"/>
- identifier:ssrn-author-id
- identifier:value-with-scheme
- identifier:value
- identifier:scheme.

Data parts for external identifier fields

The following external identifier data-part attribute values are defined, but not used:

- identifier:value
- identifier:scheme.

Interpreting external identifier strings

The field mapping below maps pubmed and email address identifiers::

```
<xwalk:field-mapping to="external-identifiers">
	<xwalk:field-source from="dc.pubmed.id" format="identifier:pubmed"/>
	<xwalk:field-source from="dc.author.email" format="identifier:email-address"/>
</xwalk:field-mapping>
```

Interpreting data parts

Data-parts are not used for external identifiers.

Keyword list field mapping

Elements uses keyword list fields to track the keywords associated with a publication. In addition to a keyword value, keywords may also have an associated scheme and a percentage.

Formats for keyword list fields

The following keyword list format attribute values are supported:

- keyword:value (this is the default format)
- keyword:scheme
- keyword:percent
- keyword:with-scheme
- keyword:with-percent
- keyword:with-scheme-percent.

Data parts for keyword list fields

The following keyword list data-part attribute values are supported:

- keyword:value
- keyword:scheme
- keyword:percent.

Interpreting keyword strings

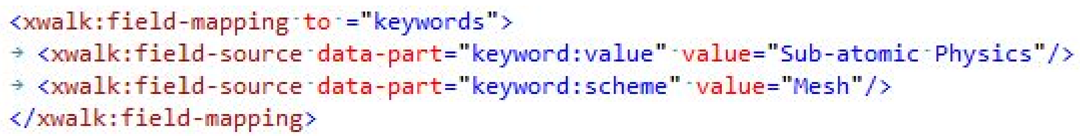
A string of keywords separated by semicolons can be mapped using the field mapping below:

```
<xwalk:field-mapping to ="keywords">
	<xwalk:field-source from="dc.keyword.list" split-using-delimiter =";"/>
</xwalk:field-mapping>
```

This field mapping will not assign a scheme or percentage to the keywords.

Interpreting data parts

The following field-mapping would define a Mesh keyword of ‘Sub-atomic Physics’.



```
<xwalk:field-mapping to ="keywords">
	<xwalk:field-source data-part="keyword:value" value="Sub-atomic Physics"/>
	<xwalk:field-source data-part="keyword:scheme" value="Mesh"/>
</xwalk:field-mapping>
```

Funding acknowledgements field mapping

Elements uses funding acknowledgement fields to record funding related information within a publication.

Formats for funding acknowledgements fields

The following funding acknowledgements format attribute values are supported:

- funding-acknowledgements-grant:id
- funding-acknowledgements-grant:org
- funding-acknowledgements-grant:id-org
- funding-acknowledgements-grant:org-id.

Data parts for funding acknowledgements fields

The following funding acknowledgements data-part attribute values are supported:

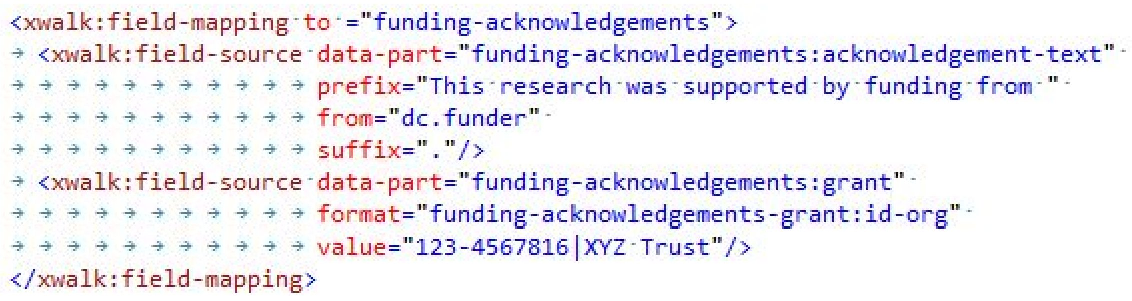
- funding-acknowledgements:acknowledgement-text
- funding-acknowledgements:grant
- funding-acknowledgements-grant:id
- funding-acknowledgements-grant:org.

Interpreting funding acknowledgements strings

With an inbound crosswalk funding acknowledgement fields must be defined using data-part attributes - interpreting a single string as a funding acknowledgement is not supported.

Interpreting data parts

A funding acknowledgement field-mapping is shown below:



```
<xwalk:field-mapping to ="funding-acknowledgements">
	<xwalk:field-source data-part="funding-acknowledgements:acknowledgement-text" 
											prefix="This research was supported by funding from " 
											from="dc.funder" 
											suffix="."/>
	<xwalk:field-source data-part="funding-acknowledgements:grant" 
											format="funding-acknowledgements-grant:id-org" 
											value="123-4567816|XYZ Trust"/>
</xwalk:field-mapping>
```

Advanced field-mapping techniques

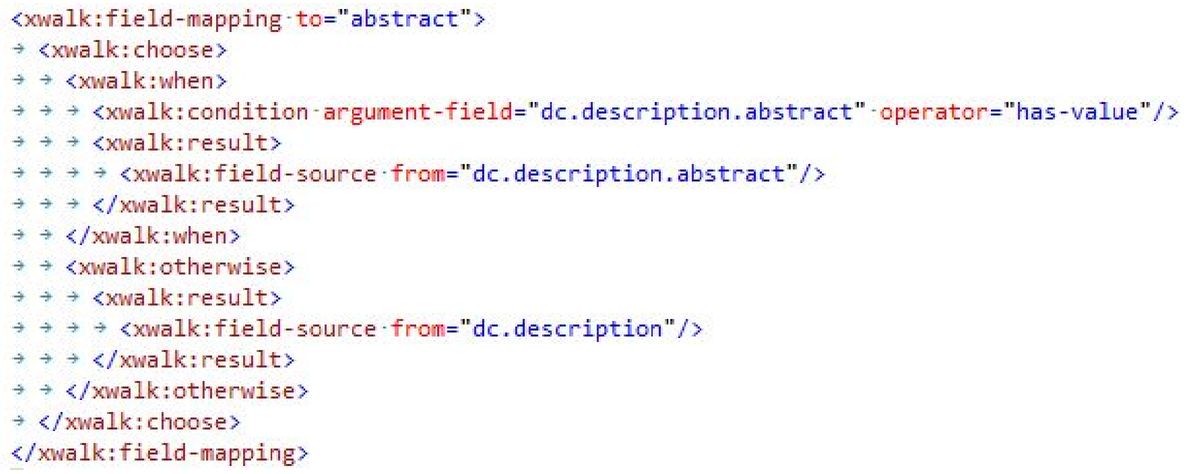
Defining conditional logic

There are times when a field mapping may depend on the data being processed. For instance:

- a single field map may be used for several object types, but the definition of one field mapping may vary by field type
- A field mapping should use one field if a value is present, otherwise use another one.

In both of the above examples some logic is needed within the <xwalk:field-mapping> element to define the desired behaviour. This is supported through the use of logic expressions. For more information, see Logic expressions - <xwalk:if> and <xwalk:choose> Elements.

The example below specifies the that abstract field should be populated from the dc.description.abstract field if it has a value, otherwise from the dc.description field.



```
<xwalk:field-mapping to="abstract">
	<xwalk:choose>
		<xwalk:when>
			<xwalk:condition argument-field="dc.description.abstract" operator="has-value"/>
			<xwalk:result>
				<xwalk:field-source from="dc.description.abstract"/>
			</xwalk:result>
		</xwalk:when>
		<xwalk:otherwise>
			<xwalk:result>
				<xwalk:field-source from="dc.description"/>
			</xwalk:result>
		</xwalk:otherwise>
	</xwalk:choose>
</xwalk:field-mapping>
```

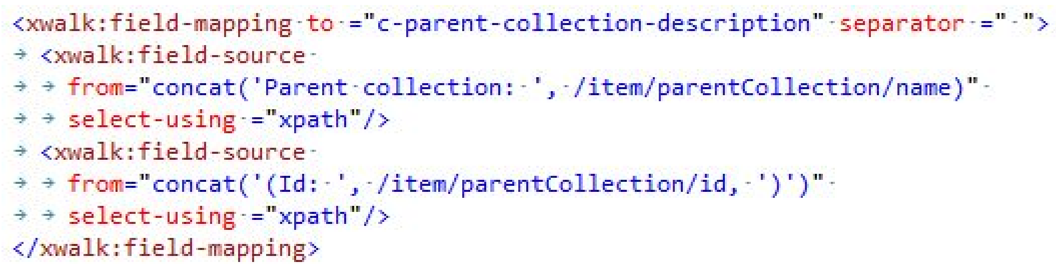
Using XPath to identify source data values

Important note: Using XPath should be avoided if possible - it relies on an assumed knowledge of the underlying xml data structures used by the crosswalk and these may change over time.

When specifying which data values are to be used we typically use field identifiers from the source and destination systems. In most situations this works fine, but there are times when this may not provide the desired flexibility/capabilities. Examples might include:

- Accessing data values that are not part of the normal metadata information (e.g. collection information)
- Performing XPath functions on data values prior to passing it into a crosswalk.

When defining a <xwalk:field-source> element the from attribute can be set to an XPath statement instead of a field identifier, and the select-using attribute set to ‘xpath’. This instructs the crosswalk to evaluate the XPath statement and pass the result for processing.

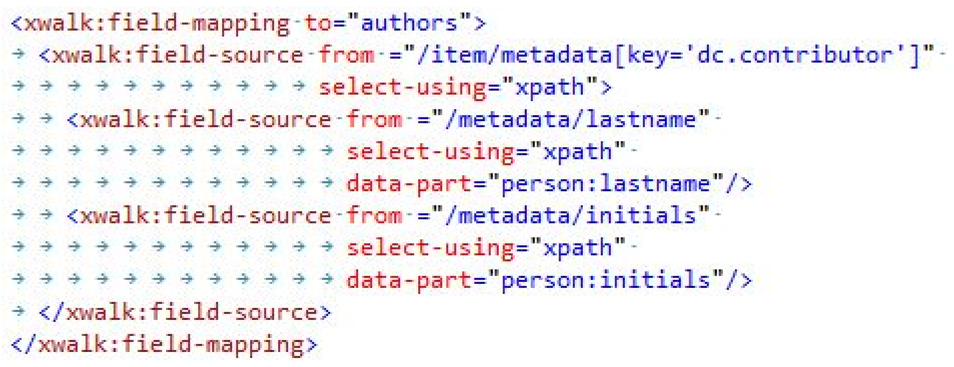


```
<xwalk:field-mapping to ="c-parent-collection-description" separator =" ">
    <xwalk:field-source
        from="concat('Parent collection: ', /item/parentCollection/name)"
        select-using ="xpath"/>
    <xwalk:field-source
        from="concat('(Id: ', /item/parentCollection/id, ')')"
        select-using ="xpath"/>
</xwalk:field-mapping>
```

The example above uses two field sources, each of which uses the XPath concat function to concatenate a text string with the result of an XPath statement. The results of each are concatenated using a ‘ ‘ separator and put into the c-parent-collection-description field.

Recursive field sources

There may be some occasions when it is not possible to extract and format data using a simple set of flat <xwalk:field-source> elements. In order to provide more flexibility when building some of the more complex data items it is possible to define <xwalk:field-source> elements recursively when using Xpath to identify data items.



```
<xwalk:field-mapping to="authors">
    <xwalk:field-source from ="/item/metadata[key='dc.contributor']"
                                            select-using="xpath">
        <xwalk:field-source from ="/metadata/lastname"
                                                select-using="xpath"
                                                data-part="person:lastname"/>
        <xwalk:field-source from ="/metadata/initials"
                                                select-using="xpath"
                                                data-part="person:initials"/>
    </xwalk:field-source>
</xwalk:field-mapping>
```

It is recommended that you seek advice from Symplectic support if you believe that you need to use this type of structure.

The <xwalk:parameters> Element

This optional element can be used to define parameters that control the behaviour of a crosswalk. Each supported parameter has a name and a value.

Generic Parameters

Some parameters are generic and apply to all crosswalks.

Parameter - default-currency

The default-currency parameter allows the default currency to be overridden. If this parameter is not specified the default currency will be “USD”.

Parameter - default-date-format

The default-date-format parameter allows the default date format to be overridden. If this parameter is not specified the default date format will be “date:YYYY-MM-DD”.

Parameter - default-date-delimiter

The default-date-delimiter parameter allows the default date format delimiter to be overridden. If this parameter is not specified the default date format delimiter will be “-”.

Parameter - default-person-format

The default-person-format parameter allows the default person format to be overridden. If this parameter is not specified the default person format will be “person:lastname-initials”.

Parameter - default-pagination-format

The default-pagination-format parameter allows the default pagination format to be overridden. If this parameter is not specified the default pagination format will be “pagination:start-end”.

Parameter - default-pagination-delimiter

The default-pagination-delimiter parameter allows the default pagination format delimiter to be overridden. If this parameter is not specified the default pagination delimiter will be “-”.

Parameter - default-funding-acknowledgements-grant-format

The default-funding-acknowledgements-grant-format parameter allows the default funding acknowledgements grant format to be overridden. If this parameter is not specified the default format will be “funding-acknowledgements-grant:org-id”.

Parameter - default-funding-acknowledgements-grant-delimiter

The default-funding-acknowledgements-grant-delimiter parameter allows the default funding acknowledgements grant delimiter to be overridden. If this parameter is not specified the default delimiter will be “|”.

Inbound crosswalk parameters

The following parameters are applicable to inbound crosswalks:

Parameter - xwalkin-without-files

The xwalkin-without-files parameter controls whether information about files (or ‘bitstreams’) is crosswalked. By default information about files *is* crosswalked, unless this parameter is present and it has a value which is not ‘false’, ‘0’ or ‘’.

Parameter - xwalkin-without-file-links

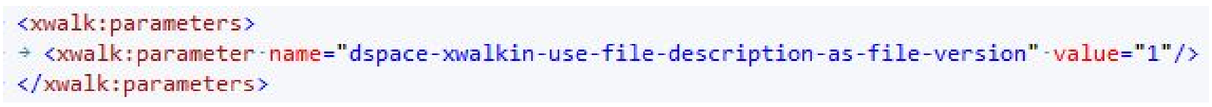
The xwalkin-without-file-links parameter controls whether a file url link should be crosswalked. By default a file url link *is* crosswalked, unless this parameter is present and it has a value which is not 'false', '0' or ''.

DSpace specific parameters

The following parameters are applicable to DSpace:

Parameter - dspace-xwalkin-use-file-description-as-file-version

The dspace-xwalkin-use-file-description-as-file-version parameter specifies whether a DSpace bitstream file description field should be treated as the file version when crosswalking data into Elements. This parameter it is treated as true if it is present and has a value which is not 'false', '0' or ''.



```
<xwalk:parameters>
    <xwalk:parameter name="dspace-xwalkin-use-file-description-as-file-version" value="1"/>
</xwalk:parameters>
```

Parameter - metadata-format

This parameter is reserved for future use.

The <xwalk:elements-metadata> Element

Elements will automatically provide this data in normal operation - it is an optional part of a crosswalk map file.

The <xwalk:elements-metadata> element can be used to define the Elements metadata definitions to be used during the crosswalk process. In normal execution from within Elements this data is provided by Elements, but if a crosswalk is being performed outside of the Elements environment this data must be provided. For example, if you have a good knowledge of XSL/XML, you may wish to test your crosswalks based on sample XML data outside of Elements. Please note to do so will require Visual Studio. The sample file contains a default value.

This Elements metadata is used in the inbound crosswalk map for the following purposes:

- To validate that all destination field identifiers exist (the 'to' attribute of <xwalk:field-mapping to="public-url">) Public URL is the field used for Handles.
- To determine the data type for each Elements field so the crosswalk can format the value accordingly.

This data is currently not used for the outbound crosswalk.

The <xwalk:value-maps> Element

This optional element is used to define lookup tables that can be used to substitute strings or substrings with other values, or filter values during the crosswalk process. Often referred to as ‘value maps’ as they are used to map one value to another, these lookup tables can be referenced from a <xwalk:field-source> element in a field map, using the value-map attribute.

Value maps can be used to:

- convert between differing sets of options in source and destination systems
- interpret free-format values into options
- remove or replace substrings
- convert to lower or upper case
- filter (or ignore) values.

Defining value maps

A value map is defined using a <xwalk:value-map> element, which has a name attribute to identify it. This contains a number of <xwalk:value-mapping> elements that define how individual strings or substrings should be mapped. An optional <xwalk:otherwise-mapping> element can be used to handle non-matching strings.

A simple value map

The value map below replaces the string ‘true’ with the string ‘Yes’ and the string ‘false’ with ’No’:

```
<xwalk:value-map name="convert-true-false-to-yes-no">
	<xwalk:value-mapping from="true"			to="Yes" />
	<xwalk:value-mapping from="false"		to="No" />
</xwalk:value-map>
```

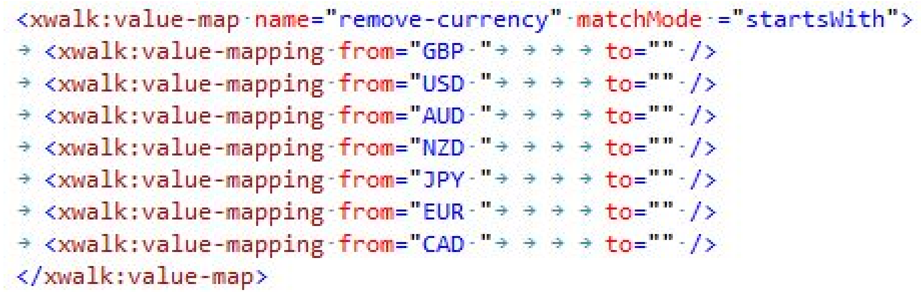
This value map above tests for an exact match in a string, but value maps can be used to match substrings. This is controlled using the matchMode attribute.

Using matchMode

The matchMode attribute of the <xwalk:value-map> element has the following options:

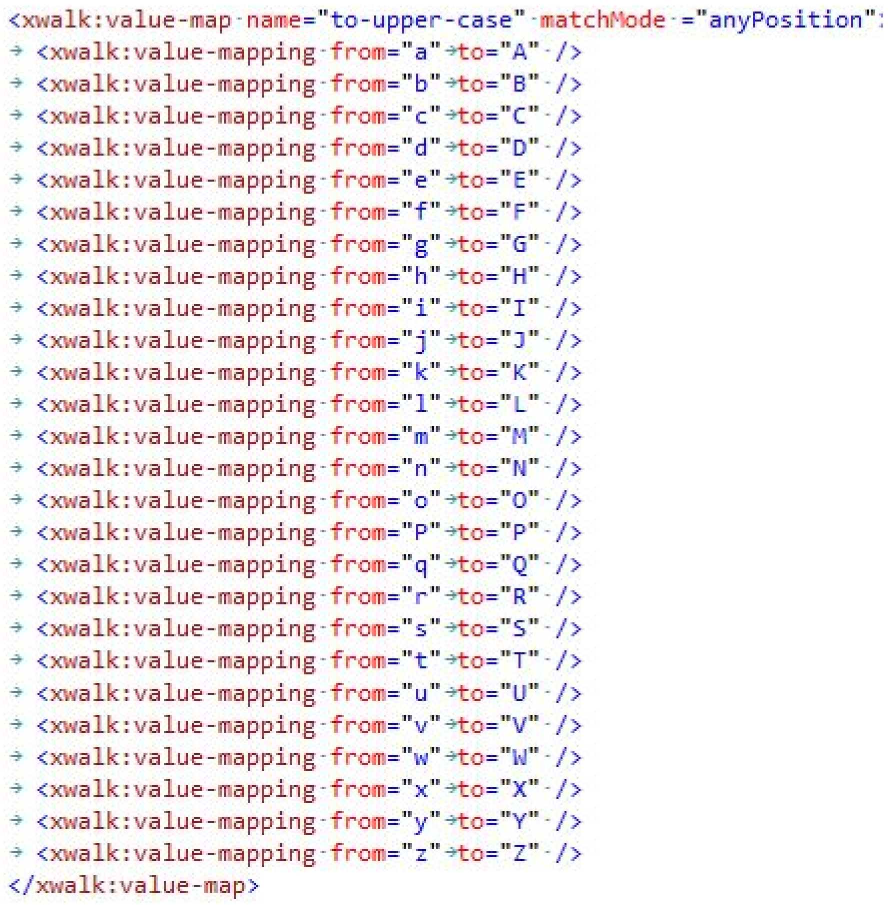
- full (this is the default value): an exact match is required
- startsWith - the string starts with the specified substring
- endsWith - the string ends with the specified substring
- anyPosition - any occurrence of the substring within the supplied string is replaced.

The following value map uses the matchMode=”startsWith“ to remove a set of currency codes from the start of a string:



```
<xwalk:value-map name="remove-currency" matchMode ="startsWith">
	<xwalk:value-mapping from="GBP "				to="" />
	<xwalk:value-mapping from="USD "				to="" />
	<xwalk:value-mapping from="AUD "				to="" />
	<xwalk:value-mapping from="NZD "				to="" />
	<xwalk:value-mapping from="JPY "				to="" />
	<xwalk:value-mapping from="EUR "				to="" />
	<xwalk:value-mapping from="CAD "				to="" />
</xwalk:value-map>
```

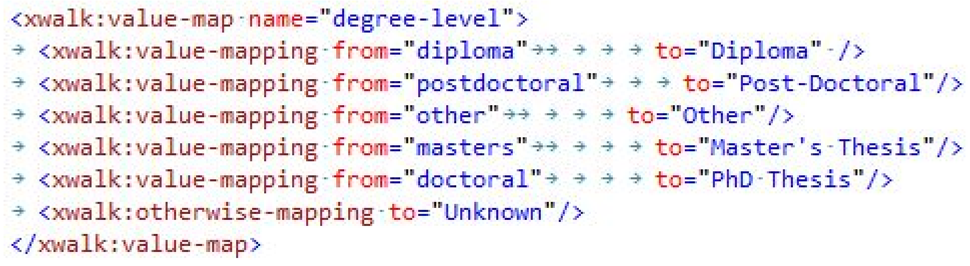
The following value map uses matchMode=”anyPosition” to convert the text string to uppercase:



```
<xwalk:value-map name="to-upper-case" matchMode ="anyPosition"
	<xwalk:value-mapping from="a"	to="A" />
	<xwalk:value-mapping from="b"	to="B" />
	<xwalk:value-mapping from="c"	to="C" />
	<xwalk:value-mapping from="d"	to="D" />
	<xwalk:value-mapping from="e"	to="E" />
	<xwalk:value-mapping from="f"	to="F" />
	<xwalk:value-mapping from="g"	to="G" />
	<xwalk:value-mapping from="h"	to="H" />
	<xwalk:value-mapping from="i"	to="I" />
	<xwalk:value-mapping from="j"	to="J" />
	<xwalk:value-mapping from="k"	to="K" />
	<xwalk:value-mapping from="l"	to="L" />
	<xwalk:value-mapping from="m"	to="M" />
	<xwalk:value-mapping from="n"	to="N" />
	<xwalk:value-mapping from="o"	to="O" />
	<xwalk:value-mapping from="P"	to="P" />
	<xwalk:value-mapping from="q"	to="Q" />
	<xwalk:value-mapping from="r"	to="R" />
	<xwalk:value-mapping from="s"	to="S" />
	<xwalk:value-mapping from="t"	to="T" />
	<xwalk:value-mapping from="u"	to="U" />
	<xwalk:value-mapping from="v"	to="V" />
	<xwalk:value-mapping from="w"	to="W" />
	<xwalk:value-mapping from="x"	to="X" />
	<xwalk:value-mapping from="y"	to="Y" />
	<xwalk:value-mapping from="z"	to="Z" />
</xwalk:value-map>
```

Converting between sets of options

A value map can be used to convert between different sets of options in source and destination systems.

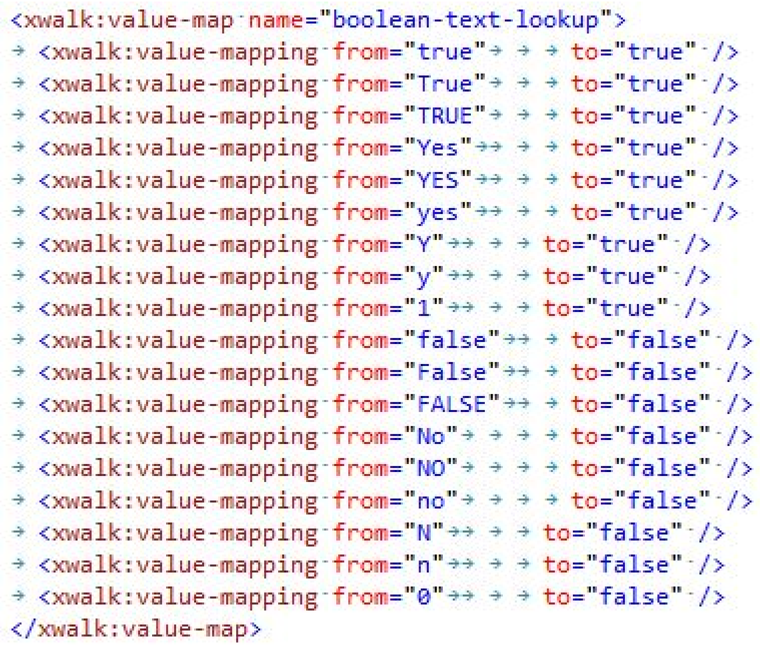


```
<xwalk:value-map name="degree-level">
	<xwalk:value-mapping from="diploma"			to="Diploma" />
	<xwalk:value-mapping from="postdoctoral"		to="Post-Doctoral"/>
	<xwalk:value-mapping from="other"			to="Other"/>
	<xwalk:value-mapping from="masters"			to="Master's Thesis"/>
	<xwalk:value-mapping from="doctoral"			to="PhD Thesis"/>
	<xwalk:otherwise-mapping to="Unknown"/>
</xwalk:value-map>
```

Note that the <xwalk:otherwise-mapping> element is used to catch values that do not match any of the previous from attribute values and assign a value of ‘Unknown’.

Interpreting free-format values

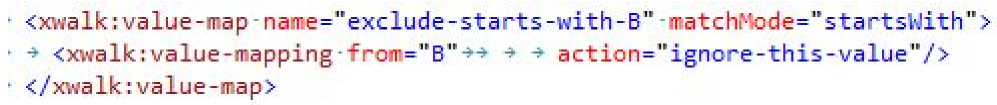
Some source systems may use free-format fields which need to be interpreted before they can be use in the destination system. The value map below shows how a free format field could be interpreted into a boolean value:



```
<xwalk:value-map name="boolean-text-lookup">
	<xwalk:value-mapping from="true"		to="true" />
	<xwalk:value-mapping from="True"		to="true" />
	<xwalk:value-mapping from="TRUE"		to="true" />
	<xwalk:value-mapping from="Yes"		to="true" />
	<xwalk:value-mapping from="YES"		to="true" />
	<xwalk:value-mapping from="yes"		to="true" />
	<xwalk:value-mapping from="Y"		to="true" />
	<xwalk:value-mapping from="y"		to="true" />
	<xwalk:value-mapping from="1"		to="true" />
	<xwalk:value-mapping from="false"		to="false" />
	<xwalk:value-mapping from="False"		to="false" />
	<xwalk:value-mapping from="FALSE"		to="false" />
	<xwalk:value-mapping from="No"		to="false" />
	<xwalk:value-mapping from="NO"		to="false" />
	<xwalk:value-mapping from="no"		to="false" />
	<xwalk:value-mapping from="N"		to="false" />
	<xwalk:value-mapping from="n"		to="false" />
	<xwalk:value-mapping from="0"		to="false" />
</xwalk:value-map>
```

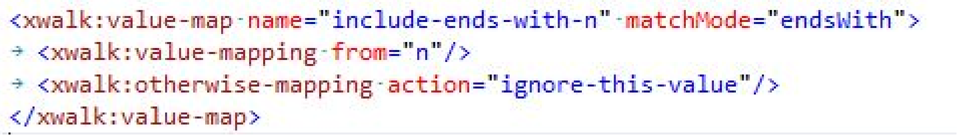
Filtering values

A value map can be used to filter or ignore values. The optional ‘action’ attribute of a value-mapping element can specify that when a match occurs, the matching value should be ignored. If no ‘action’ attribute is present (or the attribute value is “continue”) no filtering occurs.
The value map below excludes any value which begins with ‘B’:



```
<xwalk:value-map name="exclude-starts-with-B" matchMode="startsWith">
    <xwalk:value-mapping from="B"        action="ignore-this-value"/>
</xwalk:value-map>
```

Note that filtering can be applied using the xwalk:otherwise-mapping also. The value map below only includes values that end with ‘n’.



```
<xwalk:value-map name="include-ends-with-n" matchMode="endsWith">
    <xwalk:value-mapping from="n"/>
    <xwalk:otherwise-mapping action="ignore-this-value"/>
</xwalk:value-map>
```

This filtering capability can be used to handle fields that may have multiple values, many of which may not be relevant. An example might be dc.identifier.url which may be used to hold a variety of links (DOI, handle url etc), only one of which is relevant in the current context.

Logic expressions - <xwalk:if> and <xwalk:choose> Elements

There are times when you want the behaviour of your crosswalk to depend on the data values that are being processed. For instance with an inbound crosswalk the object type to be created and the field map to be used will depend on the item being processed. This requires the ability to define logic within the crosswalk map file.

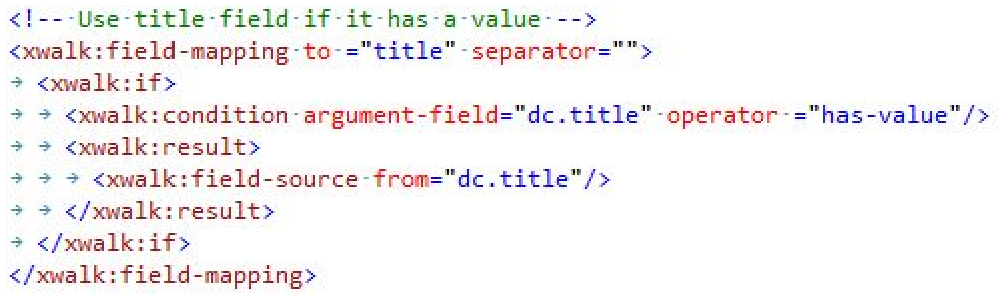
Crosswalk files support logic expressions with if statements (using an <xwalk:if> element) and choose statements (using the <xwalk:choose> element). Logic expressions can be used within the following crosswalk elements:

- <xwalk:field-mapping>
- <xwalk:field-map-selector>
- <xwalkin:object-type-selector>
- <xwalkout:collection-selector>.

Logic expressions use <xwalk:condition> elements to define one or more conditions to be evaluated and return the contents of <xwalk:result> elements.

A simple if expression

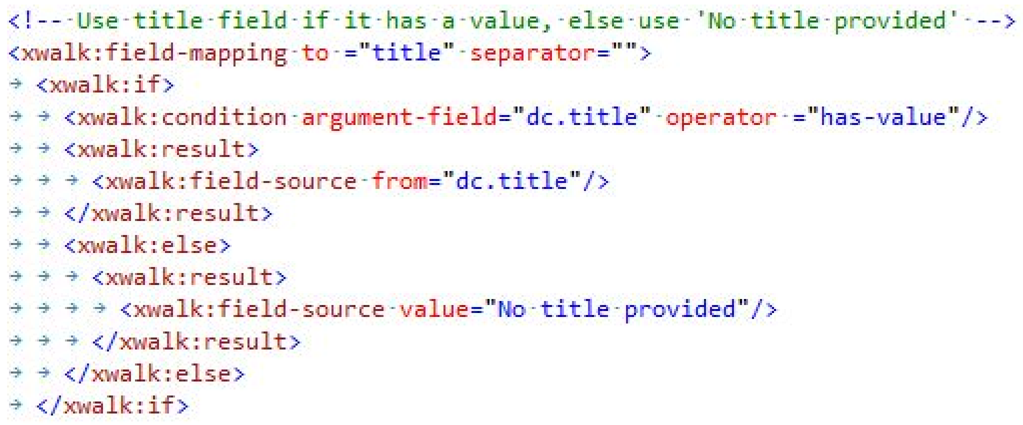
This if expression below sets the destination ‘title’ field to the value in ‘dc.title’, if dc.title has a value.



```
<!-- Use title field if it has a value -->
<xwalk:field-mapping to ="title" separator="">
    <xwalk:if>
        <xwalk:condition argument-field="dc.title" operator ="has-value"/>
        <xwalk:result>
            <xwalk:field-source from="dc.title"/>
        </xwalk:result>
    </xwalk:if>
</xwalk:field-mapping>
```

The condition to be evaluated (<xwalk:condition> element) inspects the value of the input field ‘dc.title’ (argument-field attribute) and checks whether it has a value (operator attribute). If the condition is true, the contents of the <xwalk:result> element will be used. The expression above does not include a <xwalk:else> element, so if the condition was not true no result would be available.

An extended version of this expression is shown below, which uses an <xwalk:else> element:



```
<!-- Use title field if it has a value, else use 'No title provided' -->
<xwalk:field-mapping to ="title" separator="">
    <xwalk:if>
        <xwalk:condition argument-field="dc.title" operator ="has-value"/>
        <xwalk:result>
            <xwalk:field-source from="dc.title"/>
        </xwalk:result>
        <xwalk:else>
            <xwalk:result>
                <xwalk:field-source value="No title provided"/>
            </xwalk:result>
        </xwalk:else>
    </xwalk:if>
```

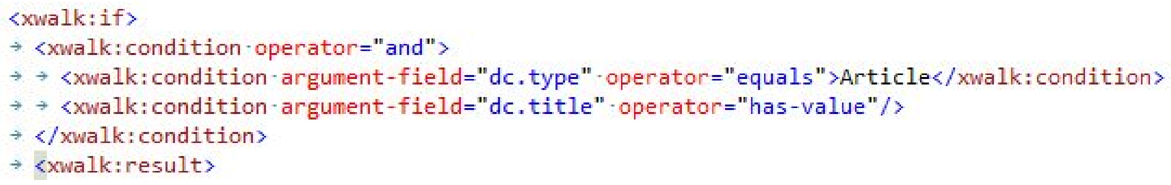
This sets the value of ‘title’ to ‘No title provided’ if dc.title has no value.

Condition Operators

<xwalk:condition> elements support the following operator attribute values:

- *equals*: performs a string comparison between values, returning true is they are the same, otherwise false
- *has-value*: true if the value string has one or more characters, otherwise false
- *not*: inverts the result of a contained condition
- *and*: ANDs the results of all contained conditions
- *or*: ORs the results of all contained conditions.

The *not*, *and* and *or* operators provide a means to build more complex conditional statements.



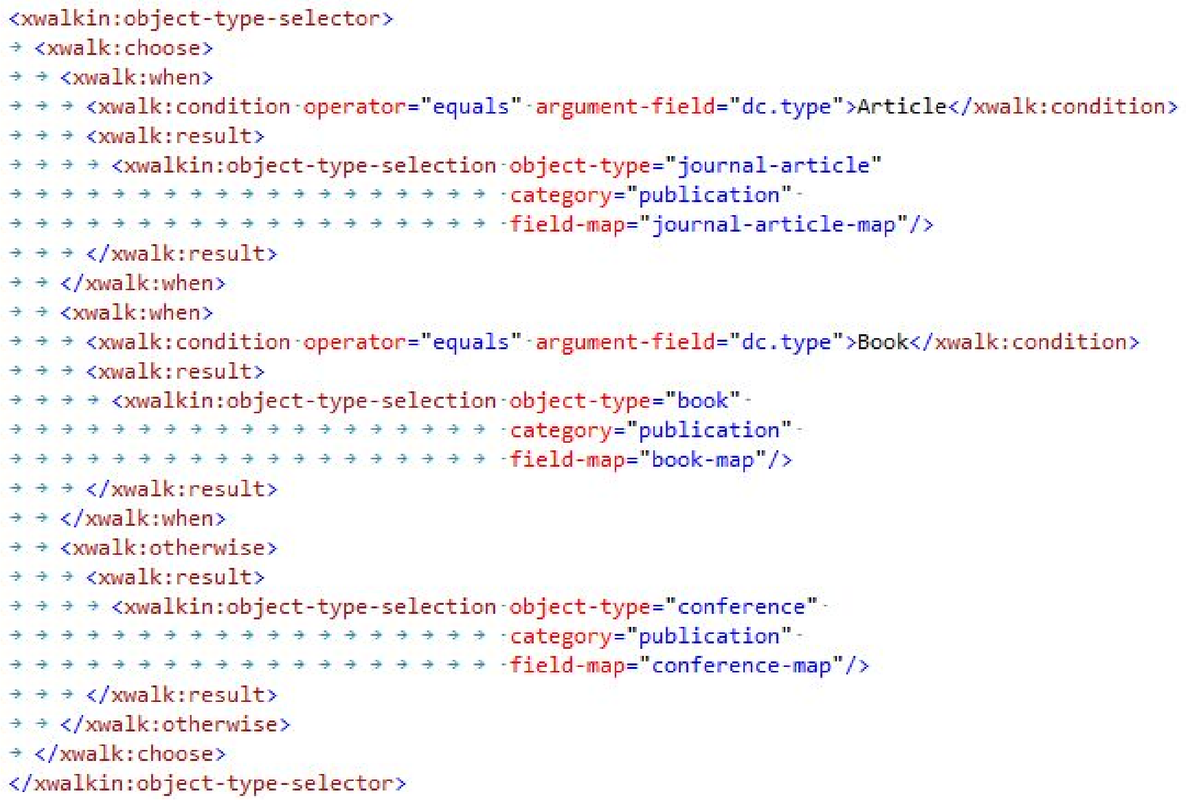
```
<xwalk:if>
    <xwalk:condition operator="and">
        <xwalk:condition argument-field="dc.type" operator="equals">Article</xwalk:condition>
        <xwalk:condition argument-field="dc.title" operator="has-value"/>
    </xwalk:condition>
    <xwalk:result>
```

The example above shows an <xwalk:condition> using the *and* operator. It returns true only if dc.type is ‘Article’ and dc.title has a value. Note the nested structure of the <xwalk:condition> elements.

A simple choose expression

Choose expressions can be used to evaluate a number of conditions in sequence, with the first valid condition being executed. The conditions to be evaluated are identified by <xwalk:when> elements, with a final optional <xwalk:otherwise> element which is executed if none of the previous conditions are true.

The choose expression below is used within a <xwalkin:object-type-selector> element to return a <xwalkin:object-type-selection> element that identifies the object type and field map to be used for an inbound crosswalk.



```
<xwalkin:object-type-selector>
	<xwalk:choose>
		<xwalk:when>
			<xwalk:condition operator="equals" argument-field="dc.type">Article</xwalk:condition>
			<xwalk:result>
				<xwalkin:object-type-selection object-type="journal-article"
																		category="publication"
																		field-map="journal-article-map"/>
			</xwalk:result>
		</xwalk:when>
		<xwalk:when>
			<xwalk:condition operator="equals" argument-field="dc.type">Book</xwalk:condition>
			<xwalk:result>
				<xwalkin:object-type-selection object-type="book"
																		category="publication"
																		field-map="book-map"/>
			</xwalk:result>
		</xwalk:when>
		<xwalk:otherwise>
			<xwalk:result>
				<xwalkin:object-type-selection object-type="conference"
																		category="publication"
																		field-map="conference-map"/>
			</xwalk:result>
		</xwalk:otherwise>
	</xwalk:choose>
</xwalkin:object-type-selector>
```

The choose expression uses the same <xwalk:condition> and <xwalk:result> elements that are used by the if expression. The use of multiple <xwalk:when> elements makes it a more flexible expression than the if expression.

More complex logic expressions

More complex logic expressions may be built up using if and choose statements by:

- Creating more complex condition statements
- Using if and choose expressions within a <xwalk:result> element.

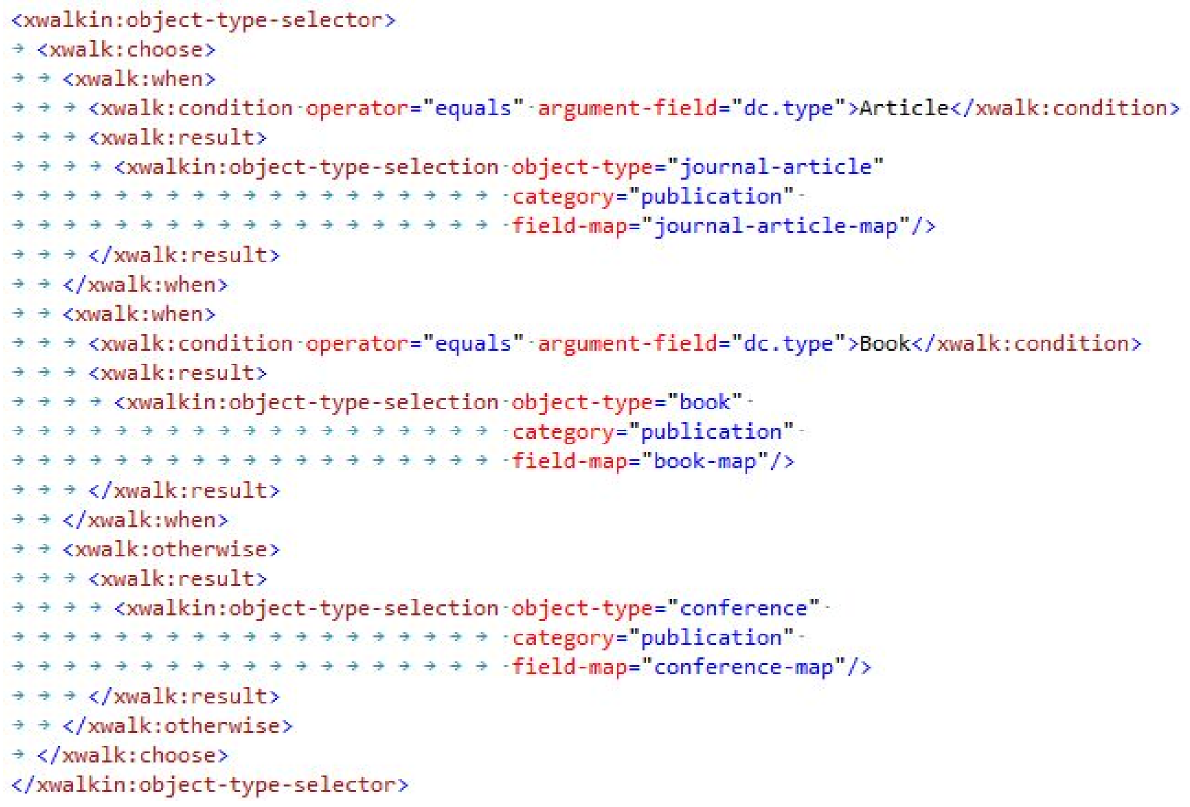
The sample below shows a <xwalk:field-mapping> that makes extensive use of <xwalk:if> expressions:

```
<xwalk:field-mapping to="dc.citation2" separator="">
	<xwalk:if>
		<xwalk:condition argument-field="title" operator="has-value"/>
		<xwalk:result>
			<!-- Note use of concatenate-with-separator attribute instead of suffix -->
			<xwalk:field-source from="authors" format="person:lastname-initials" concatenate-with-separator=", " />
			<xwalk:field-source from="publication-date" format="date:Y" prefix=" (" suffix="). " />
			<xwalk:field-source from="title" prefix=" " suffix=". " />
			<xwalk:field-source value="In " />
			<xwalk:if>
				<!-- Only show Editors if they are present -->
				<xwalk:condition argument-field="editors" operator="has-value"/>
				<xwalk:result>
					<xwalk:field-source from="editors" concatenate-with-separator=", " />
					<xwalk:field-source value=" (Ed.), " />
				</xwalk:result>
			</xwalk:if>
			<xwalk:field-source from="parent-title" suffix=". " />
			<xwalk:if>
				<!-- Only show (edition, pagination) if at least one exists -->
				<xwalk:condition operator="or">
					<xwalk:condition argument-field="edition" operator="has-value"/>
					<xwalk:condition argument-field="pagination" operator="has-value"/>
				</xwalk:condition>
				<xwalk:result>
					<xwalk:field-source value=" (" />
					<xwalk:field-source from="edition" />
					<xwalk:field-source from="pagination" />
					<xwalk:field-source value=") " />
				</xwalk:result>
			</xwalk:if>
			<xwalk:if>
				<xwalk:condition argument-field="place-of-publication" operator="has-value"/>
				<xwalk:result>
					<xwalk:field-source from="place-of-publication"/>
					<xwalk:if>
						<!-- Only show ": " if both place-of-publication and publisher exist -->
						<xwalk:condition argument-field ="publisher" operator ="has-value"/>
						<xwalk:result>
							<xwalk:field-source value=": "/>
						</xwalk:result>
					</xwalk:if>
				</xwalk:result>
			</xwalk:if>
			<xwalk:field-source from="publisher" />
			<xwalk:field-source value=". " />
		</xwalk:result>
	</xwalk:if>
</xwalk:field-mapping>
```

The <xwalkin:object-type-selector> Element

Each inbound crosswalk must be able to select the object type and field map for the item being crosswalked. Typically these will depend on the data item being processed, and will require use of logic expressions to specify the desired behaviour. These logic expressions should return a <xwalkin:object-type-selection> element (see Logic expressions for more information).

This is performed using the <xwalkin:object-type-selector> element. An example is shown below:

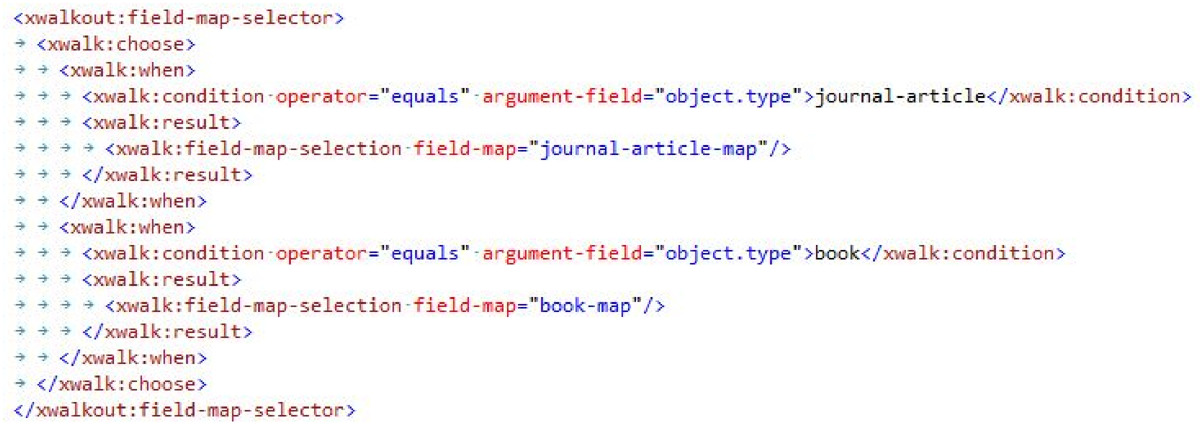


```
<xwalkin:object-type-selector>
    <xwalk:choose>
        <xwalk:when>
            <xwalk:condition operator="equals" argument-field="dc.type">Article</xwalk:condition>
            <xwalk:result>
                <xwalkin:object-type-selection object-type="journal-article"
                                               category="publication"
                                               field-map="journal-article-map"/>
            </xwalk:result>
        </xwalk:when>
        <xwalk:when>
            <xwalk:condition operator="equals" argument-field="dc.type">Book</xwalk:condition>
            <xwalk:result>
                <xwalkin:object-type-selection object-type="book"
                                               category="publication"
                                               field-map="book-map"/>
            </xwalk:result>
        </xwalk:when>
        <xwalk:otherwise>
            <xwalk:result>
                <xwalkin:object-type-selection object-type="conference"
                                               category="publication"
                                               field-map="conference-map"/>
            </xwalk:result>
        </xwalk:otherwise>
    </xwalk:choose>
</xwalkin:object-type-selector>
```

Three different logic conditions have been identified by the <xwalk:choose> expression, with each returning a different <xwalkin:object-type-selection> element. These are what specify the object type and field map to be used (note that the category attribute should always be 'publication'.

The <xwalkout:field-map-selector> Element

Each outbound crosswalk must specify the field map to be used for the item being crosswalked. Typically these will depend on the data item being processed, and will require use of logic expressions to specify the desired behaviour. These logic expressions should return a <xwalk:field-map-selection> element (see Logic expressions for more information). This is performed using the <xwalkout:field-map-selector> element. An example is shown below:



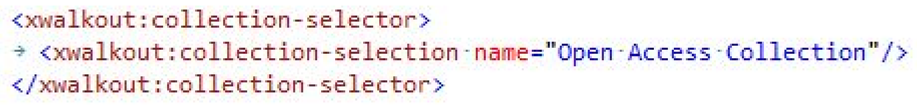
```
<xwalkout:field-map-selector>
	<xwalk:choose>
		<xwalk:when>
			<xwalk:condition operator="equals" argument-field="object.type">journal-article</xwalk:condition>
			<xwalk:result>
				<xwalk:field-map-selection field-map="journal-article-map"/>
			</xwalk:result>
		</xwalk:when>
		<xwalk:when>
			<xwalk:condition operator="equals" argument-field="object.type">book</xwalk:condition>
			<xwalk:result>
				<xwalk:field-map-selection field-map="book-map"/>
			</xwalk:result>
		</xwalk:when>
	</xwalk:choose>
</xwalkout:field-map-selector>
```

This example uses a choose expression to return different <xwalk:field-map-selection> elements if the object type is a journal article or a book. If the object-type is not one of the ones specified no element is returned and the item being processed will not be crosswalked.

The <xwalkout:collection-selector> Element

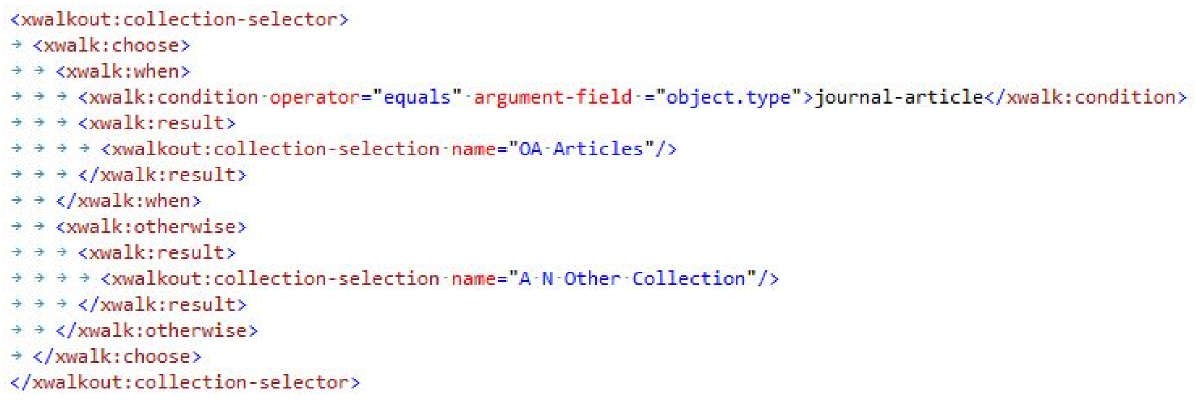
When an item is being crosswalked out of Elements into a repository we need to identify the destination collection that the item should go into. This is done using the <xwalkout:collection-selector> element.

If all items are to be put into the same collection, this can be defined simply:



```
<xwalkout:collection-selector>
    <xwalkout:collection-selection name="Open Access Collection"/>
</xwalkout:collection-selector>
```

However, if multiple collections may be used as the destination, logic will be needed to select the appropriate one for the item being crosswalked. The example below puts journal articles into the collection named ‘OA Articles’, with all other items going to ‘A N Other Collection’.



```
<xwalkout:collection-selector>
    <xwalk:choose>
        <xwalk:when>
            <xwalk:condition operator="equals" argument-field ="object.type">journal-article</xwalk:condition>
            <xwalk:result>
                <xwalkout:collection-selection name="OA Articles"/>
            </xwalk:result>
        </xwalk:when>
        <xwalk:otherwise>
            <xwalk:result>
                <xwalkout:collection-selection name="A N Other Collection"/>
            </xwalk:result>
        </xwalk:otherwise>
    </xwalk:choose>
</xwalkout:collection-selector>
```