
# Repository Tools 2
# Defining Crosswalks Guide

Version 1.8

Symplectic

May 22, 2020

# Contents

# 1 Introduction

This document describes how to define crosswalks that map data between your repository and Elements, using Repository Tools 2. It is designed primarily for technical staff to enable them to create and amend crosswalk map files.

There are two types of crosswalk, each defined by a separate file:

- a **Harvest** crosswalk (also called an inbound crosswalk). This defines a map from item data in your repository to publication data in Elements.
- a **Deposit** crosswalk (also called an outbound crosswalk). This defines a map from publication data in Elements to item data in your repository.

The files are structured XML that conform to an XML schema. There are three schemas – one for harvest, one for deposit, and a common schema that applies to both crosswalks. You can download these schema files from Elements, by navigating to:

**System Admin** → **Data** → **Data Source Management**

and clicking the data source which corresponds to your repository. Then scroll down to 'Crosswalk Map Files', expand the 'Schemas' section and click 'Download Schemas'.

Sample crosswalk map files are provided with Elements; however, Elements administrators will usually 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. For an overview of Repository Tools 2 and the crosswalking process, see the following articles on the Symplectic support site:

- A Functional Overview
- Repository as a Data Source
- Enabling Deposit via Elements
- Updating and Testing Crosswalks

## 1.1 Components of a harvest crosswalk map file

The structure of a harvest crosswalk map is as follows. Each of the top-level elements is discussed in its own section of the documentation.

```
<!-- Root element of the harvest crosswalk map -->
<xwalkin:consolidated-maps
↪  xmlns:xwalk="http://www.symplectic.co.uk/elements/xwalkcommon"
↪  xmlns:xwalkin="http://www.symplectic.co.uk/elements/xwalkin">

  <xwalk:parameters>
    <!-- Optional element. Contains <xwalk:parameter> elements, which are used to
    ↪  adjust some aspects of the crosswalk behaviour. -->
  </xwalk:parameters>

  <xwalk:elements-metadata>
    <!-- Optional element. Provides information about Elements' metadata structure.
    ↪  In normal operation, Elements will automatically create this. -->
  </xwalk:elements-metadata>

  <xwalk:field-maps>
    <!-- Required element. Contains <xwalk:field-map> elements, each of which define
    ↪  a set of data mappings from the source repository system to Elements. -->
  </xwalk:field-maps>

  <xwalk:value-maps>
    <!-- Optional element. Contains <xwalk:value-map> elements, which provide
    ↪  functionality to modify and filter data. -->
  </xwalk:value-maps>

  <xwalkin:object-type-selector>
    <!-- Required element. Selects the object type of the Elements object to be
    ↪  outputted by the crosswalk (e.g journal-article, book), along with the
    ↪  specification of which <xwalk:field-map> element should be used to crosswalk
    ↪  the data. -->
  </xwalkin:object-type-selector>

</xwalkin:consolidated-maps>
```

**Note:** the `<xwalk:elements-metadata>` element, containing details of the set of Elements fields and their data-types, is not usually required: Elements will automatically create the element and insert it into the map file when a harvest occurs. If the crosswalks are being run outside of the Elements environment – e.g. if manually invoking an XSL transform – this element will be required.

## 1.2 Components of a deposit crosswalk map file

The structure of a deposit crosswalk map is as follows. Each of the top-level elements is discussed in its own section of the documentation.

```
<!-- Root element of the deposit crosswalk map -->
<xwalkout:consolidated-maps
↪  xmlns:xwalk="http://www.symplectic.co.uk/elements/xwalkcommon"
↪  xmlns:xwalkout="http://www.symplectic.co.uk/elements/xwalkout">

  <xwalk:parameters>
    <!-- Optional element. Contains <xwalk:parameter> elements, which are used to
    ↪  adjust some aspects of the crosswalk behaviour. -->
  </xwalk:parameters>

  <xwalk:elements-metadata>
    <!-- Optional element. Provides information about Elements' metadata structure.
    ↪  In normal operation, Elements will automatically create this. -->
  </xwalk:elements-metadata>

  <xwalk:field-maps>
    <!-- Required element. Contains <xwalk:field-map> elements, each of which define
    ↪  a set of data mappings from Elements to the destination repository system.
    ↪  -->
  </xwalk:field-maps>

  <xwalk:file-metadata-map>
    <!-- Optional element. Defines the file-level metadata mapping to the destination
    ↪  repository system. -->
  </xwalk:file-metadata-map>

  <xwalk:value-maps>
    <!-- Optional element. Contains <xwalk:value-map> elements, which provide
    ↪  functionality to modify and filter data. -->
  </xwalk:value-maps>

  <xwalk:field-authority-lists>
    <!-- Optional element. Contains <xwalk:field-authority-list> elements, which
    ↪  provide functionality to choose which metadata fields at the repository to
    ↪  update with automated metadata push. -->
  </xwalk:field-authority-lists>

  <xwalkout:field-map-selector>
    <!-- Required element. Selects which <xwalk:field-map> element should be used to
    ↪  crosswalk the data. -->
  </xwalkout:field-map-selector>

  <xwalkout:collection-selector>
    <!-- Optional element. Required if the destination system has a concept of
    ↪  collections (e.g. DSpace); should be omitted otherwise. This element
    ↪  specifies which collection the item should be deposited to. -->
```

```
  </xwalkout:collection-selector>
</xwalkout:consolidated-maps>
```

**Note:** the `<xwalk:elements-metadata>` element, containing details of the set of Elements fields and their data-types, is not usually required: Elements will automatically create the element and insert it into the map file when a deposit occurs. If the crosswalks are being run outside of the Elements environment – e.g. if manually invoking an XSL transform – this element will be required.

# 2 The <xwalk:field-maps> element

## 2.1 Introduction

The <xwalk:field-maps> element contains the specification for how data should be mapped from the source system to the destination system. The element must contain one or more <xwalk:field-map> elements, each of which specify a number of field-mapping instructions. Each <xwalk:field-map> element must have a unique name, specified with the name attribute. For each crosswalked item, a single field map is used.

An simple example of a <xwalk:field-maps> element is shown below:

```
<xwalk:field-maps>

  <xwalk:field-map name="article-map">
    <!-- Contains the data mapping appropriate for articles. -->
  </xwalk:field-map>

  <xwalk:field-map name="book-map">
    <!-- Contains the data mapping appropriate for books. -->
  </xwalk:field-map>

</xwalk:field-maps>
```

## 2.2 The <xwalk:field-mapping> element

The <xwalk:field-mapping> element describes a mapping to a destination field. Each <xwalk:field-map> element can contain multiple <xwalk:field-mapping> elements. Every <xwalk:field-mapping> must specify a destination field using the to attribute, and should contain <xwalk:field-source> elements, which are used to select data from the source system. The from attribute of the <xwalk:field-source> element can be used to specify a field in the source system. For example:

```
<xwalk:field-map name="example-field-map">

  <xwalk:field-mapping to="destination-field">
    <xwalk:field-source from="source-field"/>
  </xwalk:field-mapping>

</xwalk:field-map>
```

This example specifies that the value of the *source-field* in the source system is to be mapped to the *destination-field* in the destination system.

A field-mapping can be used to concatenate multiple values, which can be useful if there are multiple field-sources, or if a field-source selects data from a multivalued field. To perform this concatenation, the `separator` attribute can be used:

```
<xwalk:field-mapping to="destination-field" separator="; ">
  <xwalk:field-source from="source-field-1" />
  <xwalk:field-source from="source-field-2" />
</xwalk:field-mapping>
```

The above field-mapping will concatenate the values of *source-field-1* and *source-field-2*, separated by a semicolon, the result of which will be mapped to *destination-field*.

## 2.3 Field identifiers

The `to` and `from` attributes of the `<xwalk:field-mapping>` and `<xwalk:field-source>` elements respectively use field identifiers to specify fields in the source and destination systems. Depending on the crosswalk direction, these identifiers can refer to either Elements fields or repository fields.

### 2.3.1 Elements field identifiers

When specifying an Elements field, use the underlying Elements field name. These can be found in Elements, by navigating to:

**Module Admin → Publications → Underlying Fields**

Note that crosswalks always use the underlying field names (e.g. *title*, *abstract*, *authors* etc) – common to all publications in Elements – not the display names, which can change from one publication type to another. Any Elements custom fields can also be referenced – these field names always begin with "c-".

The following is an example of a field-mapping in a harvest crosswalk, specifying as its destination the Elements field *abstract*:

```
<xwalk:field-mapping to="abstract">
```

The following is an example of a field-source in a deposit crosswalk, specifying as its source the Elements field *abstract*:

```
<xwalk:field-source from="abstract" />
```

There is also a set of special field identifiers which can reference other Elements properties to be used as source data in deposit crosswalks; for further details of these see Section 6.1.

Elements fields have an associated data type. Many data types are simple scalar fields (e.g. text, integer, boolean, ISSN, etc); other data types are more complex. The more complex types can be categorised as follows:

| Field type | Description |
|---|---|
| **Simple multivalued** | contains a list of text or other scalar type values. |
| **Compound** | contains multiple labelled components which themselves can be simple or compound (e.g date type fields have day, month and year components). |
| **Compound multivalued** | contains a list of compound type values (e.g. field of type address-list contain multiple addresses, which are each compound). |

### 2.3.2 Repository field identifiers

Each repository has its own set of fields. The set of repository field identifiers will therefore vary according to the specific repository platform, and additionally may depend on any repository field customisation in place. The repository fields are usually referred to by their underlying name.

For DSpace, the field identifiers are the fully qualified field names. For a Dublin Core field, this will be something like *dc.description.title*. For EPrints, the field identifiers will be something like *book_title*.

The following is an example of a `<xwalk:field-mapping>` element in a deposit crosswalk, specifying the destination repository field *dc.title*:

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

The following is an example of a `<xwalk:field-source>` element in a harvest crosswalk, specifying the source repository field *dc.title*.

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

Some repositories (e.g. DSpace) only contain text fields; others (e.g. EPrints, Figshare for Institutions) can contain compound or multivalued fields types, in the same way as Elements.

## 2.4 Manipulating text

The `<xwalk:field-source>` element has the capability to manipulate text values using various attributes. For example, the `prefix` attribute can be used to prefix the source field value with a fixed text value. In the example below, if the *dc.description.abstract* field has a value of "123", then the value "Abstract: 123" will be mapped to the "abstract" field.

```
<xwalk:field-mapping to="abstract">
  <xwalk:field-source from="dc.description.abstract" prefix="Abstract: " />
</xwalk:field-mapping>
```

All `<xwalk:field-source>` text manipulation attributes are described in the following table. These attributes can be used when manipulating simple or simple multivalued fields.

| Attribute | Description |
|---|---|
| `value` | Selects the attribute value verbatim (ignoring any other value selected in the `from` attribute). |
| `split-using-delimiter` | Splits the source value string(s) at every occurrence of the attribute value. This will produce a multivalued result set. |
| `concatenate-with-separator` | Concatenates multiple source values or a multivalued field into a single string value, separating each with the attribute value. |
| `value-map` | Applies to the source value(s) the value-map with name equal to the attribute value. Value-maps are described in full details in Section 3. |
| `prefix` | Adds the attribute value before the source value(s). |
| `suffix` | Adds the attribute value after the source value(s). |
| `distinct` | Inspects multiple source value strings and ensures there are no duplicates. |

**Note:** a field-source can specify multiple text manipulation attributes, and those manipulation operations are performed in the order that the attributes are listed in the table. For example, if both a value-map and a prefix are specified, the value-map will be applied first, followed by the prefix.

## 2.5 Applying multiple processing operations to text

The field-source text processing described in the previous section should be sufficient for most field-mappings. However, there may be cases when multiple rounds of processing are required in order to achieve the desired result. For example, if two value-maps are to be applied in succession; or if two processing actions should be applied in an non-standard order, such as applying a prefix followed by a value-map.

To achieve this, a `<xwalk:field-source>` element specifying the second processing operation should wrap the `<xwalk:field-source>` element which performs the first processing operation. For example, to apply `value-map-1` and then `value-map-2` to the value of the field *source-field*, the following structure can be used:

```
<xwalk:field-mapping to="destination-field">
  <xwalk:field-source value-map="value-map-2">
    <xwalk:field-source from="source-field" value-map="value-map-1" />
```

```
  </xwalk:field-source>
</xwalk:field-mapping>
```

## 2.6 Field-mappings involving compound & compound multivalued Elements fields

When constructing field-mappings either to or from compound or compound multivalued Elements fields (in a harvest or deposit crosswalk, respectively), it may not be sufficient to just use the text manipulation functionality described in the previous sections.

To enable flexible crosswalking of these fields, two additional attributes on the `<xwalk:field-source>` element can be used: `data-part` and `format`. The term 'data-part' refers to the components of a compound Elements field. For example, an Elements date field has data-parts `date:day`, `date:month` and `date:year`. Each compound Elements field is made up of a set of data-parts and these parts can be combined into a string in a number of ways known as formats.

The behaviour of these attributes depends on whether the crosswalk is for deposit or harvest. The following tables provide a summary of the behaviour; for full details and examples, see the sections on harvest and deposit specific compound field handling.

| Harvest | |
|---|---|
| **Attribute** | **Description** |
| `data-part` | Specifies which component of an Elements compound field to target with a piece of repository data. |
| `format` | Specifies how to parse a single string value into the relevant data-parts of an Elements compound field. |

| Deposit | |
|---|---|
| **Attribute** | **Description** |
| `data-part` | Specifies which component of an Elements compound to select data from to pass to a repository. |
| `format` | Specifies how to combine the data-parts of an Elements compound field into a single string to pass to a repository. |

See Appendix A (harvest) and Appendix B (deposit) for a full list of options for both of these attributes, for each compound field type.

## 2.7 Conditional logic

There are times when it may be desirable to use different field-sources depending on certain conditions. To achieve this, conditional logic can be used within the `<xwalk:field-mapping>` element. Appendix D: Logic expressions describes how these can be written.

## 2.8 Reusing field-maps

When building field-maps there may be several field-mappings repeated in each field-map. In order to avoid this repetition and reduce the opportunity for errors, any field-map can be referenced from within another using the `<xwalk:include-field-map>` element. This allows a single field-map to be defined for common fields and reused in many places. For example, the following shows an example of a common fields map, which maps the *title*, *abstract* and *date* fields. This is then included in the field-map for journal articles below it.

```
<!-- A field map for common field mappings -->
<xwalk:field-map name="common-fields-map">
  <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="publication-date">
    <xwalk:field-source from="dc.date.available" />
  </xwalk:field-mapping>
</xwalk:field-map>

<!-- A field map which includes the common mappings -->
<xwalk:field-map name="journal-article-map">

  <!-- Import the contents of the "common-fields-map" above -->
  <xwalk:include-field-map name="common-fields-map"/>

  <!-- Include additional field mappings -->
  <xwalk:field-mapping to="publisher">
    <xwalk:field-source from="dc.publisher" suffix=" (from journal-article-map)"/>
  </xwalk:field-mapping>

</xwalk:field-map>
```

All the field-mappings in `common-field-map` will be used, as well as the explicit `<xwalk:field-mapping>` elements. If an explicit field-mapping has a `to` attribute which is the same as one defined in the included field-map, the explicit field-mapping will be used and the included one will be ignored. Note that multiple `<xwalk:include-field-map>` elements may be used, but they must all occur before any `<xwalk:field-mapping>` elements:

```
<xwalk:field-map name="journal-article-map">

  <!-- Include two other field-maps -->
  <xwalk:include-field-map name="common-fields-map" />
  <xwalk:include-field-map name="more-fields-map" />

  <!-- One field-mapping specific to this map -->
```

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

</xwalk:field-map>
```

It is possible to include a field-map, but exclude one or more field-mappings from the included field-map by using an `<xwalk:exclude-field-mapping>` element:

```
<xwalk:field-map name="journal-article-map">

  <!-- Includes all field-mappings from "common-fields-map" except the field-mapping
  ↪  to the "abstract" field -->
  <xwalk:include-field-map name="common-fields-map">
    <xwalk:exclude-field-mapping to="abstract" />
  </xwalk:include-field-map>

  <!-- One field-mapping specific to this field-map -->
  <xwalk:field-mapping to="publisher">
    <xwalk:field-source from="dc.publisher" />
  </xwalk:field-mapping>

</xwalk:field-map>
```

## 2.9 XPath

**Note:** Using XPath should be avoided if possible – it relies on an assumed knowledge of the underlying XML structure used by the crosswalk, which may change over time. Field-mappings which use XPath are not guaranteed to work between different Elements versions.

When selecting data from the source system using a field-source, field identifiers are typically used. In most situations this works fine, but there are times when this may not provide the desired flexibility or capability. For example, accessing data values that are not part of the normal set of metadata fields, such as the DSpace collection information.

If the `select-using` attribute of a `<xwalk:field-source>` element is set to `xpath`, then the `from` attribute will be evaluated as an XPath statement instead of a field identifier. For example, in DSpace, the collection information (not stored in metadata fields) could be accessed using the following:

```
<xwalk:field-mapping to="c-parent-collection-description">
  <xwalk:field-source from="/item/parentCollection/name" select-using="xpath" />
</xwalk:field-mapping>
```

# 3 The <xwalk:value-maps> element

This optional element is used to provide additional functionality for manipulating data selected using field-sources. Multiple <xwalk:value-map> elements may exist within the <xwalk:value-maps> element. Each has a name attribute that is used to identify it.

Each value-map can be thought of as a store of custom functionality that can be invoked from a <xwalk:field-source> element by setting its value-map attribute to the name of the desired value-map. The value-map will be applied to the results of the field-source. Note that if the field-source produces a multivalued result set, the value-map will be applied to each value in the set.

Typical uses of value-maps include:

- Converting between differing sets of options in source and destination systems
- Selecting specific substrings
- Removing or replacing specific substrings
- Converting to lower or upper case
- Filtering or ignoring certain values

## 3.1 The <xwalk:value-mapping> element

Each <xwalk:value-map> element must contain one or more <xwalk:value-mapping> elements. Each value-mapping must specify a pattern to match against using the from attribute, and may specify a replacement value using the to attribute.

By default, a value-mapping triggers a match when the input string is equal to the pattern. This behaviour is known as a "full" match; other behaviours can be specified, as described in Section 3.2. For each value-mapping which produces a match, the input string is replaced with the replacement.

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 can be invoked from a field-mapping such as the one below:

```
<xwalk:field-mapping to="eprints_field">
   <xwalk:field-source from="c-boolean-field"
   ↪  value-map="convert-true-false-to-yes-no" />
</xwalk:field-mapping>
```

The value of *c-boolean-field* in this example is either 'true' or 'false'; the resulting value which is mapped to the destination field *eprints_field* is either 'Yes' or 'No'.

## 3.2 Match modes

The `matchMode` attribute of the `<xwalk:value-map>` element can be used to control the conditions under which the value-mapping pattern is considered to match. The available options are:

| Option | Match condition |
|---|---|
| `full` | The input string is exactly equal to the supplied pattern (default). |
| `startsWith` | The input string starts with the supplied pattern. |
| `endsWith` | The input string ends with the supplied pattern. |
| `anyPosition` | The input string contains the supplied pattern as a substring, at any position (including the start and end positions). |
| `regex` | The pattern, evaluated as a regular expression, produces at least one match in the input string. See Section 3.2.2: `regex` match mode for full details. |

With some match modes it is possible for some value-mappings to produce multiple matches. In this case, all matching ranges of the input string are replaced with the replacement. For example, consider the following value-map and field-mapping:

```
<xwalk:value-map name="make-exciting" matchMode="anyPosition">
  <xwalk:value-mapping from="." to="!" />
</xwalk:value-map>
```

```
<xwalk:field-mapping to="c-excited-field">
  <xwalk:field-source value="Here. Are. Some. Words." value-map="make-exciting" />
  <!-- Outputs the text "Here! Are! Some! Words!" -->
</xwalk:field-mapping>
```

Each of the four matches found in the input string will be replaced with the replacement string, producing the result "Here! Are! Some! Words".

### 3.2.1 Multiple matching value-mappings

The previous section described how a value-mapping can match an input string multiples times. In addition, there can be multiple value-mappings which produce matches. In this case, every value-mapping which finds a match is applied. The value-mappings are applied in the order they are declared in the value-map. See the following example:

```
<xwalk:value-map name="switch-letters" matchMode="anyPosition">
  <xwalk:value-mapping from="a" to="b" />
  <xwalk:value-mapping from="bb" to="xx" />
</xwalk:value-map>
```

```
<xwalk:field-mapping to="sentence">
  <xwalk:field-source value="ab-cd-ad" value-map="switch-letters" />
  <!-- Outputs the text "xx-cd-bd" -->
</xwalk:field-mapping>
```

The first value-mapping produces two matches, replacing both occurrences of the letter “a” with the letter “b”. The second value-mapping, using the result of the first value-mapping as its input, produces one match, replacing the single occurrence of “bb” with “xx”.

### 3.2.2 `regex` match mode

The `regex` match mode allows regular expressions to be used in value-mappings, to achieve powerful text manipulation.

The following is a simple example which uses the regex pattern `\d` to match on all digit characters, and specifies a replacement string “9”. This value-map would therefore replace *all* number characters with the number “9”, for example mapping the input “ABC 123 DEF 567” to “ABC 999 DEF 999”.

```
<xwalk:value-map name="replace-numbers-with-nines" matchMode="regex">
  <xwalk:value-mapping from="\d" to="9" />
</xwalk:value-map>
```

Regex capture groups can be referenced in the `to` attribute using the text `$1` for the first capture group, `$2` for the second capture group, etc. This can allow for powerful operations to be performed by value-maps. For example, the following is a value-map which will prefix four digit numbers with the text “FoR: ”, but will not apply to any four-digit substrings of longer numbers:

```
<!-- Note that the XML encoding of the < and > characters must be used: &lt; and &gt;
↪  -->
<xwalk:value-map name="prefix-4-digit-numbers-with-for" matchMode="regex">

  <!-- The regex pattern contains a lookbehind for non-digit characters, a capture
  ↪  group for four-digit number strings, and a lookahead for non-digit characters.
  ↪  The replacement specifies the text "FoR: " followed by the value of the first
  ↪  capture group (i.e. the found four digit number) -->
  <xwalk:value-mapping from="(?&lt;=[^\d])(\d{4})(?=[^\d])" to="FoR: $1" />

</xwalk:value-map>
```

This value-map will map the input “There are 15000 publications with label 0101” to “There are 15000 publications with label FoR: 0101”.

The full capabilities of regex patterns are described on the Microsoft Regular Expression Quick Reference page.

### 3.3 The `<xwalk:otherwise-mapping>` element

A `<xwalk:value-map>` element can contain, as its final child element, a single `<xwalk:otherwise-mapping>` element. This behaves similarly to a value-mapping, but no pattern can be specified, and it will be applied only if none of the value-mappings produced a match. If a replacement is specified with the `to` attribute, then the whole input string is replaced.

For example, the following value-map could be used to adjust some publication type strings:

```
<xwalk:value-map name="adjust-pub-type">
  <xwalk:value-mapping from="article" to="Article" />
  <xwalk:value-mapping from="journal-article" to="Article" />
  <xwalk:value-mapping from="book" to="Book" />
  <xwalk:value-mapping from="conference" to="Conference" />
  <xwalk:value-mapping from="conference-proceeding" to="Conference" />
  <xwalk:otherwise-mapping to="Other" />
</xwalk:value-map>
```

In the above example, if none of the five value-mappings produced a match, the value-map will output the value “Other”.

### 3.4 Filtering values

A value-map can be used to filter or ignore values. The optional `action` attribute of a `<xwalk:value-mapping>` or `<xwalk:otherwise-mapping>` element can be used to specify actions to be taken when a match occurs. The available options are:

| Option | Behaviour |
|---|---|
| `continue` | Continue processing of the value-map (default) |
| `ignore-this-value` | Exit the value-map processing and discard the input value |

By specifying `ignore-this-value`, specific values of a multivalued results set can be excluded. If no `action` attribute is present, the default of `continue` is used, and 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>
```

By setting the `action` attribute on the `<xwalk:otherwise-mapping>`, and value-map can exclude values which do not match. For example, the value-map below excludes all values that do not end with “n”.

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

This filtering capability can be used to handle fields that may have multiple values, of which only some may be relevant. For example, the DSpace Dublin Core field *dc.identifier.url* may be used to hold a variety of links (DOI, handle URL, etc), for which only some of are relevant in a given context.

## 3.5 Filtering values based on data-part

When dealing with complex compound fields, “data-parts” are used to describe each component of data. The values of specific data-parts can be used in value-maps. For harvest-specific details of data-parts see Section 5.1.1; for deposit-specific details see Section 6.2.1.

The `data-part` attribute of a `<xwalk:value-map>` can be used to specify which component to use when evaluating a `<xwalk:value-mapping>`. This can allow value-maps to perform actions based on the value of one part of the data structure.

### 3.5.1 When harvesting from a repository

**Note:** this section is only relevant for repositories which can store compound data types (EPrints and Figshare for Institutions). As such, the value-maps used in this context are only applicable to complex nested field-sources. For more details on this functionality, see 5.1.1.

**Filtering based on an Elements data-part**

When harvesting data into an Elements field which has a compound field type (i.e. consisting of multiple components, such as a person-list field), the `data-part` attribute can be used to apply a value-map to the result of the mapping. For example, to map a series of text values into the person-list data structure, and then filter out entries with a given surname component, the following field-mapping and value-map can be used:

```
<xwalk:field-mapping to="authors">
  <xwalk:field-source value-map="no-greens">
    <xwalk:field-source from="dc.contributor.author" />
  </xwalk:field-source>
</xwalk:field-mapping>
```

```
<xwalk:value-map name="no-greens" data-part="person:lastname">

  <xwalk:value-mapping from="Green" action="ignore-this-value" />
  <!-- This otherwise-mapping is optional since "continue" is the default action -->
  <xwalk:otherwise-mapping action="continue" />

</xwalk:value-map>
```

#### Filtering based on other data

When harvesting from a compound field, it may be desirable to filter the source data based on some data which is not being mapped into the repository. For example, the EPrints field *related_url* is a multivalued field containing subfields *url* and *type*. We may want to harvest the *url* subfield, but only when the corresponding *type* subfield is equal to "author". To do so, the *type* subfield can be marked as containing a temporary data-part `temp:value1`, which is then used in the value-map. All `temp:` data-parts components are discarded after the field-source has been processed.

```
<xwalk:field-mapping to="author-url">

  <!-- The value-map "author-url-only" is applied to the result of the this
  ↪  field-source (i.e. after evaluating its inner field-sources). -->
  <xwalk:field-source from="related_url" value-map="author-url-only">

    <!-- The value of the "url" subfield is selected, and the value of the "type"
    ↪  subfield is harvested into a temporary data-part -->
    <xwalk:field-source from="url" />
    <xwalk:field-source from="type" data-part="temp:value1" />

  </xwalk:field-source>
</xwalk:field-mapping>
```

```
<!-- This value-map applies to the "temp:value1" data-part of the input data -->
<xwalk:value-map name="author-url-only" data-part="temp:value1">

  <!-- If the "temp:value1" data-part has a value other than "author", the item is
  ↪  discarded -->
  <xwalk:value-mapping from="author" action="continue" />
  <xwalk:otherwise-mapping action="ignore-this-value" />
</xwalk:value-map>
```

### 3.5.2 When depositing from Elements

When depositing data from Elements, a value-map can be applied to a specific data-part/component of the source data to transform or filter the data. The example below shows how to crosswalk

only labels which are associated with the *Fields of Research* scheme. The field-mapping uses the *object.labels* identifier to obtain all the labels associated with the object being deposited, and applies the *include-for-labels-only* value-map to each label.

```
<xwalk:field-mapping to="dc.labels.for">
  <xwalk:field-source from="object.labels" format="keyword:with-scheme"
  ↪  value-map="include-for-labels-only" />
</xwalk:field-mapping>
```

```
<xwalk:value-map name="include-for-labels-only" data-part="keyword:scheme">
  <xwalk:value-mapping from="for" action="continue" />
  <xwalk:otherwise-mapping action="ignore-this-value" />
</xwalk:value-map>
```

This *include-for-labels-only* value-map has a `data-part` attribute to specify that the matching is to be performed on the *keyword:scheme* data-part. If the scheme of the keyword is "for", the value continues to be processed; keywords of any other scheme are ignored.

## 3.6 Examples

The following are some examples of the sort of value-maps which are typically useful when crosswalking data between different formats.

### 3.6.1 General text manipulation

The following value-map replaces currency codes appearing at the start of the input string with the corresponding symbol:

```
<xwalk:value-map name="replace-currency-symbol" matchMode="startsWith">
  <xwalk:value-mapping from="GBP" to="£"/>
  <xwalk:value-mapping from="USD" to="$"/>
  <xwalk:value-mapping from="AUD" to="AU$"/>
  <xwalk:value-mapping from="NZD" to="NZ$"/>
  <xwalk:value-mapping from="JPY" to="¥"/>
  <xwalk:value-mapping from="EUR" to="€"/>
  <xwalk:value-mapping from="CAD" to="CA$"/>
</xwalk:value-map>
```

This value-map uses converts a 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" />
  <!-- ... etc ... -->
  <xwalk:value-mapping from="z" to="Z" />
</xwalk:value-map>
```

### 3.6.2 Converting between sets of options

A value-map can be used to map between fields in source and destination systems that have differing sets of options. For example:

```
<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'.

### 3.6.3 Interpreting free-format values

Some source systems may use free-format fields which need to be interpreted before they can be used in the destination system. The value map below shows how a free format field could be interpreted into a boolean value, matching (case-insensitively) the values "true", "yes", "y" or "1" for "true", and "false", "no", "n" or "0" for "false":

```
<xwalk:value-map name="boolean-text-lookup" matchMode="regex">
  <!-- The "(?i)" option enables case-insensitive matching -->
  <xwalk:value-mapping from="^(?i)(true|yes|y|1)$" to="true" />
  <xwalk:value-mapping from="^(?i)(false|no|n|0)$" to="false" />
  <xwalk:otherwise-mapping action="ignore-this-value" />
</xwalk:value-map>
```

The following value-map replaces a citation-like input string with an extracted "edition" string:

```
<xwalk:value-map name="edition-from-citation" matchMode="regex">
  <xwalk:value-mapping from="(?i)^.*(?:edition|ed)[^\w]*(\w+).*$" to="$1"/>
  <xwalk:otherwise-mapping action="ignore-this-value"/>
</xwalk:value-map>
```

This value-map will match if the input string contains the word “edition” or “ed”, optionally followed by any non-word characters (colon, full-stop, space, etc), followed by some text consisting of word characters, followed by any other text. The value-map will match the entire input, and use a capture group to capture the text following “edition” (or “ed”). The value-map will replace the matched portion (i.e. the entire string) with the captured text (i.e. the text following "edition").

The following table shows some example inputs and outputs of this value-map:

| Input text | Output text |
|---|---|
| Example book, edition 11 page 17-32 | 11 |
| Example article, pg. 12, ed. 4, vol. 12 | 4 |
| Example conference; edition: 2; Volume: 1. | 2 |
| Example chapter. Edition XII. Page 17. | XII |
| Example photograph. Date: July 2017. | None – value ignored |

# 4 Other elements

## 4.1 The <xwalk:parameters> element

This optional element can be used to set parameters that control the behaviour of a crosswalk. Parameters are set using a <xwalk:parameter> element with a name attribute which specifies the parameter and a value attribute which specifies the parameter value. For example:

```
<xwalk:parameters>
  <xwalk:parameter name="default-currency" value="money:gbp" />
</xwalk:parameters>
```

### 4.1.1 Common parameters

The following table contains the parameters that apply to both deposit and harvest crosswalks. The default value is the value that takes affect if the parameter is not specified inside the <xwalk:parameters> element.

| Name | Default value |
|---|---|
| default-currency | USD |
| default-date-format | date:YYYY-MM-DD |
| default-date-delimiter | - |
| default-person-format | person:lastname-initials |
| default-person-format-delimiter | space |
| default-pagination-format | pagination:start-end |
| default-pagination-delimiter | - |
| default-funding-acknowledgements-grant-format | funding-acknowledgements -grant:org-id |
| default-funding-acknowledgements-grant-delimiter | \| |
| use-leading-zeroes-for-dates | false |

For a list of allowed values for the format parameters, see Appendix A and Appendix B.

### 4.1.2 Harvest crosswalk parameters

The following parameters are applicable only to harvest crosswalks. If any parameter is missing then its default value applies:

**xwalkin-without-files**
Controls whether information about files is crosswalked. By default, this is set to false. If this parameter is present and set to true, then file information will not be crosswalked.

**xwalkin-without-file-urls**
Note: this parameter only applies to harvests from DSpace. This parameter controls whether file URLs should be crosswalked. By default, this is set to `false`. If this parameter is present and set to `true`, then file URLs will not be crosswalked.

**dspace-xwalkin-public-access-group-ids**
This parameter enables the crosswalk to interpret the embargo status of files. This should be set to a comma-separated-list of IDs of DSpace Groups which are considered "public". For example, if there are two DSpace "Anonymous" groups, with IDs 123 and 456, this parameter should be set to "123,456". If not supplied, a default value of "0" is used.

This parameter is particularly importing when harvesting from DSpace 6, as the default ID of the Anonymous group is no longer 0.

### 4.1.3 Deposit crosswalk parameters

The following parameters are applicable only to deposit crosswalks. If any parameter is missing then its default value applies:

**use-leading-zeroes-for-dates**
Controls whether single digit day or month date-components will be prefixed with a leading "0". By default, this is set to `false`. For example, if depositing the date 2nd February 2018, in the format `yyyy-mm-dd`, the output will depend on the parameter value as follows:

| Parameter value | Outputted value |
|---|---|
| `true` | `2018-02-02` |
| `false` | `2018-2-2` |

**use-default-values-for-missing-date-components**
Date fields in Elements have varying levels of precision: day (fully precise), month, or year. When depositing a date which is precise only to the month or year, this parameter controls whether missing date components should default to "1". By default, this parameter is set to `false`. The behaviour of the crosswalk for combinations of parameter values and date precisions is shown below:

| Parameter value | Input date | Output value |
|---|---|---|
| `true` | 2nd February 2018 | `2018-02-02` |
| `true` | February 2018 | `2001-02-01` |
| `true` | 2018 | `2018-01-01` |
| `false` | 2nd February 2018 | `2018-02-02` |
| `false` | February 2018 | `2018-02` |
| `false` | 2018 | `2018` |

**dspace-xwalkout-enable-sword-on-behalf-of-header**
This parameter can be used to opt-in to behaviour for DSpace deposit. If set to true, the initial Sword v2 deposit operation will be performed with the `On-Behalf-Of` header set to the HR email address of the user performing the deposit.

Note that if this is enabled, and DSpace does not have accounts for every Elements user with access to the Repository Tools module, then this could lead to errors when depositing.

**dspace-xwalkout-enable-metadata-post-for-redeposit**
This parameter determines whether any metadata is added to the DSpace item when performing a subsequent deposit. If set to true, the result of the crosswalk will be added to the DSpace item.

Before enabling this, you should add a specific field-map to handle the subsequent deposit case. See Appendix E.1 for more details on subsequent deposit.

## 4.2 The `<xwalkin:object-type-selector>` element (harvest only)

The harvest crosswalk must be able to select the field-map for the item being crosswalked, and also the object type of the Elements publication record that is the output of the crosswalk. This is specified with the `<xwalkin:object-type-selector>` element. The selected field-map and object type can depend on the data of the item being processed; this will require use of logic expressions to specify the desired behaviour. These logic expressions should return a `<xwalkin:object-type-selection>` element. See Appendix D for full details on the use of logic expressions. An example is shown below:

```
<xwalkin:object-type-selector>
  <xwalk:choose>

    <!-- When the source field "dc.type" has the value "Article", the Elements object
    ↪  type will be set to "journal-article" and the field-map "journal-article-map"
    ↪  will be used. -->
    <xwalk:when>
      <xwalk:condition argument-field="dc.type"
      ↪  operator="equals">Article</xwalk:condition>
      <xwalk:result>
        <xwalkin:object-type-selection category="publication"
        ↪  object-type="journal-article" field-map="journal-article-map" />
      </xwalk:result>
    </xwalk:when>

    <!-- When the source field "dc.type" has the value "Book", the Elements object
    ↪  type will be set to "book" and the field-map "book-map" will be used. -->
    <xwalk:when>
      <xwalk:condition argument-field="dc.type"
      ↪  operator="equals">Book</xwalk:condition>
      <xwalk:result>
        <xwalkin:object-type-selection category="publication" object-type="book"
        ↪  field-map="book-map" />
      </xwalk:result>
    </xwalk:when>

    <!-- In all other cases, the Elements object type will be set to "conference" and
    ↪  the field-map "conference-map" will be used. -->
```

```
    <xwalk:otherwise>
      <xwalk:result>
        <xwalkin:object-type-selection category="publication"
        ↪  object-type="conference" field-map="conference-map" />
      </xwalk:result>
    </xwalk:otherwise>
  </xwalk:choose>
</xwalkin:object-type-selector>
```

**Note:** the specified object type may be ignored if the publication already exists and the harvested record it to be inserted into the existing publication object. In this case, the existing object's type will remain.

### 4.3 The `<xwalkout:field-map-selector>` element (deposit only)

Each deposit crosswalk must specify the field-map to be used for the item being crosswalked. The selected field-map can depend on the data of the item being processed; this will require use of logic expressions to specify the desired behaviour. These logic expressions should return a `<xwalkin:field-map-selection>` element. See Appendix D for full details on the user of logic expressions. An example is shown below:

```
<xwalkout:field-map-selector>
  <xwalk:choose>

    <!-- When the source object type is "journal-article", the field-map
    ↪  "journal-article-map" will be used. -->
    <xwalk:when>
      <xwalk:condition argument-field="object.type"
      ↪  operator="equals">journal-article</xwalk:condition>
      <xwalk:result>
        <xwalk:field-map-selection field-map="journal-article-map" />
      </xwalk:result>
    </xwalk:when>

    <!-- When the source object type is "book", the field-map "book-map" will be
    ↪  used. -->
    <xwalk:when>
      <xwalk:condition argument-field="object.type"
      ↪  operator="equals">book</xwalk:condition>
      <xwalk:result>
        <xwalkin:field-map-selection field-map="book-map" />
      </xwalk:result>
    </xwalk:when>

  </xwalk:choose>
</xwalkin:object-type-selector>
```

**Note:** When defining deposit crosswalks for automated metadata updates a `field-authority-list` attribute should be used to specify the name of a field authority list to be applied (see Section 4.6).

## 4.4 The `<xwalkout:collection-selector>` element (deposit only)

**Note:** a collection selector element is required only when depositing to DSpace.

When depositing from Elements to a destination system which has a concept of collections, this element is used to identify the destination collection that the item should be deposited into. The destination collection is specified using the `<xwalkout:collection-selection>` element, which specifies the collection by its name in the `name` attribute.

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 deposits journal articles into the collection named “OA Articles”, and all other items to “A N Other Collection”.

```
<xwalkout:collection-selector>
  <xwalk:choose>
    <!-- Journal Articles will be deposited to the "OA Articles" collection -->
    <xwalk:when>
      <xwalk:condition argument-field="object.type"
      ↪  operator="equals">journal-article</xwalk:condition>
      <xwalk:result>
        <xwalkout:collection-selection name="OA Articles" />
      </xwalk:result>
    </xwalk:when>

    <!-- All other objects will be deposited to "A N Other Collection" -->
    <xwalk:otherwise>
      <xwalk:result>
        <xwalkout:collection-selection name="A N Other Collection" />
      </xwalk:result>
    </xwalk:otherwise>
  </xwalk:choose>
</xwalkout:collection-selector>
```

## 4.5 The `<xwalk:file-metadata-map>` element (deposit only)

**Note:** this element is only supported when depositing to EPrints.

When depositing files from Elements to a destination system, this element can be used to specify the file-level metadata which should be supplied to the destination system. The `<xwalk:file-metadata-map>` can contain any number of `<xwalk:field-mapping>` elements (just like a `<xwalk:field-map>` element). The file-metadata-map is evaluated for *every* file included in the deposit operation, and each result is sent to the repository for the corresponding file.

In order to access some file-level metadata, some file-specific field identifiers can be used. These are prefixed with *file*; a full list is shown in section 6.1.9. For each evaluation of the file-metadata-map, the value returned by these file field identifiers will be equal to the value for the current file being processed.

A default file-metadata-map is used if none is provided. This includes the default implementation of the File Version and Reuse Licence functionality. If the file-metadata-map is overridden by explicitly including one, then the File Version and Reuse Functionality mappings will need to be explicitly specified.

An example of a file-metadata-map for EPrints is as follows:

```
<xwalk:file-metadata-map>
  <xwalk:field-mapping to="content">
    <xwalk:field-source from="file.version" />
  </xwalk:field-mapping>
  <xwalk:field-mapping to="license">
    <xwalk:field-source from="requested-reuse-licence.short-name" />
  </xwalk:field-mapping>
  <xwalk:field-mapping to="security">
    <xwalk:if>
      <xwalk:condition argument-field="deposit.type"
      ↪  operator="equals">subsequent</xwalk:condition>
      <xwalk:result>
      <xwalk:field-source value="staffonly"/>
      </xwalk:result>
    </xwalk:if>
  </xwalk:field-mapping>
</xwalk:file-metadata-map>
```

Note that the field-mapping to the *security* field sets the value “staffonly” when the initiating deposit operation is a *subsequent* deposit, rather than an initial deposit (see Appendix E.2 for more details).

## 4.6 The `<xwalk:field-authority-lists>` element (automatic metadata push only)

**Note:** this element is only used in the deposit crosswalk file when pushing Elements data to external systems (e.g. RT2 Automated metadata push).

The `<xwalk:field-authority-lists>` element defines one or more named field authority lists. A field authority list specifies a list of fields in the destination system (e.g. repository) which

can be updated by Elements when performing metadata updates. (It is important to only have one system in which metadata values can be updated - if not, updates in one system can be overwritten by updates from another.)

Here is an example of a set of field authority lists.

```
<xwalk:field-authority-lists>
  <xwalk:field-authority-list name="authority-list-1">
    <xwalk:field-authority field-name="title" />
    <xwalk:field-authority field-name="volume" />
    <xwalk:field-authority field-name="creators" />
    <xwalk:field-authority field-name="official_url" />
  </xwalk:field-authority-list>
  <xwalk:field-authority-list name="authority-list-2">
    <xwalk:field-authority field-name ="issn"/>
    <xwalk:field-authority field-name ="publication"/>
  </xwalk:field-authority-list>
</xwalk:field-authority-lists>
```

Multiple field authority lists can be defined. Within each `<xwalk:field-authority-list>`, the fields which Elements has authority to update are defined by the `field-name` attribute in the `<xwalk:field-authority>` elements. The `field-name` must be the same as the field name at the **repository**, not the Elements API.

The following is an example showing how a field authority list is selected:

```
<xwalkout:field-map-selector>
  <xwalk:field-map-selection field-map ="all-types"
  ↪  field-authority-list="authority-list-1"/>
</xwalkout:field-map-selector>
```

The `field-authority-list` of `<xwalk:field-map-selection>` chooses which field authority list to use by using the `name` in `<xwalk:field-authority-list>` as the unique identifier. Hence, `field-authority-list` must be equal to the `name` of the wanted field authority list. Since the selector has chosen *authority-list-1*, only the metadata fields “title“, “volume“, “creators“ and “official_url“ can be updated by Elements.

**Note:** the specification of a `field-authority-list` attribute will not affect standard deposit behaviour when initially depositing an item to a repository - it is only used for automated metadata updates.

The table below shows, for a metadata field, how a metadata push affects a repository value for the range of possible states of field authority and the presence of the field metadata in Elements and in the repository.

| Elements has metadata | Repository has metadata | Field Authority present | Result at Repository |
|---|---|---|---|
| Yes | Yes | Yes | Elements |
| Yes | Yes | No | Repository |
| Yes | No | Yes | Elements |
| Yes | No | No | No data |
| No | Yes | Yes | Repository |
| No | Yes | No | Repository |
| No | No | Yes | No data |
| No | No | No | No data |

# 5 Harvest-specific <xwalk:field-map> usage

This section describes harvest-specific functionality of the <xwalk:field-map> element and its children.

## 5.1 Harvest specific <xwalk:field-source> usage

As described in Section 2, each <xwalk:field-mapping> element should contain <xwalk:field-source> elements which specify the source data. This section describes the harvest-specific features of field-sources.

### 5.1.1 Harvesting into compound Elements fields

When harvesting into an Elements field which is of a compound type, the map needs to specify how to translate data into the Elements data-type.

**The `data-part` attribute**

The `data-part` attribute specifies which component of an Elements compound field to target with a piece of repository data.

For example, the Elements "keyword" field type has data-parts `keyword:scheme` and `keyword:value`. To harvest two field values (*dc.keyword.scheme* and *dc.keyword.value*) into these components of the Elements *keywords* field, the following field-mapping can be used:

```
<xwalk:field-mapping to="keywords">
  <xwalk:field-source>
    <xwalk:field-source from="dc.keyword.value" data-part="keyword:value"/>
    <xwalk:field-source from="dc.keyword.scheme" data-part="keyword:scheme"/>
  </xwalk:field-source>
</xwalk:field-mapping>
```

Note that the outer <xwalk:field-source> element is used to identify grouped data-parts. It is not required in the above example since there are no other values with data-parts, but is good practice to include it.

For a full list of the `data-part` attribute values available when harvesting, see Appendix A.

Some Elements compound field types contain subfields that are themselves compound. To harvest data into into these subfields it is necessary to use nested field-sources, wrapping a <xwalk:field-source> with a `from` attribute with another <xwalk:field-source> with a `data-part` attribute.

For example, the each item an Elements field of type "person-list" has a `person:identifier` data-part, which is itself a compound field of type "identifier-list". Furthermore, each item in a multivalued Elements field of type "identifier-list" has data-parts `identifier:scheme` and

`identifier:value`. To harvest into these sub-components of the Elements *authors* field, the following field-mapping can be used:

```
<xwalk:field-mapping to="authors">

  <!-- Map some field to the author name -->
  <xwalk:field-source from="dc.creator.primaryAuthor" />

  <!-- A wrapping field-source contains all the "person:identifier" data-parts -->
  <xwalk:field-source data-part="person:identifier">
    <xwalk:field-source from="dc.creator.arxiv-id" data-part="identifier:value" />
    <xwalk:field-source value="arxiv" data-part="identifier:scheme" />
  </xwalk:field-source>

</xwalk:field-mapping>
```

Note the use of the `value` attribute to place a hard-coded string into the `identifier:scheme` data-part.

**The `format` attribute**

Some compound Elements fields can be parsed from common string representations. For example, the Elements field-type "date" consists of three components (day, month and year); a string like "1994-09-22" can easily be parsed into these three components. To simplify the crosswalks, the `format` attribute can be used to automatically map into compound Elements fields from some standard string representations.

For example, the following field-mapping interprets a string of the form `yyyy-mm-dd`, parsing it into the Elements data-part components `date:year`, `date:month` and `date:day`:

```
<xwalk:field-mapping to="start-date">
  <xwalk:field-source from="dc.date.started" format="date:yyyy-mm-dd" />
</xwalk:field-mapping>
```

This is much more concise than the alternative of manually identifying each component of the input string and mapping each component to the relevant data-part.

For a full list of the `format` attribute values available when harvesting, see Appendix A.

### 5.1.2 Harvesting from multivalued text fields

If the crosswalk needs to apply processing to each value of a multivalued field from the source system, then nested `<xwalk:field-source>` elements can be used. By setting the `from` attribute of an inner field-source to the '.' character, any processing specified (see Section 2.4: Manipulating text) will apply to *each* result of the outer field-source. For example:

```
<xwalk:field-mapping to="c-list">
  <xwalk:field-source from="list-field">
    <!-- Each item in the multivalued field "list-field" will be passed to the inner
    ↪  field-source, which will prefix each value with the text "List item: ". -->
    <xwalk:field-source from="." prefix="List item: " />
  </xwalk:field-source>
</xwalk:field-mapping>
```

This field-mapping is harvesting data from *list-field*; the innermost field-source specifies that each item within the list should be prefixed with the value “List item: ”.

If multiple rounds of processing are required on each item within a multivalued field, this functionality can be combined with that described in Section 2.5: Applying multiple processing operations to text. For example, applying two value-maps to each item in a list can be achieved by the following:

```
<xwalk:field-mapping to="c-list">
  <xwalk:field-source from="list-field">

    <!-- Each item in the multivalued field "list-field" is passed to the inner
    ↪  block, which uses a wrapping field-source to apply two value-maps in
    ↪  succession -->
    <xwalk:field-source value-map="value-map-2">
      <xwalk:field-source from="." value-map="value-map-1" />
    </xwalk:field-source>

  </xwalk:field-source>
</xwalk:field-mapping>
```

### 5.1.3 Harvesting from compound fields

**Note:** this section only applies to source systems which support compound field types (i.e. structured data rather than text) – EPrints, Figshare for Institutions and Hyrax.

To select a subfield from a compound field, nested field-sources can be used. For example, if an EPrints field *outer-field* contains a subfield *subfield-1*, this can be selected with the following field-sources:

```
<xwalk:field-mapping to="c-text-field">
  <xwalk:field-source from="outer-field">
    <xwalk:field-source from="subfield-1" />
  </xwalk:field-source>
</xwalk:field-mapping>
```

An alternative, more concise, expression of this field-mapping uses a feature of the `from` attribute of a field-source: if the attribute value contains the ‘/’ character, then the text after the

slash will be interpreted as a subfield. For example, the following field-mapping is equivalent to the one above:

```
<xwalk:field-mapping to="c-text-field">
  <xwalk:field-source from="outer-field/subfield-1" />
</xwalk:field-mapping>
```

Multiple '/' characters can be used if there are multiple levels of subfields; this can help simplify complex field-sources.

### 5.1.4 Harvesting from multivalued compound fields

**Note:** this section only applies to source systems which support compound field types (i.e. structured data rather than text) – EPrints, Figshare for Institutions and Hyrax.

To select data from a multivalued field, where each item is a compound data type, the approach described in the previous section can be used, with the subfield name supplied for the value of the `from` attribute of a nested field-source.

For example, if the field *authors* is a multivalued field, where each item is a compound object which contains the subfield *full_name*, then the following field-mapping will map the set of full names to the field *c-author-names*:

```
<xwalk:field-mapping to="c-author-names">
  <xwalk:field-source from="authors">
    <!-- For each entry in the multivalued "authors" field, the subfield "full_name"
    ↪  is selected -->
    <xwalk:field-source from="full_name" />
  </xwalk:field-source>
</xwalk:field-mapping>
```

Note that the use of the '/' character as described in Section 5.1.3 is not applicable here, since the *authors* field is multivalued.

### 5.1.5 Harvesting from multivalued compound fields into compound Elements fields

**Note:** this section only applies to source systems which support compound field types (i.e. structured data rather than text) – EPrints, Figshare for Institutions and Hyrax.

Combining the features described in the previous sections can achieve some complex data mappings, harvesting from multivalued compound repository fields into compound Elements fields.

For example, harvesting the *creators* field in EPrints (a field which contains multiple compound objects representing a creator) into the *authors* field in Elements (a field of type "person-list",

which supports name components, along with various identifiers) can be achieved with the following field-mapping:

```
<xwalk:field-mapping to="authors">

  <!-- Loops over every item in the multivalued "creators" field -->
  <xwalk:field-source from="creators">

    <!-- Selects the subfields "family" and "given" from the subfield "name", and
    ↪  maps them to the "person:lastname" and "person:firstnames" Elements
    ↪  data-parts -->
    <xwalk:field-source from="name/family" data-part="person:lastname" />
    <xwalk:field-source from="name/given" data-part="person:firstnames" />

    <!-- Selects the subfield "id" and maps it to the "person:email-address"
    ↪  data-part -->
    <xwalk:field-source from="id" data-part="person:email-address" />

    <!-- Adds a "person:identifier" data-part to the person, the contents of which is
    ↪  obtained by evaluating the inner field-sources -->
    <xwalk:field-source data-part="person:identifier">
      <!-- Maps the "orcid" subfield to the "identifier:value" data-part; maps the
      ↪  value "orcid" to the "identifier:scheme" data-part. -->
      <xwalk:field-source from="orcid" data-part="identifier:value" />
      <xwalk:field-source value="orcid" data-part="identifier:scheme" />
    </xwalk:field-source>

  </xwalk:field-source>

</xwalk:field-mapping>
```

### 5.1.6 Harvesting from namespaced XML

**Note:** this section only applies to source systems which may contain XML data from multiple namespaces – Hyrax.

When harvesting from a Hyrax system, the `from` attribute within a field-source will be assumed to refer to an XML element in the MODS metadata format, unless specified otherwise. For example, the following field-source:

```
<xwalk:field-source from="titleInfo" />
```

will select the `titleInfo` element from the MODS namespace:

```
<mods:titleInfo xmlns:mods="https://www.loc.gov/standards/mods/">
    <mods:title>Title here</mods:title>
</mods:titleInfo>
```

To select elements belonging to other namespaces, define the namespace alias at the root element of the crosswalk map, as follows:

```
<xwalkin:consolidated-maps
↪  xmlns:xwalk="http://www.symplectic.co.uk/elements/xwalkcommon"
↪  xmlns:xwalkin="http://www.symplectic.co.uk/elements/xwalkin"
↪  xmlns:dc="http://purl.org/dc/elements/1.1/">
```

and then use the defined namespace prefix within the `from` attribute, as follows:

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

### 5.1.7 Harvesting from XML attributes

**Note:** this section only applies to source systems which may contain data within XML attributes – Hyrax.

To select an attribute value from source data, the `from` attribute within a field-source should be prefixed with the `@` character. For example:

```
<xwalk:field-source from="@type" />
```

If this field-source is applied to the following source data:

```
<mods:relatedItem type="host">
```

then the result of the evaluating the field-source will be the text “host”.

This can be combined with the syntax described in section Section 5.1.3, as follows:

```
<xwalk:field-source from="titleInfo/title/@type" />
```

The above field-source will select the *type* attribute from the *title* element contained within the *titleInfo* element.

# 6 Deposit-specific <xwalk:field-map> usage

This section describes deposit-specific functionality of the <xwalk:field-map> element and its children.

## 6.1 Field identifiers

When performing a deposit from Elements, a number of special identifiers can be used to reference other Elements values:

### 6.1.1 Object field identifiers

Identifiers with the prefix “object” reference object-level information for the publication that is being deposited.

| Field identifier | Description |
|---|---|
| object.type | The Elements object type (journal-article, book etc). |
| object.id | The internal Elements ID of the object. |
| object.reporting-date-1 | The object’s reporting date 1. |
| object.reporting-date-2 | The object’s reporting date 2. |
| object.labels | The list of all labels for the object. |

Note that object.labels can also be referred to by publication.labels.

### 6.1.2 Depositor field identifiers

Identifiers with the prefix “depositor” or “impersonated-depositor” reference the Elements user who is logged in or being impersonated when performing the deposit.

| Field identifier | Description |
|---|---|
| depositor.id | The internal Elements ID of the user. |
| depositor.proprietary-id | The proprietary ID of the Elements user (this is the user's ID as supplied by the HR feed). |
| depositor.username | The username of the Elements user (used when logging into Elements). |
| depositor.initials | The initials of the Elements user. |
| depositor.first-name | The first name of the Elements user. |
| depositor.last-name | The last name of the Elements user. |
| depositor.full-name | The full name of the Elements user. |
| depositor.email-address | The email address of the Elements user. |
| depositor.primary-group-descriptor | The primary group that the Elements user belongs to. |
| depositor.is-academic | Whether the Elements user is an academic or not. |
| depositor.is-impersonating | If examining the logged in user, returns whether or not that user is currently impersonating another Elements user. |

The field identifiers above are shown using the "depositor" prefix; if the logged in user is impersonating someone else, these identifiers will select values relating to the the delegate or administrator, *not* the impersonated user. To select data relating to the impersonated user, replace the prefix with "impersonated-depositor". E.g. `impersonated-depositor.full-name`.

**Note**: the `depositor.is-impersonating` field identifier is only valid with the `depositor` prefix; specifying `impersonated-depositor.is-impersonating` will never return a value.

### 6.1.3 Requested embargo field identifiers

These reference the requested embargo data specified by a user on deposit.

| Field identifier | Description |
|---|---|
| requested-embargo.id | The internal Elements ID that identifies the selected embargo period. |
| requested-embargo.display-name | The display text associated with the selected embargo period. |
| requested-embargo.comment | The comment text entered by a user when requesting an embargo. |

Requested embargo fields are only available if you have enabled the 'Support embargo advice' option in the Deposit Settings for your data source.

### 6.1.4 Reuse licence field identifiers

These reference the reuse licence selected by a user on deposit.

| Field identifier | Description |
|---|---|
| `requested-reuse-licence.id` | The internal Elements ID that identifies the selected reuse licence. |
| `requested-reuse-licence.name` | The full name of the selected reuse licence (e.g. “CC BY-ND Attribution-NoDerivs - CC BY-ND”) |
| `requested-reuse-licence.short-name` | The short name of the selected reuse licence (e.g. “CC BY-ND”) |
| `requested-reuse-licence.comment` | The comment text entered by a user when requesting a reuse licence. |

Reuse licence fields are only available if you have enabled the ‘Support reuse licence’ option in the Deposit Settings for your data source.

### 6.1.5 Deposit licence field identifiers

These reference the deposit licence selected by a user on deposit

| Field identifier | Description |
|---|---|
| `deposit-licence.identifier` | The internal Elements ID that identifies the selected deposit licence. |
| `deposit-licence.display-name` | The name of the selected deposit licence. |

Deposit licence fields are only available if you have defined and enabled one or more deposit licences for your data source.

### 6.1.6 Object neighbourhood field identifiers

These reference the links between the object being deposited and other objects in Elements. For a full explanation of these fields and how to reference them in a deposit crosswalk see Appendix C: Depositing object neighbourhood data.

| Field identifier | Description |
|---|---|
| `object.relationships` | A multivalued field detailing all Elements relationships (approved links) involving the object being deposited, along with data for the object on the opposite end of each relationship (see Appendix C.2). |
| `object.groups` | A multivalued field detailing all Elements groups this publication is related to (see Appendix C.3). |

### 6.1.7 OA Location field identifiers

If an OA Location was specified when depositing, these field identifiers reference the provided values.

| Field identifier | Description |
|---|---|
| `oa-location-url` | The URL of the Open Access location provided by the depositing user. |
| `oa-location-file-version` | The selected File Version for the “OA Location” file. |

**Note:** if OA Locations are used, the inbound crosswalk map file **must** symmetrically map OA Location back to the Elements field *oa-location-url*. If Elements detects that a crosswalk round-trip did not preserve this value, a warning will be logged to the system log.

### 6.1.8 Files field identifiers

These field allow selection of properties of the files being uploaded to the repository, for use in item-level metadata. Note that this is distinct from the identifiers described in Section 6.1.9, which are used when crosswalking file-level metadata.

| Field identifier | Description |
|---|---|
| `files.names` | A multivalued field containing the names of all the files being uploaded, including their extensions. |
| `files.count` | The total number of files being uploaded.. |

### 6.1.9 File field identifiers

These field identifiers are only valid within a file-metadata-map (see Section 4.5), and so are only available for use when crosswalking file-level metadata. These identifiers reference certain properties on the files being uploaded to the repository.

| Field identifier | Description |
|---|---|
| `file.name` | The name of the file, including its extension. |
| `file.version` | If a File Version was selected for this file at the point of deposit, will be equal to that selected file version. |

### 6.1.10 Other field identifiers

There are some other field identifiers which reference some general deposit-related information:

| Field identifier | Description |
|---|---|
| `deposit.date` | The date on which the deposit is taking place. |
| `deposit.type` | Equal to either “initial”, “subsequent” or “push“, depending on whether the operation is an Initial Deposit, a Subsequent Deposit (that is, an operation which is supplementing an existing repository item with additional files) or an automated metadata push. |

## 6.2 Deposit specific `<xwalk:field-source>` usage

As described in Section 2, each `<xwalk:field-mapping>` element should contain `<xwalk:field-source>` elements which specify the source data. This section describes the deposit-specific features of field-sources.

### 6.2.1 Depositing from Elements compound fields

#### The data-part attribute

The `data-part` attribute on a `<xwalk:field-source>` element can be used to select a specific component of data to be used. For example, to crosswalk only the last names of authors, the following field-mapping can be used:

```
<xwalk:field-mapping to="destination-field">
  <xwalk:field-source from="authors" data-part="person:lastname" />
</xwalk:field-mapping>
```

Some Elements compound fields have data-parts that are themselves of compound field type. To access components of these compound subfields, nested field-sources must be used. For example, to crosswalk the identifier values of authors, the following field-mapping could be used:

```
<xwalk:field-mapping to="destination-field" separator="; ">
  <xwalk:field-source from="authors" data-part="person:identifiers">
    <xwalk:field-source data-part="identifier:value"/>
  </xwalk:field-source>
</xwalk:field-mapping>
```

For a full list of the `data-part` attribute values available when depositing, see Appendix B.

#### The `format` attribute

The `format` attribute on a `<xwalk:field-source>` element can be used to format an Elements field as a single string. For example, to crosswalk a list of authors, with each author formatted as last name followed by initials, the following field-mapping can be used:

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

For a full list of the `format` attribute values available when depositing, see Appendix B.

**Note:** the `format` attribute is not supported when used in nested field-sources.

### 6.2.2 Depositing to multivalued fields

To deposit data to a multivalued field, simply use a field-source which selects multiple values. For example:

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

#### Destinations which distinguish lists from scalars

EPrints and Figshare for Institutions have a different way of representing data in multivalued fields compared with scalar fields. For these repositories, it may be necessary to specify whether the destination field is multivalued using the `is-list` attribute. This is only necessary in the case where the source data *may* sometimes consist of only one item (but the result of the crosswalk should still represent this as a list).

The following example demonstrates the different output produced for a source field *c-list* which, in this case, contains only one entry (the text “Single value”):

```
<field-mapping to="single-valued-field">
  <field-source from="c-list" />
</field-mapping>

<field-mapping to="multivalued-field" is-list="true">
  <field-source from="c-list" />
</field-mapping>
```

The output of these two field-mappings would differ in whether the destination field is marked up as a multivalued field. For example, the output for EPrints would be:

```
<single-valued-field>Single value</single-valued-field>
<multivalued-field>
  <item>Single value</item>
</multivalued-field>
```

Note the presence of the `<item>` element within the `multivalued-field` element; this signifies that *multivalued-field* is multivalued.

The output for Figshare for Institutions would be:

```
{
  "single-valued-field": "Single value",
  "multivalued-field": [
    "Single value"
  ]
}
```

Note that the value for `multivalued-field` is a JSON array (i.e. is multivalued), whereas the value for `single-valued-field` is a string.

The attribute `is-list` is valid on both `<xwalk:field-mapping>` elements, and `<xwalk:field-source>` elements which describe a mapping to a subfield (see Section 6.2.3).

If `is-list` is `false` but the field-source produces multiple values, only the first value will be used.

### 6.2.3 Depositing to compound fields

**Note:** this section only applies to destination systems which support compound field types: EPrints and Figshare for Institutions.

When depositing data to a compound repository field containing subfields, nested field-sources can be used to specify the appropriate mapping. The `subfield` attribute can be used to specify a mapping to a subfield.

For example, if the destination system has a compound field *year-and-month* which contains two subfields: *year* and *month*, the following field-mapping will map data from two Elements fields to the compound repository field:

```
<xwalk:field-mapping to="year-and-month">
  <xwalk:field-source from="c-year" subfield="year" />
  <xwalk:field-source from="c-month" subfield="month" />
</xwalk:field-mapping>
```

The output of this field-mapping for EPrints would look like the following:

```
<year-and-month>
  <year>2012</year>
  <month>June</month>
</year-and-month>
```

For Figshare for Institutions, the output would look like the following:

```
{
  "year-and-month": {
    "year": 2012,
    "month": "June"
  }
}
```

### 6.2.4 Depositing from Elements compound fields to repository compound fields

**Note:** this section only applies to destination systems which support compound field types: EPrints and Figshare for Institutions.

Combining the features described in the previous sections we can achieve some complex data mappings when depositing from compound Elements fields into multivalued compound repository fields.

The following example shows a field-mapping suitable for depositing the *authors* field in Elements to the *creators* field in EPrints. By default, the *creators* field includes structured name data and an email address. With the EPrints ORCID plugin, this also has a subfield *orcid*.

```
<xwalk:field-mapping to="creators" is-list="true">

  <!-- Loops over every item in the "authors" field -->
  <xwalk:field-source from="authors">

    <!-- Selects the data-parts "person:lastname" and "person:firstnames" from the
    ↪  Elements author and maps them to the "family" and "given" subfields of the
    ↪  EPrints creators field -->
    <xwalk:field-source subfield="family" data-part="person:lastname" />
    <xwalk:field-source subfield="given" data-part="person:firstnames" />

    <!-- Selects the data-part "person:email-address" from the Elements author field
    ↪  and maps it to the EPrints "id" subfield -->
    <xwalk:field-source subfield="id" data-part="person:email-address"/>

    <!-- Selects the data-part "person:identifier" from the Elements author. This is
    ↪  a multivalued compound subfield of the authors field. Each item is passed to
    ↪  the "orcid-person-identifier" value-map which excludes any non-ORCID
    ↪  identifiers. The remaining identifier (if present) is passed to the inner
    ↪  field-source -->
    <xwalk:field-source data-part="person:identifier"
    ↪  value-map="orcid-person-identifier">

      <!-- Selects the subcomponent with data-part "identifier:value" and maps it to
      ↪  the "orcid" subfield -->
      <xwalk:field-source subfield="orcid" data-part="identifier:value" />
```

```
    </xwalk:field-source>

  </xwalk:field-source>

</xwalk:field-mapping>
```

The value-map used in the above field-source above to filter out non-ORCID identifiers is as follows:

```
<xwalk:value-map name="orcid-person-identifier" data-part="identifier:scheme">
  <xwalk:value-mapping from="orcid" action="continue" />
  <xwalk:otherwise-mapping action="ignore-this-value" />
</xwalk:value-map>
```

### 6.2.5 Deposting JSON data

**Note:** this section only applies to Figshare for Institutions.

When depositing to a system where the underlying data is represented in JSON rather than XML, some extra considerations are required when using field-sources.

Unlike XML, JSON data is typed: the representation of data is dependant on the type of the data. As a result, numerical values and booleans are represented differently to text. The crosswalk framework will inspect the outgoing data to format the data in a suitable way, but this cannot always be perfect. If depositing to a text field a value which may sometimes look like a number or a boolean (for example, the text string “true”, or the text string “42”), the destination data-type can be specified with the `data-type` attribute. The allowed values are:

| data-type |
|---|
| `json:string` |
| `json:number` |
| `json:boolean` |

This attribute is used to force the output data to be rendered in the specified format. If the data cannot be rendered as the specified type (e.g. rendering “hello” as `json:number`), an error will occur at the point of the crosswalk. It is therefore inadvisable to ever attempt to coerce text fields to boolean or number data-types (since there will usually fail); coercing boolean or number data into text, however, always succeeds.

An example of the usage of this attribute and the resulting output is shown below:

```
<xwalk:field-mapping to="number-field">
  <xwalk:field-source from="c-number" />
<xwalk:field-mapping>

<xwalk:field-mapping to="text-field">
```

```
  <xwalk:field-source from="c-number" data-type="json:string" />
<xwalk:field-mapping>
```

If the field *c-number* has the value 42, then the resulting output will be as follows:

```
{
  "number-field": 42,
  "text-field": "42"
}
```

Note the lack of quotes surrounding the *number-field* value: this indicates that the value is a numerical data type. The presence of quotes surrounding the *text-field* value indicates that the value has a data type of text. When depositing to Figshare for Institutions, the data types must be correct or the deposit may fail.

### 6.2.6 Deposting to namespaced XML

**Note:** this section only applies to Hyrax

When depositing to a Hyrax system, the metadata format output by the crosswalks is MODS, wrapped in a METS envelope. By default, any field-mappings will map to the MODS namespace. For example:

```
 <xwalk:field-mapping to="abstract">
    <xwalk:field-source from="abstract" />
<xwalk:field-mapping>
```

will produce XML with the following structure:

```
<mods:abstract xmlns:mods="http://www.loc.gov/mods/v3">
    We prove the title.
</mods:abstract>
```

However, the namespace of the output XML can be specified by using the XML namespace aliases present at the top-level of the map file. For example, if the top-level `consolidated-maps` element is as follows:

```
<xwalkout:consolidated-maps
↪  xmlns:xwalk="http://www.symplectic.co.uk/elements/xwalkcommon"
↪  xmlns:xwalkout="http://www.symplectic.co.uk/elements/xwalkout"
↪  xmlns:dc="http://purl.org/dc/elements/1.1/">
```

then the defined namespace prefix `dc` can be used within field-mappings and subfield field-sources. The metadata prefix `mods` is always available and doesn't need to be defined on the `consolidated-maps` element. The following field-mapping:

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

will produce output XML like:

```
<mods:extension xmlns:mods="http://www.loc.gov/mods/v3">
  <dc:title xmlns:dc="http://purl.org/dc/elements/1.1/">Article title</dc:title>
</mods:extension>
```

### 6.2.7 Depositing to XML attributes

**Note:** this section only applies to Hyrax

The MODS metadata standard used in the Hyrax crosswalks makes significant use of XML attributes. The crosswalk map file allows specification of a mapping to an attibute by prefixing the attribute name with an `@` symbol within a `subfield`. For example:

```
<xwalk:field-mapping to="relatedItem">
    <xwalk:field-source subfield="@otherType" value="related_item" />
    <xwalk:field-source subfield="titleInfo">
        <xwalk:field-source subfield="title" value="My related item" />
    </xwalk:field-source>
<xwalk:field-mapping>
```

produces the following MODS XML:

```
<mods:relatedItem otherType="related_item">
    <mods:titleInfo>
        <mods:title>My related item</mods:title>
    </mods:titleInfo>
</mods:relatedItem>
```

The handling of field-mappings where the selected source data is multivalued is to repeat the output XML element for each occurence of the source data. For example, consider the following field-mapping:

```
<xwalk:field-mapping to="name">
    <xwalk:field-source subfield="@type" value="personal" />
    <xwalk:field-source from="authors">
```

```
      <xwalk:field-source subfield="displayForm" format="person:firstnames-lastname"
      ↪ />
    </xwalk:field-source>
</xwalk:field-source>
```

This produces a `name` element for each value within the selected *authors* field:

```
<mods:name type="personal">
    <mods:displayForm>Andrew Bennet</mods:displayForm>
</mods:name>
<mods:name type="personal">
    <mods:displayForm>Dave Budenberg</mods:displayForm>
</mods:name>
```

Also note that any subfields mapping to attributes which are present alongside a multivalued-selecting field-source have their attribute repeated for each instance of the field generated.

# A Harvesting into Elements compound fields: formats and data-parts

This section details the options for the `data-part` and `format` attributes when constructing field-mappings from repository fields to Elements compound fields.

The `data-part` attribute specifies which part of the Elements compound field the repository data should be crosswalked into. Required data-parts that must be populated when harvesting are marked with a dagger†.

The `format` attribute describes how to parse a single string of repository data into the data-parts of the Elements field. For each compound field type for which the `format` attribute is implemented, the default value to be used if none is specified is marked with an asterisk*. Some of these default values can be overridden using the `<xwalk:parameters>` element – see Section 4.1 for more details.

For multivalued fields (i.e. field types ending in "list"), the options apply to each element of the list. For example, the data-parts of the person-list type are applicable for a single person item within the list.

## A.1 Date fields

| data-part | format |
|---|---|
| date:day | date:YYYY-MM-DD* |
| date:month | date:DMY |
| date:year† | date:MDY |
| | date:YMD |
| | date:YM |
| | date:MY |
| | date:Y |

**Note:** the `format-delimiter` attribute can also be used in conjunction with the `format` attribute to specify the character that separates the elements of a date string.

## A.2 Money fields

When harvesting into money fields, the `format` attribute is used to specify the currency of incoming amounts. If the repository field only contains a number (the amount), then using a format of `money:gbp` will include "GBP" as the currency component.

| data-part | format |
|---|---|
| money:amount† | money:usd* |
| money:currency | money:gbp |
| | money:aud |
| | money:nzd |
| | money:cad |
| | money:eur |
| | money:jpy |
| | money:sgd |

## A.3 Pagination fields

| data-part | format |
|---|---|
| pagination:begin-page | pagination:start-end* |
| pagination:end-page | pagination:start |
| pagination:page-count | pagination:end |
| | pagination:page-count |

**Note:** the `format-delimtier` attribute can also be used in conjunction with `format` to specify the character that separates the elements of a `pagination:start-end` string.

## A.4 Address list fields

Interpreting a single string as a formatted address or address list is not supported; there are no available `format` options for this field type.

| data-part |
|---|
| address:name |
| address:organisation |
| address:suborganisation |
| address:streetaddress |
| address:city |
| address:state |
| address:zipcode |
| address:country |
| address:type |
| address:iso-country-code |

## A.5 Keyword list fields

Interpreting a single string as a formatted keyword or keyword list is not supported; there are no available `format` options for this field type.

| data-part |
|---|
| keyword:value |
| keyword:scheme |
| keyword:percent |

## A.6 Identifier list fields

When harvesting into identifier list fields, the `format` attribute is used to specify the identifier scheme of incoming identifiers. If the repository field only contains an identifier string, then using a format of `identifier:arxiv` will include `arxiv` as the scheme component.

| data-part |
|---|
| identifier:value |
| identifier:scheme |

| format |
|---|
| identifier:arxiv |
| identifier:arxiv-author-id |
| identifier:cinii-article-id |
| identifier:crossref-article-id |
| identifier:dais |
| identifier:dimensions-grant-id |
| identifier:dimensions-publication-id |
| identifier:email-address |
| identifier:epmc-article-id |
| identifier:figshare-for-institutions-user-account-id |
| identifier:figshare-for-institutions-user-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 |

## A.7 Person list fields

| data-part |
| --- |
| person:lastname |
| person:initials |
| person:firstnames |
| person:email-address |
| person:identifier |
| person:address |
| person:role |

| format |
| --- |
| person:lastname-initials* |
| person:initials-lastname |
| person:lastname-firstnames |
| person:firstnames-lastname |
| |

**Note:** the `format-delimiter` attribute can also be used in conjunction with the `format` attribute to specify the character that separates the `lastname` and `firstnames` components of a person string.

**Note:** the `person:identifier` and `person:role` data-parts are compound identifier list subfields of the Elements person-list field type. Data-parts of this subfield must be referenced using nested field-sources (see Section 5.1.1) and the available `data-part` options for an identifier or role respectively (see Appendix A.6).

# B Depositing from Elements compound fields: formats and data-parts

This section details the options for the `data-part` and `format` attributes when constructing field-mappings from compound Elements fields to repository fields. The `data-part` attribute is used to select data contained within compound fields and the `format` attribute describes how to combine the data-parts into a string. For each compound field type for which the `format` attribute is implemented, the default value to be used if none is specified is marked with an asterisk*. Some of these default values can be overridden using the `<xwalk:parameters>` element, see Section 4.1 for more details.

For multivalued fields (i.e. field types ending in "list"), the options apply to each element of the list. For example, the data-parts of the person-list type are applicable for a single person item within the list.

## B.1 Date fields

| data-part |
|---|
| date:day |
| date:month |
| date:year |
| |
| |
| |
| |

| format |
|---|
| date:YYYY-MM-DD* |
| date:DMY |
| date:MDY |
| date:YMD |
| date:YM |
| date:MY |
| date:Y |

**Note:** the `format-delimtiter` attribute can also be used in conjunction with `format` to specify the character that separates the elements of a date string.

## B.2 Money fields

| data-part |
|---|
| money:amount |
| money:currency |
| |
| |

| format |
|---|
| money:currency-amount |
| money:amount-currency |
| money:amount |
| money:currency |

## B.3 Pagination fields

**Note:** the `format-delimiter` attribute can also be used in conjunction with `format` to specify the character that separates the elements of a `pagination:start-end` string.

| data-part | format |
|---|---|
| pagination:begin-page | pagination:start-end* |
| pagination:end-page | pagination:start |
| pagination:page-count | pagination:end |
| | pagination:page-count |

## B.4 Address list fields

| data-part | format |
|---|---|
| address:name | address:full-single-line* |
| address:organisation | address:full-multi-line |
| address:suborganisation | address:name |
| address:streetaddress | address:organisation |
| address:city | address:suborganisation |
| address:state | address:streetaddress |
| address:zipcode | address:city |
| address:country | address:state |
| address:type | address:zipcode |
| address:iso-country-code | address:country |
| | address:type |
| | address:iso-country-code |

## B.5 Keyword list fields

| data-part | format |
|---|---|
| keyword:value | keyword:value* |
| keyword:scheme | keyword:scheme |
| keyword:percent | keyword:percent |
| keyword:source | keyword:source |
| keyword:origin | keyword:origin |
| | keyword:with-scheme |
| | keyword:with-percent |
| | keyword:with-scheme-percent |

## B.6 Identifier list fields

| data-part | format |
|---|---|
| identifier:value | identifier:value-with-scheme* |
| identifier:scheme | identifier:value |
| identifier:status | identifier:scheme |
| identifier:decision | |

## B.7 Person list fields

| data-part |
| --- |
| person:lastname |
| person:initials |
| person:firstnames |
| person:email-address |
| person:id |
| person:address |
| person:identifier |
| person:resolved-user |
| person:role |

| format |
| --- |
| person:lastname-initials* |
| person:initials-lastname |
| person:lastname-firstnames |
| person:firstnames-lastname |
| |
| |
| |

**Note:** the last four data-parts are compound subfields of the Elements person-list field type and must be selected from using the functionality described in Section 6.2.

- `person:address`: field type address list (see section Appendix B.4)
- `person:identifier`: field type identifier list (see section Appendix B.6)
- `person:resolved-user`: this is an Elements User object (see section Appendix C.1)
- `person:role`: field type person role list (see section Appendix B.8)

## B.8 Person Role list fields

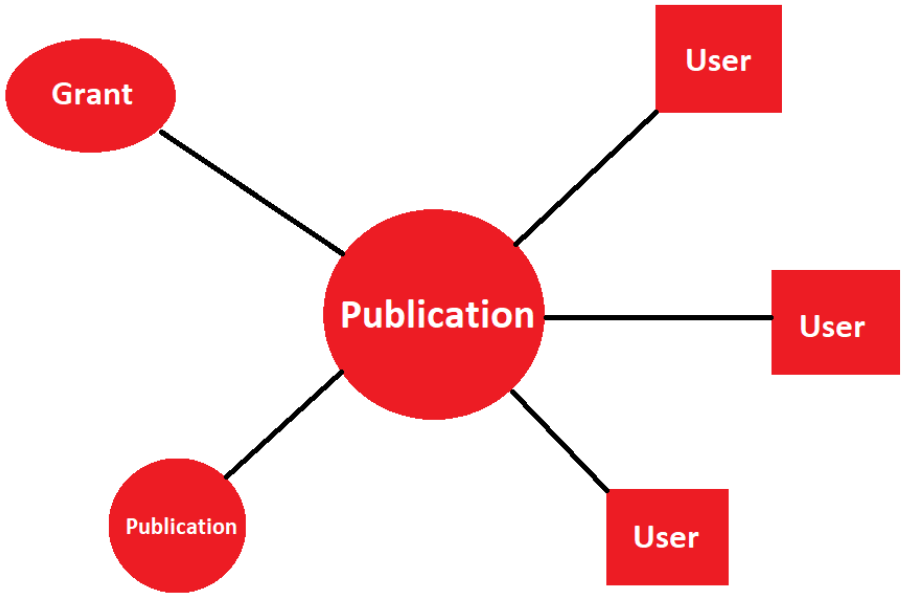
| data-part |
| --- |
| person-role:type |
| person-role:value |

# C Depositing object neighbourhood data

When depositing an Elements publication object to a repository, it may be desirable to include data about the publication's links to other Elements objects, and even data on the other objects themselves. For example, data on grants that funded a publication, or users that authored it.

This can be achieved with the object neighbourhood functionality. This functionality is disabled by default, but can be enabled on the **Data Source** page for the relevant repository.

The 'neighbourhood' of an object consists of the object's approved links along with the objects on the other end of those links, as shown in the image below.




In a deposit crosswalk map, the neighbourhood data can be accessed through the special fields *object.relationships* and *object.groups*, and also the special `person:resolved-user` data-part for person-list fields.

## C.1 The `person:resolved-user` data-part

For person-list fields, Elements attempts to resolve each person item in the list to a User object linked to the publication. These resolutions are what drive – in the Elements UI – the hyperlinks from author name metadata to an Elements user profile. Note that this resolution is performed by a background process which is run as part of the Synchronise scheduled job. If this job is not enabled, new records may not have their person-list fields resolved to linked users. Note that this behaviour can be configured with the "Resolve to link type" setting, available when editing the field for a specific object type. The User object which the person is resolved to is in the neighbourhood of the object, as defined above.

An Elements field of type person-list is multivalued and compound; the components of each of its values can be accessed using the `data-part` attribute on `<xwalk:field-source>` elements, as described in Section 6.2.1. To access data on the linked User (if present) for each person in the list, the special `data-part` option `person:resolved-user` can be used.

This data-part option behaves differently to others: usually the `data-part` attribute references either scalar values (e.g. text, number, etc) or subfield components (e.g. addresses on person-list fields). When the `data-part` attribute is set to `person:resolved-user`, however, the field-source references a full Elements object.

This full object has its own set of fields. These fields can then be referenced using nested field-sources with the `from` attribute. Note the distinction between the use of the `from` attribute for selecting fields, and the `data-part` attribute for selecting data components/subfields. The following example demonstrates a field-mapping which selects the email addresses of each author which has been resolved to a linked user:

```
<xwalk:field-mapping to="related-users-emails">
  <xwalk:field-source from="authors" data-part="person:resolved-user">
    <xwalk:field-source from="user.email-address"/>
  </xwalk:field-source>
</xwalk:field-mapping>
```

### C.1.1 User object field identifiers

The field identifiers available for a User object are as follows:

| User field identifiers |
| --- |
| user.id |
| user.proprietary-id |
| user.username |
| user.initials |
| user.first-name |
| user.last-name |
| user.email-address |
| user.institutional-email-is-public |
| user.primary-group-descriptor |
| user.ever-approved |
| user.is-public |
| user.is-current-staff |
| user.is-academic |
| user.is-login-allowed |
| user.title |
| user.department |
| user.claimed |
| user.public-url-path-fragment |
| user.depositor-type |
| user.genericXX |
| user.identifiers |
| user.preferred-first-name |
| user.preferred-last-name |
| user.arrive-date |
| user.leave-date |
| user.known-as |
| user.suffix |

All of these reference scalar values, except `user.identifiers`, which is of type identifier-list (see Appendix B.6). To access data in this subfield, additional nested field-sources, making use of the `data-part` or `format` attributes, may be required.

The field identifier `user.genericXX` allows specification of any generic user field, by replacing `XX` with the generic field number. For example, `user.generic04`.

If the user does not match either the logged in user or the impersonated user, then `user.depositor-type` will have no value. Otherwise, it will be equal to one of:

- `user`: if the user is the logged in user, and there is no impersonated user
- `impersonating-user`: if the user the logged in user, and that user is impersonating some other user
- `impersonated-user`: if the user the impersonated user

## C.2 The *object.relationships* field identifier

The *object.relationships* field identifier refers to a list of relationships on the object. In this context, a 'relationship' consists of the details of an approved link connected to the publication being deposited, along with data about the Elements object on the other end of the link.

The components of each relationship can be referenced with the following data-parts:

| data-part |
|---|
| `relationship:id` |
| `relationship:type` |
| `relationship:type-id` |
| `relationship:is-visible` |
| `relationship:is-favourite` |
| `relationship:direction` |
| `relationship:other-object-id` |
| `relationship:other-object-category` |
| `relationship:other-object` |

### C.2.1 The `relationship:other-object` data-part

As with the `person:resolved-user` data-part described in Appendix C.1, the `relationship:other-object` data-part value behaves differently to others, in that it references a full Elements object with its own set of fields. These fields can be referenced using nested field-sources with the `from` attribute. The following example demonstrates a field-mapping which selects the values of the *title* field for each related object:

```
<xwalk:field-mapping to="related-object-titles">
  <xwalk:field-source from="object.relationships"
  ↪  data-part="relationship:other-object">
    <xwalk:field-source from="title" />
  </xwalk:field-source>
</xwalk:field-mapping>
```

Again, note the distinction between the use of the `from` attribute for selecting fields, and the `data-part` attribute for selecting data components/subfields.

For each Elements object category (Grant, Teaching Activity, etc) excluding User, a list of valid field identifiers can be found by navigating in Elements to:

**Module Admin → [Category] → Underlying Fields**

As described in Section 2.3.1, crosswalks always use the underlying field names (e.g. *title*, *abstract*, *authors* etc) – common to all object types for the given category – not the display names. Any Elements custom fields can also be referenced – these field names always begin with "c-".

### C.2.2 Filtering relationships

When selecting data from the *object.relationships* field, it may be necessary to filter the values to obtain only relationships which fulfil some criteria. For example, selecting all Elements Users that have an authorship link to the publication being deposited, or selecting all grant objects which are linked to the publication being deposited. This can be achieved using value-maps to filter out items based on data-part (see Section 3.4).

For example, to crosswalk details about all users with an authorship link to the publication being deposited, the following value-map can be used:

```
<xwalk:value-map name="publication-authors-only" data-part="relationship:type">
  <xwalk:value-mapping from="publication-user-authorship" action="continue"/>
  <xwalk:otherwise-mapping action="ignore-this-value" />
</xwalk:value-map>
```

When invoked from a field-source with the `from` attribute set to the value `object.relationship`, this value-map filters the *object.relationships* field to include only the relationships of type 'publication-user-authorship':

The following two sections contain some examples of using the *object.relationships* field within a deposit field-mapping.

### C.2.3 Example: Related User objects

```
<xwalk:field-mapping to="author-surnames">
  <xwalk:field-source from="object.relationships"
  ↪  value-map="publication-authors-only">
    <xwalk:field-source data-part="relationship:other-object">
      <xwalk:field-source from="user.last-name" />
    </xwalk:field-source>
  </xwalk:field-source>
</xwalk:field-mapping>
```

In the above example, the `value-map` attribute references the example value-map shown in the previous section, which filters the *object.relationships* field as described previously. Next, the nested field-source uses the `data-part` attribute to select the other Elements object component of those relationships (i.e. the users that authored the publication). Finally, the innermost field-source uses the `from` attribute to select the value of the *user.last-name* field from each User object.

### C.2.4 Example: Related Grant objects

**Note:** the following example demonstrates deposit into a compound data-structure, which is only relevant for EPrints and Figshare for Institutions.

```
<xwalk:field-mapping to="funder-details">
  <xwalk:field-source from="object.relationships" value-map="grant-relationships">
    <xwalk:field-source data-part="relationship:other-object">
      <xwalk:field-source subfield="name" from="funder-name" />
      <xwalk:field-source subfield="reference" from="funder-reference" />
    </xwalk:field-source>
  </xwalk:field-source>
</xwalk:field-mapping>
```

```
<xwalk:value-map name="grant-relationships"
↪  data-part="relationship:other-object-category">
  <xwalk:value-mapping from="grant" action="continue"/>
  <xwalk:otherwise-mapping action="ignore-this-value"/>
</xwalk:value-map>
```

The above field-mapping contains a field-source which uses a value-map to select only relationships to grant objects. The inner field-sources then select the *funder-name* and *funder-reference* fields on the related grants, and use the `subfield` attribute to construct a compound object containing two subfields holding these values.

An example of the structure of the outputted XML, if depositing to EPrints, is shown below:

```
<funder-details>
  <item>
    <name>BBSRC</name>
    <reference>BB/I001662/1</reference>
  </item>
  <item>
    <name>EPSRC</name>
    <reference>EP/H500928/1</reference>
  </item>
</funder-details>
```

## C.3 The *object.groups* field identifier

The *object.groups* field contains a list of groups that the publication being deposited is related to. Specifically, this is the set of groups that any Elements users linked to the publication belong to (via approved links only).

You must use `data-part` attribute on `<xwalk:field-source>` elements to reference specific components of a group. The following data-parts are supported:

| data-part |
| --- |
| group:id |
| group:parent-id |
| group:full-name |
| group:membership-type |
| group:name |

### C.3.1 Filtering groups

As with relationships, group filtering can be achieved using value-maps. Suppose, for example, information about only the top-level related groups should be crosswalked – that is, the related groups that are children of the root group in the hierarchical structure (the root group is also known as the Organisational group).

The root group always has an ID of 1, so a value-map can be constructed to include only groups that have a parent ID of 1:

```
<xwalk:value-map name="top-level-groups" data-part="group:parent-id">
  <xwalk:value-mapping from="1" action="continue"/>
  <xwalk:otherwise-mapping action="ignore-this-value"/>
</xwalk:value-map>
```

This can be used in a field-mapping, as shown in the following example:

```
<xwalk:field-mapping to="top-level-groups">
  <xwalk:field-source from="object.groups" value-map="top-level-groups">
    <xwalk:field-source data-part="group:full-name"/>
  </xwalk:field-source>
</xwalk:field-mapping>
```

This field-mapping will output a list of the full names of all top-level groups which are related to the publication being deposited.

# D Logic expressions

Sometimes the behaviour of the crosswalk should depend on the data that is being processed. For instance, with a harvest crosswalk the object type to be created and the field-map to be used will often depend on the item being processed. This requires the ability to define logic within the crosswalk map file.

Crosswalk maps support logic expressions with **if** statements (using an `<xwalk:if>` element) and **choose** statements (using the `<xwalk:choose>` element). A logic expression uses a `<xwalk:condition>` element to define a condition to be evaluated, and a `<xwalk:result>` element to hold the result to be used when the condition is true.

Logic expressions can be used within the following crosswalk elements:

- `<xwalk:field-mapping>` (see Section 2.2)
- `<xwalk:field-map-selector>` (see Section 4.3)
- `<xwalkin:object-type-selector>` (see Section 4.2)
- `<xwalkout:collection-selector>` (see Section 4.4)

## D.1 The `<xwalk:if>` element

The `<xwalk:if>` element should contain an `<xwalk:condition>` and an `<xwalk:result>` element. The `<xwalk:condition>` element inspects the value of the input field specified by the field identifier in the `argument-field` attribute and performs the test specified by the `operator` attribute (see Appendix D.2 for a list of operators). If the condition is true, the contents of the `<xwalk:result>` element will be used.

A `<xwalk:if>` element can also contain a `<xwalk:else>` element; if the condition within the `<xwalk:if>` is not true, the result of the `<xwalk:else>` element is used.

The following example uses the `has-value` condition operator to set the destination *title* field to the value in *dc.title*, only if *dc.title* has a value.

```
<xwalk:field-mapping to="title">
  <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 following example additionally includes an `<xwalk:else>` element to set the value of the *title* field to “No title provided” if *dc.title* has no value, and to the value of *dc.title* otherwise:

```
<xwalk:field-mapping to="title">
  <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>
</xwalk:field-mapping>
```

## D.2 Condition operators

The `<xwalk:condition>` elements support the following options for the `operator` attribute:

| Option | Description |
|---|---|
| `equals` | Performs a string comparison between the field referenced by the `argument-field` attribute and the value within the element. Returning true if they are the same, otherwise false. |
| `contains` | Returns true if the field referenced by the `argument-field` attribute contains as a substring the value within the element, otherwise false. |
| `has-value` | Returns true if the field referenced by the `argument-field` attribute has one or more characters, otherwise false. |
| `not` | Applies the logical NOT operator to the result of all child `<xwalk:condition>` elements. |
| `and` | Applies the logical AND operator to the result of all child `<xwalk:condition>` elements. |
| `or` | Applies the logical OR operator to the result of all child `<xwalk:condition>` elements. |

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

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

```
<xwalk:field-mapping to="title">
  <xwalk:if>

    <!-- Applies a logical AND to the two inner conditions -->
    <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>
      <xwalk:field-source from="dc.title" />
    </xwalk:result>

  </xwalk:if>
</xwalk:field-mapping>
```

### D.3 The <xwalk:choose> element

The <xwalk:choose> element can be used to evaluate a number of conditions in sequence, and return the result of the first condition which is true. The conditions to be evaluated are identified by <xwalk:when> elements, with a final optional <xwalk:otherwise> element which can contain a result to be used 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 a harvest crosswalk.

```
<xwalkin:object-type-selector>

  <xwalk:choose>

    <!-- Used if the "dc.type" field is equal to "Article" -->
    <xwalk:when>
      <xwalk:condition argument-field="dc.type"
      ↪  operator="equals">Article</xwalk:condition>
      <xwalk:result>
        <xwalkin:object-type-selection category="publication"
        ↪  object-type="journal-article" field-map="journal-article-map" />
      </xwalk:result>
    </xwalk:when>

    <!-- Used if the "dc.type" field is equal to "Book" -->
    <xwalk:when>
      <xwalk:condition argument-field="dc.type"
      ↪  operator="equals">Book</xwalk:condition>
      <xwalk:result>
```

```
        <xwalkin:object-type-selection category="publication" object-type="book"
        ↪  field-map="book-map" />
      </xwalk:result>
    </xwalk:when>

    <!-- Used if the "dc.type" field is not equal to either "Article" or "Book" -->
    <xwalk:otherwise>
      <xwalk:result>
        <xwalkin:object-type-selection category="publication"
        ↪  object-type="conference" field-map="conference-map" />
      </xwalk:result>
    </xwalk:otherwise>

  </xwalk:choose>

</xwalkin:object-type-selector>
```

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

# E Subsequent deposit

Elements has the ability to perform "subsequent" deposits to EPrints, DSpace and Hyrax. In this operation, files can be added via Elements to items which already exist in the destination systems. Elements will upload the files to the existing items, and optionally perform some other actions. The configuration of those other actions are described in this section.

## E.1 DSpace

Any files which are added in a subsequent deposit operation to DSpace will have all read access policies removed. This prevents new files being sent straight to Live. In order to flag these items as having new content requiring review, additional item-level metadata can be supplied.

To achieve this, the field-map-selector should select a specific field-map in the case that the field *deposit.type* is equal to "subsequent" (see Section 6.1.10 for more details). Additionally, the DSpace parameter `dspace-xwalkout-enable-metadata-post-for-redeposit` should be set to "true" to specify that the result of the crosswalk should be added to the DSpace item (see Section 4.1.3).

## E.2 EPrints

The files added to an EPrints item via subsequent deposit will be adjusted by the configured file-metadata-map (see Section 4.5). A default file-metadata-map takes effect if none is specified. When performing a subsequent deposit, this default sets the *security* document-level field for the new files to "staffonly", and adds some descriptive text, including the name of the depositing user, to the *formatdesc* field.

If the file-metadata-map is explicitly specified, then any subsequent-deposit-specific behaviour will have to be manually specified, by checking the value of the *deposit.type* field (see Section 6.1.10). Note that if a file-metadata-map is specified, the default will not be used; if File Versions and Reuse Licences are used, the file-metadata-map override will need to reimplement the mappings of these values.

# F Automated Metadata Push to Repositories

Elements supports automated metadata push of publications to EPrints and DSpace. This operation is performed by the synchroniser and its purpose is to continuously update repository items with Elements object metadata. The reindexer uses relevance schemes to identify which objects are relevant for the push and the synchroniser performs the push. It is vital for the **harvest crosswalk**, **deposit crosswalk** and **relevance scheme** files to be configured correctly for the push to succeed.

## F.1 RT2 Push Process

The synchroniser retrieves the objects from Elements's operational database and performs the push in batches. It is hardcoded within the synchroniser to ignore objects that are not publications or objects that do not contain an active record (see Appendix F.2).

Each object is masked by the relevance scheme followed by an application of deposit crosswalk. The crosswalked object metadata is merged with the repository item, thus updating the item with Elements metadata. The merge process consists of pulling the item metadata from the repository, merging the repository item metadata with Elements's metadata using the field authority list (see Section 4.6) and pushing the merged metadata back to the repository.

The updated metadata at the repository is immediately fetched back into Elements's operational database (which uses the harvest crosswalk) to keep Elements metadata aligned with the repository metadata.

## F.2 Active Record

Elements performs metadata push to the most active record at the repository. An active record is defined as an item that is not withdrawn or deleted. The most active record is the most recently deposited active record, according to Elements's database. If, during the push, the most active record was found to be inactive, the second-most active record will be used. If the second-most active record is found inactive, then the third-most active record is used. This continues until no active records are left, which will force the synchroniser to ignore the object.