Eberlein Consulting

DITA training, information architecture, and strategy

Subject scheme maps

Contents

Subject scheme maps and their usage

Subject scheme maps can be used to define controlled values, taxonomies, and ontologies. The controlled values can be bound to attributes, as well as element and attribute pairs. The taxonomic subjects can contain metadata and provide links to more detailed information; the taxonomic subjects can be used to classify content.

A DITA map can reference a subject scheme map by using a `<mapref>` element. Processors also *MAY* provide parameters by which subject scheme maps are referenced.

Subject scheme maps

Subject scheme maps use key definitions to define taxonomic subjects and collections of controlled values.

Controlled values are keywords that can be used as values for attributes. For example, the `@audience` attribute can take a value that identifies the users that are associated with a particular product. Typical values for a medical-equipment product line might include "therapist", "oncologist", "physicist", and "radiologist". In a subject scheme map, an information architect can define a list of these values for the `@audience` attribute. Controlled values can be used to classify content for filtering and flagging at build time.

Taxonomic subjects are classifications and sub-classifications that compose a taxonomy tree. In conjunction with the classification domain, taxonomic subjects can be used for retrieval and traversal of the content at run time when used with information viewing applications that provide such functionality.

Key references to controlled values are resolved to a key definition using the same precedence rules as apply to any other key. However, once a key is resolved to a controlled value, that key reference does not typically result in links or generated text.

Defining controlled values for attributes

Subject scheme maps can define controlled values for DITA attributes without having to define specializations or constraints. The list of available values can be modified quickly to adapt to new situations.

Each controlled value is defined using a `<subjectdef>` element, which is a specialization of the `<topicref>` element. The `<subjectdef>` element is used to define both a subject category and a list of controlled values. The top-level `<subjectdef>` element defines the category, and the children `<subjectdef>` elements define the controlled values.

The subject definitions can include additional information within a `<topicmeta>` element to clarify the meaning of a value:

- The `<navtitle>` element can provide a more readable value name.
- The `<shortdesc>` element can provide a definition.

In addition, the `<subjectdef>` element can use an `@href` attribute to refer to a more detailed definition of the subject.

Comment by Kristen Eberlein on 5 February 2015

In the targeted review, Eliot Kimber suggested revising the above sentence as: "In addition, the `<subjectdef>` element can refer to a more detailed definition of the subject.

I am leaving the sentence as is, as I think the original wording is correct. I suspect that using `@keyref` in a subjectScheme map to refer to other than a subject might cause circular processing.

The following behavior is expected of processors:

- Authoring tools *SHOULD* use these lists of controlled values to provide lists from which authors can select values when they specify attribute values.
- Authoring tools *MAY* give an organization a list of readable labels, a hierarchy of values to simplify selection, and a shared definition of the value.
- An editor *MAY* support accessing and displaying the content of the subject definition resource in order to provide users with a detailed explanation of the subject.
- Tools *MAY* produce a help file, PDF, or other readable catalog to help authors better understand the controlled values.

Example: Controlled values that provide additional information about the subject

The following code fragment illustrates how a subject definition can provide a richer level of information about a controlled value:

```
<subjectdef keys="terminology" href="https://www.oasis-open.org/policies-guidelines/
keyword-guidelines">
    <subjectdef keys="rfc2119" href="rfc-2119.dita">
        <topicmeta>
            <navtitle>RFC-2119 terminology</navtitle>
            <shortdesc>The normative terminology that the DITA TC uses for the DITA
specification</shortdesc>
        </topicmeta>
    </subjectdef>
    <subjectdef keys="iso" href="iso-terminology.dita">
        <topicmeta>
            <navtitle>ISO keywords</navtitle>
            <shortdesc>The normative terminology used by some other OASIS technical
committees</shortdesc>
        <topicmeta>
    </subjectdef>
</subjectdef>
```

The content of the `<navtitle>` and `<shortdesc>` elements provide additional information that a processor might display to users as they select attribute values or classify content. The resources referenced by the `@href` attributes provide even more detailed information; a processor might render clickable links as part of a user interface that implements a progressive disclosure strategy

Binding controlled values to an attribute

The controlled values defined in a subject scheme map can be bound to an attribute or an element and attribute pair. This affects the expected behavior for processors and authoring tools.

The `<enumerationdef>` element binds the set of controlled values to an attribute. Valid attribute values are those that are defined in the set of controlled values; invalid attribute values are those that are not defined in the set of controlled values. An enumeration can specify an empty `<subjectdef>` element. In that case, no value is valid for the attribute. An enumeration also can specify an optional default value by using the `<defaultSubject>` element.

If an enumeration is bound, processors *SHOULD* validate attribute values against the controlled values that are defined in the subject scheme map. For authoring tools, this validation prevents users from entering misspelled or undefined values. Recovery from validation errors is implementation specific.

The default attribute values that are specified in a subject scheme map apply only if a value is not otherwise specified in the DITA source or as a default value by the XML grammar.

To determine the effective value for a DITA attribute, processors check for the following in the order outlined:

1. An explicit value in the element instance
2. A default value in the XML grammar
3. Cascaded value within the document
4. Cascaded value from a higher level document to the document
5. A default controlled value, as specified in the `<defaultSubject>` element
6. A value set by processing rules

Example: Binding a list of controlled values to the `@audience` attribute

The following example illustrates the use of the `<subjectdef>` element to define controlled values for types of users. It also binds the controlled values to the `@audience` attribute:

```
<subjectScheme>
<!-- Define types of users -->
    <subjectdef keys="users">
        <subjectdef keys="therapist"/>
        <subjectdef keys="oncologist"/>
        <subjectdef keys="physicist"/>
        <subjectdef keys="radiologist"/>
    </subjectdef>

 <!-- Bind the "users" subject to the @audience attribute.
      This restricts the @audience attribute to the following
      values: therapist, oncologist, physicist, radiologist -->
     <enumerationdef>
         <attributedef name="audience"/>
         <subjectdef keyref="users"/>
     </enumerationdef>
</subjectScheme>
```

When the above subject scheme map is used, the only valid values for the `@audience` attribute are "therapist", "oncologist", "physicist", and "radiologist". Note that "users" is not a valid value for the `@audience` attribute; it merely identifies the parent or container subject.

Example: Binding an attribute to an empty set

The following code fragment declares that there are no valid values for the `@outputclass` attribute.

```
<subjectScheme>
    <enumerationdef>
        <attributedef name="outputclass"/>
        <subjectdef/>
    </enumerationdef>
</subjectScheme>
```

Processing controlled attribute values

An enumeration of controlled values can be defined with hierarchical levels by nesting subject definitions. This affects how processors perform filtering and flagging.

The following algorithm applies when processors apply filtering and flagging rules to attribute values that are defined as a hierarchy of controlled values and bound to an enumeration:

1. If an attribute specifies a value in the taxonomy, and a DITAVAL or other categorization tool is configured with that value, the rule matches.
2. Otherwise, if the parent value in the taxonomy has a rule, that matches.

3. Otherwise, continue up the chain in the taxonomy until a matching rule is found.

The following behavior is expected of processors:

- Processors *SHOULD* be aware of hierarchies of attribute values that are defined in subject scheme maps for purposes of filtering, flaggging, or other metadata-based categorization.
- Processors *SHOULD* validate that the values of attributes that are bound to controlled values contain only valid values from those sets. (The list of controlled values is not validated by basic XML parsers.)
- Processors *SHOULD* check that all values listed for an attribute in a DITAVAL file are bound to the attribute by the subject scheme before filtering or flagging. If a processor encounters values that are not included in the subject scheme, it *SHOULD* issue a warning.

Example: A hierarchy of controlled values and conditional processing

The following example illustrates a set of controlled values that contains a hierarchy.

```
<subjectScheme>
    <subjectdef keys="users">
        <subjectdef keys="therapist">
            <subjectdef keys="novice-therapist"/>
            <subjectdef keys="expert-therapist"/>
        </subjectdef>
        <subjectdef keys="oncologist"/>
        <subjectdef keys="physicist"/>
        <subjectdef keys="radiologist"/>
    </subjectdef>
    <enumerationdef>
        <attributedef name="audience"/>
        <subjectdef keyref="users"/>
    </enumerationdef>
</subjectScheme>
```

Processors that are aware of the hierarchy that is defined in the subject scheme map will handle filtering and flagging in the following ways:

- If "therapist" is excluded, both "novice-therapist" and "expert-therapist" are by default excluded (unless they are explicitly set to be included).
- If "therapist" is flagged and "novice-therapist" is not explicitly flagged, processors automatically should flag "novice" since it is a type of therapist.

Extending subject schemes

The `<schemeref>` element provides a mechanism for extending a subject scheme. This makes it possible to add new relationships to existing subjects and extend enumerations of controlled values.

The `<schemeref>` element provides a reference to another subject scheme map. Typically, the referenced subject-scheme map defines a base set of controlled values that are extended by the current subject-scheme map. The values in the referenced subject-scheme map are merged with the values in the current subject-scheme map; the result is equivalent to specifying all of the values in a single subject scheme map.

Comment by Kristen Eberlein on 5 February 2015

I added this topic to address comments made by Chris Nitchie in the targeted review. FYI, the paragraph above was in the DITA 1.2 Language Reference topic about `<schemeref>`.

Re *Example: Extending a subject scheme* on page 10, Chris stated "That's not how keyref normally works. We definitely need normative language describing the expected processing of keyref in the context of a subject scheme. It's completely different from normal keyref behavior. Normally, a keyref is a reference to the content

referenced by the key-defining topicref, not a reference to the topicref itself. And there's no transclusion or extension, as is implied here. So it's completely different processing rules for keys in a subject scheme, and it needs normative description."

Re *Example: Extending a subject scheme upwards* on page 11, Chris stated " ... the use of keyref in subject schemes to recombine subject definitions is never covered anywhere."

I suspect that we need additional normative content in this topic ...

Scaling a list of controlled values to define a taxonomy

Optional classification elements make it possible to create a taxonomy from a list of controlled values.

A taxonomy differs from a controlled values list primarily in the degree of precision with which the metadata values are defined. A controlled values list sometimes is regarded as the simplest form of taxonomy. Regardless of whether the goal is a simple list of controlled values or a taxonomy:

- The same core elements are used: `<subjectScheme>` and `<subjectdef>`.
- A category and its subjects can have a binding that enumerates the values of an attribute.

Beyond the core elements and the attribute binding elements, sophisticated taxonomies can take advantage of some optional elements. These optional elements make it possible to specify more precise relationships among subjects. The `<hasNarrower>`, `<hasPart>`, `<hasKind>`, `<hasInstance>`, and `<hasRelated>` elements specify the kind of relationship in a hierarchy between a container subject and its contained subjects.

While users who have access to sophisticated processing tools benefit from defining taxonomies with this level of precision, other users can safely ignore this advanced markup and define taxonomies with hierarchies of `<subjectdef>` elements that are not precise about the kind of relationship between the subjects.

Example: A taxonomy defined using subject scheme elements

The following example defines San Francisco as both an instance of a city and a geographic part of California.

```
<subjectScheme>
    <hasInstance>
        <subjectdef keys="city" navtitle="City">
            <subjectdef keys="la" navtitle="Los Angeles"/>
            <subjectdef keys="nyc" navtitle="New York City"/>
            <subjectdef keys="sf" navtitle="San Francisco"/>
        </subjectdef>
        <subjectdef keys="state" navtitle="State">
            <subjectdef keys="ca" navtitle="California"/>
            <subjectdef keys="ny" navtitle="New York"/>
        </subjectdef>
    </hasInstance>
    <hasPart>
        <subjectdef keys="place" navtitle="Place">
            <subjectdef keyref="ca">
                <subjectdef keyref="la"/>
                <subjectdef keyref="sf"/>
            </subjectdef>
            <subjectdef keyref="ny">
                <subjectdef keyref="nyc"/>
            </subjectdef>
        </subjectdef>
    </hasPart>
</subjectScheme>
```

Sophisticated tools can use this subject scheme map to associate content about San Francisco with related content about other California places or with related content about other cities (depending on the interests of the current user).

The subject scheme map also can define relationships between subjects that are not hierarchical. For instance, cities sometimes have "sister city" relationships. An information architect could add a `<subjectRelTable>` element to define these associative relationships, with a row for each sister-city pair and the two cities in different columns in the row.

Examples of subject scheme maps

This section contains examples and scenarios that illustrate the use of subject scheme maps.

Example: How hierarchies defined in a subject scheme map affect filtering

This scenario demonstrates how a processor evaluates attribute values when it performs conditional processing for an attribute that is bound to a set of controlled values.

An company defines a subject category for "Operating system," with a key set to "os. There are sub-categories for Linux, Windows, and z/OS, as well as specific Linux variants: Red Hat Linux and SuSE Linux. The company then binds the values that are enumerated in the "Operating system" category to the `@platform` attribute.

```
<!-- This examples uses @navtitle rather than <navtitle> solely
to conserve space. Best practises for translate include using <navtitle>. -->
<subjectScheme>
  <subjectdef keys="os" navtitle="Operating system">
    <subjectdef keys="linux" navtitle="Linux">
      <subjectdef keys="redhat" navtitle="RedHat Linux"/>
      <subjectdef keys="suse"   navtitle="SuSE Linux"/>
    </subjectdef>
    <subjectdef keys="windows" navtitle="Windows"/>
    <subjectdef keys="zos"   navtitle="z/OS"/>
  </subjectdef>
  <enumerationdef>
    <attributedef name="platform"/>
    <subjectdef keyref="os"/>
  </enumerationdef>
</subjectScheme>
```

The enumeration limits valid values for the `@platform` attribute to the following: "linux", "redhat", "suse", "windows", and "zos". If any other values are encountered, processors validating against the scheme should issue a warning.

The following table illustrates how filtering and flagging operate when the above map is processed by a processor. The first two columns provide the values specified in the DITAVAL file; the third and fourth columns indicate the results of the filtering or flagging operation

att="platform" val="linux"	att="platform" val="redhat"	How platform="redhat" is evaluated	How platform="linux" is evaluated
action="exclude"	action="exclude""	Excluded.	Excluded.
	action="include"or action="flag"	Excluded. This is an error condition, because if all "linux" content is	Excluded.

att="platform" val="linux"	att="platform" val="redhat"	How platform="redhat" is evaluated	How platform="linux" is evaluated
		excluded, "redhat" also is excluded. Applications may recover by generating an error message.	
	Unspecified	Excluded, because "redhat" is a kind of "linux", and "linux" is excluded.	Excluded.
action="include"	action="exclude"	Excluded, because all "redhat" content is excluded.	Included.
	action="include"	Included.	Included.
	action="flag"	Included and flagged with the "redhat" flag.	Included.
	Unspecified	Included, because all "linux" content is included.	Included.
action="flag"	action="exclude"	Excluded, because all "redhat" content is excluded.	Included and flagged with the "linux" flag.
	action="include"	Included and flagged with the "linux" flag, because "linux" is flagged and "redhat" is a type of "linux".	Included and flagged with the "linux" flag.
	action="flag"	Included and flagged with the "redhat" flag, because a flag is available that is specifically for "redhat".	Included and flagged with the "linux" flag.
	Unspecified	Included and flagged with the "linux" flag, because "linux" is flagged and "redhat" is a type of linux	Included and flagged with the "linux" flag.
Unspecified	action="exclude"	Excluded, because all "redhat" content is excluded	If the default for `@platform` values is "include", this is included. If the default for `@platform` values is "exclude", this is excluded.
	action="include"	Included.	Included, because all "redhat" content is included, and general

att="platform" val="linux"	att="platform" val="redhat"	How platform="redhat" is evaluated	How platform="linux" is evaluated
			Linux content also applies to RedHat
	action="flag"	Included and flagged with the "redhat" flag.	Included, because all "redhat" content is included, and general Linux content also applies to RedHat
	Unspecified	If the default for `@platform` values is "include", this is included. If the default for `@platform` values is "exclude", this is excluded.	If the default for `@platform` values is "include", this is included. If the default for `@platform` values is "exclude", this is excluded.

Example: Extending a subject scheme

You can extend a subject scheme by creating another subject scheme map and referencing the original map using a `<schemeref>` element. This enables information architects to add new relationships to existing subjects and extend enumerations of controlled values.

A company uses a common subject scheme map (`baseOS.ditamap`) to set the values for the `@platform` attribute.

```
<subjectScheme>
  <subjectdef keys="os" navtitle="Operating system">
    <subjectdef keys="linux" navtitle="Linux">
      <subjectdef keys="redhat" navtitle="RedHat Linux"/>
      <subjectdef keys="suse" navtitle="SuSE Linux"/>
    </subjectdef>
    <subjectdef keys="windows" navtitle="Windows"/>
    <subjectdef keys="zos" navtitle="z/OS"/>
  </subjectdef>
  <enumerationdef>
    <attributedef name="platform"/>
    <subjectdef keyref="os"/>
  </enumerationdef>
</subjectScheme>
```

The following subject scheme map extends the enumeration defined in `baseOS.ditamap`. It adds "macos" as a child of the existing "os" subject; it also adds special versions of Windows as children of the existing "windows" subject:

```
<subjectScheme>
  <schemeref href="baseOS.ditamap"/>
  <subjectdef keyref="os">
    <subjectdef keys="macos" navtitle="Macintosh"/>
    <subjectdef keyref="windows">
      <subjectdef keys="winxp" navtitle="Windows XP"/>
      <subjectdef keys="winvis" navtitle="Windows Vista"/>
    </subjectdef>
```

```
  </subjectdef>
</subjectScheme>
```

Note that the references to the subjects that are defined in `baseOS.ditamap` use the `@keyref` attribute. This avoids duplicate definitions of the keys and ensures that the new subjects are added to the base enumeration.

The effective result is the same as the following subject scheme map:

```
<subjectScheme>
  <subjectdef keys="os" navtitle="Operating system">
    <subjectdef keys="linux" navtitle="Linux">
      <subjectdef keys="redhat" navtitle="RedHat Linux"/>
      <subjectdef keys="suse" navtitle="SuSE Linux"/>
    </subjectdef>
    <subjectdef keys="macos" navtitle="Macintosh"/>
    <subjectdef keys="windows" navtitle="Windows">
      <subjectdef keys="winxp" navtitle="Windows XP"/>
      <subjectdef keys="win98" navtitle="Windows Vista"/>
    </subjectdef>
    <subjectdef keys="zos" navtitle="z/OS"/>
  </subjectdef>
  <enumerationdef>
    <attributedef name="platform"/>
    <subjectdef keyref="os"/>
  </enumerationdef>
</subjectScheme>
```

Example: Extending a subject scheme upwards

You can broaden the scope of a subject category by creating a new subject scheme map that defines the original subject category as a child of a broader category.

The following subject scheme map creates a "Software" category that includes operating systems as well as applications. The subject scheme map that defines the operation system subjects is pulled in by reference, while the application subjects are defined directly in the subject scheme map below.

```
<subjectScheme>
  <schemeref href="baseOS.ditamap"/>
  <subjectdef keys="sw" navtitle="Software">
    <subjectdef keyref="os"/>
    <subjectdef keys="app" navtitle="Applications">
      <subjectdef keys="apacheserv" navtitle="Apache Web Server"/>
      <subjectdef keys="mysql" navtitle="MySQL Database"/>
    </subjectdef>
  </subjectdef>
</subjectScheme>
```

If the subject scheme that is defined in `baseOS.ditamap` binds the "os" subject to the `@platform` attribute, the app subjects that are defined in the extension subject scheme do not become part of that enumeration, since they are not part of the "os" subject

To enable the upward extension of an enumeration, information architects can define the controlled values in one subject scheme map and bind the controlled values to the attribute in another subject scheme map. This approach will let information architects bind an attribute to a different set of controlled values with less rework.

An adopter would use the extension subject scheme as the subject scheme that governs the controlled values. Any subject scheme maps that are referenced by the extension subject scheme are effectively part of the extension subject scheme.

Example: Defining values for `@deliveryTarget`

You can use a subject scheme map to define the values for the `@deliveryTarget` attribute. This filtering attribute, which is new in DITA 1.3, is intended for use with a set of hierarchical, controlled values.

In this scenario, one department produces electronic publications (EPUB, EPUB2, EPUB3, Kindle, etc.) while another department produces traditional, print-focused output. Each department needs to exclude a certain category of content when they build documentation deliverables.

The following subject scheme map provides a set of values for the `@deliveryTarget` attribute that accommodates the needs of both departments.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE subjectScheme PUBLIC "-//OASIS//DTD DITA Subject Scheme Map//EN"
"subjectScheme.dtd">
<subjectScheme>
  <subjectHead>
    <subjectHeadMeta>
      <navtitle>Example of values for the @deliveryTarget attribute</navtitle>
      <shortdesc>Provides a set of values for use with the
        @deliveryTarget conditional-processing attribute. This set of values is
        illustrative only; you can use any values with the @deliveryTarget
        attribute.</shortdesc>
    </subjectHeadMeta>
  </subjectHead>
  <subjectdef keys="deliveryTargetValues">
    <topicmeta><navtitle>Values for @deliveryTarget attributes</navtitle></topicmeta>
    <!-- A tree of related values -->
    <subjectdef keys="print">
      <topicmeta><navtitle>Print-primary deliverables</navtitle></topicmeta>
      <subjectdef keys="pdf">
        <topicmeta><navtitle>PDF</navtitle></topicmeta>
      </subjectdef>
      <subjectdef keys="css-print">
        <topicmeta><navtitle>CSS for print</navtitle></topicmeta>
      </subjectdef>
      <subjectdef keys="xsl-fo">
        <topicmeta><navtitle>XSL-FO</navtitle></topicmeta>
      </subjectdef>
      <subjectdef keys="afp">
        <topicmeta><navtitle>Advanced Function Printing</navtitle></topicmeta>
      </subjectdef>
      <subjectdef keys="ms-word">
        <topicmeta><navtitle>Microsoft Word</navtitle></topicmeta>
      </subjectdef>
      <subjectdef keys="indesign">
        <topicmeta><navtitle>Adobe InDesign</navtitle></topicmeta>
      </subjectdef>
      <subjectdef keys="open-office">
        <topicmeta><navtitle>Open Office</navtitle></topicmeta>
      </subjectdef>
    </subjectdef>
    <subjectdef keys="online">
      <topicmeta><navtitle>Online deliverables</navtitle></topicmeta>
      <subjectdef keys="html-based">
        <topicmeta><navtitle>HTML-based deliverables</navtitle></topicmeta>
        <subjectdef keys="html">
          <topicmeta><navtitle>HTML</navtitle></topicmeta>
          <subjectdef keys="html5">
            <topicmeta><navtitle>HTML5</navtitle></topicmeta>
          </subjectdef>
        </subjectdef>
        <subjectdef keys="help">
          <topicmeta><navtitle>Contextual help</navtitle></topicmeta>
```

```
          <subjectdef keys="htmlhelp">
            <topicmeta><navtitle>HTML Help</navtitle></topicmeta>
          </subjectdef>
          <subjectdef keys="webhelp">
            <topicmeta><navtitle>Web help</navtitle></topicmeta>
          </subjectdef>
          <subjectdef keys="javahelp">
            <topicmeta><navtitle>Java Help</navtitle></topicmeta>
          </subjectdef>
          <subjectdef keys="eclipseinfocenter">
            <topicmeta><navtitle>Eclipse InfoCenter</navtitle></topicmeta>
          </subjectdef>
        </subjectdef>
        <subjectdef keys="epub">
          <topicmeta><navtitle>EPUB</navtitle></topicmeta>
          <subjectdef keys="epub2">
            <topicmeta><navtitle>EPUB2</navtitle></topicmeta>
          </subjectdef>
          <subjectdef keys="epub3">
            <topicmeta><navtitle>EPUB3</navtitle></topicmeta>
          </subjectdef>
          <subjectdef keys="ibooks">
            <topicmeta><navtitle>iBooks</navtitle></topicmeta>
          </subjectdef>
          <subjectdef keys="nook">
            <topicmeta><navtitle>nook</navtitle></topicmeta>
          </subjectdef>
        </subjectdef>
        <subjectdef keys="kindle">
          <topicmeta><navtitle>Amazon Kindle</navtitle></topicmeta>
          <subjectdef keys="kindle8">
            <topicmeta><navtitle>Kindle Version 8</navtitle></topicmeta>
          </subjectdef>
        </subjectdef>
      </subjectdef>
    </subjectdef>
  </subjectdef>
  <enumerationdef>
    <attributedef name="deliveryTarget"/>
    <subjectdef   keyref="deliveryTargetValues"/>
  </enumerationdef>
</subjectScheme>
```

Index