dita13_archspec_examples

Contents

URI-based (direct) addressing

Content reference and link relationships can be established from DITA elements by using URI references. DITA uses URI references in href, conref, and other attributes for all direct addressing of resources.

URI references address *resources* and (in some cases) subcomponents of those resources. In this context, a resource is a DITA document (map, topic, or DITA base document) or a non-DITA resource (for example, an image, a Web page, or a PDF document).

URI references that are URLs must conform to the rules for URLs and URIs. Windows paths that contain a backslash (\) are not valid URLs.

URIs and fragment identifiers

For DITA resources, fragment identifiers can be used with the URI to address individual elements. The fragment identifier is the part of the URI that starts with a number sign (#), for example, `#topicid/elementid`. URI references also can include a query component that is introduced with a question mark (?). DITA processors *MAY* ignore queries on URI references to DITA resources. URI references that address components in the same document *MAY* consist of just the fragment identifier.

For addressing DITA elements within maps and topics or individual topics within documents containing multiple topics, URI references must include the appropriate DITA-defined fragment identifier. URI references can be relative or absolute. A relative URI reference can consist of just a fragment identifier. Such a reference is a reference to the document that contains the reference.

Addressing non-DITA targets using a URI

DITA can use URI references to directly address non-DITA resources. Any fragment identifier used must conform to the fragment identifier requirements that are defined for the target media type or provided by processors.

Addressing elements within maps using a URI

When addressing elements within maps, URI references can include a fragment identifier that includes the ID of the map element, for example, `filename.ditamap#mapId` or `#mapId`. The same-topic, URI-reference fragment identifier of a period (.) can not be used in URI references to elements within maps.

Addressing topics using a URI

When addressing a DITA topic element, URI references can include a fragment identifier that includes the ID of the topic element (`filename.dita#topicId` or `#topicId`). When addressing the DITA topic element that contains the URI reference, the URI reference might include the same topic fragment identifier of "." (`#.`).

Topics always can be addressed by a URI reference whose fragment identifier consists of the topic ID. For the purposes of linking, a reference to a topic-containing document addresses the first topic within that document in document order. For the purposes of rendering, a reference to a topic-containing document addresses the root element of the document.

Consider the following examples:

- Given a document whose root element is a topic, a URI reference (with no fragment identifier) that addresses that document implicitly references the topic element.
- Given a dita document that contains multiple topics, for the purposes of linking, a URI reference that addresses the dita document implicitly references the first child topic.

- Given a dita document that contains multiple topics, for the purposes of rendering, a URI reference that addresses the dita document implicitly references all the topics that are contained by the dita element. This means that all the topics that are contained by thedita element are rendered in the result.

Addressing non-topic elements using a URI

When addressing a non-topic element within a DITA topic, a URI reference must use a fragment identifier that contains the ID of the ancestor topic element of the non-topic element being referenced, a slash ("/"), and the ID of the non-topic element (`filename.dita#topicId/elementId` or `#topicId/elementId`). When addressing a non-topic element within the topic that contains the URI reference, the URI reference can use an abbreviated fragment-identifier syntax that replaces the topic ID with "." (`#./elementId`).

This addressing model makes it possible to reliably address elements that have values for the id attribute that are unique within a single DITA topic, but which might not be unique within a larger XML document that contains multiple DITA topics.

Examples: URI reference syntax

The following table shows the URI syntax for common use cases.

Use case	Sample syntax
Reference a table in a topic at a network location	`"http://example.com/file.dita#topicID/tableID"`
Reference a section in a topic on a local file system	`"directory/file.dita#topicID/sectionID"`
Reference a figure contained in the same XML document	`"#topicID/figureID"`
Reference a figure contained in the same topic of an XML document	`"#./figureID"`
Reference an element within a map	`"http://example.com/map.ditamap#elementID"` (and a value of "ditamap" for the format attribute)
Reference a map element within the same map document	`"#elementID"` (and a value of "ditamap" for the format attribute)
Reference an external Web site	`"http://www.example.com"`, `"http://www.example.com#somefragment"` or any other valid URI
Reference an element within a local map	`"filename.ditamap#elementid"` (and a value of "ditamap" for the format attribute)
Reference a local map	`"filename.ditamap"` (and a value of "ditamap" for the format attribute)
Reference a local topic	Reference a local topic `"filename.dita"` or "`path/filename.dita"`

Use case	Sample syntax
Reference a specific topic in a local document	`"filename.dita#topicid"` or `"path/filename.dita#topicid"`
Reference a specific topic in the same file	`"#topicid"`
Reference the same topic in the same XML document	`"#."`
Reference a peer map for cross-deliverable linking	`"../book-b/book-b.ditamap"` (and a value of "ditamap" for the format attribute, a value of "peer" for the scope attribute, and a value for the keyscope attribute)

Chunking

Content can be chunked (divided or merged into new output documents) in different ways for the purposes of delivering content and navigation. For example, content best authored as a set of separate topics might need to be delivered as a single Web page. A map author can use the chunk attribute to split up multi-topic documents into component topics or to combine multiple topics into a single document as part of output processing.

The chunk attribute is commonly used for the following use cases.

Reuse of a nested topic
A content provider creates a set of topics as a single document. Another user wants to incorporate only one of the nested topics from the document. The new user can reference the nested topic from a DITA map, using the chunk attribute to specify that the topic should be produced in its own document.

Identification of a set of topics as a unit
A curriculum developer wants to compose a lesson for a SCORM LMS (Learning Management System) from a set of topics without constraining reuse of those topics. The LMS can save and restore the learner's progress through the lesson if the lesson is identified as a referenceable unit. The curriculum developer defines the collection of topics with a DITA map, using the chunk attribute to identify the learning module as a unit before generating the SCORM manifest.

Chunking examples

The following examples cover many common chunking scenarios, such as splitting one document into many rendered objects or merging many documents into one rendered object.

In the examples below, an extension of ".xxxx" is used in place of the actual extensions that will vary by output format. For example, when the output format is HTML, the extension might actually be ".html", but this is not required.

The examples below assume the existence of the following files:

- `parent1.dita`, `parent2.dita`, etc., each containing a single topic with id P1, P2, etc.
- `child1.dita`, `child2.dita`, etc., each containing a single topic with id C1, C2, etc.
- `grandchild1.dita`, `grandchild2.dita`, etc., each containing a single topic with id GC1, GC2, etc.
- `nested1.dita`, `nested2.dita`, etc., each containing two topics: parent topics with id N1, N2, etc., and child topics with ids N1a, N2a, etc.
- `ditabase.dita`, with the following contents:

```
<dita xml:lang="en-us">
  <topic id="X">
    <title>Topic X</title><body><p>content</p></body>
  </topic>
  <topic id="Y">
    <title>Topic Y</title><body><p>content</p></body>
    <topic id="Y1">
      <title>Topic Y1</title><body><p>content</p></body>
      <topic id="Y1a">
       <title>Topic Y1a</title><body><p>content</p></body>
      </topic>
    </topic>
    <topic id="Y2">
      <title>Topic Y2</title><body><p>content</p></body>
    </topic>
  </topic>
  <topic id="Z">
    <title>Topic Z</title><body><p>content</p></body>
    <topic id="Z1">
      <title>Topic Z1</title><body><p>content</p></body>
    </topic>
  </topic>
</dita>
```

1. The following map causes the entire map to generate a single output chunk.

   ```
   <map chunk="to-content">
      <topicref href="parent1.dita">
         <topicref href="child1.dita"/>
         <topicref href="child2.dita"/>
      </topicref>
   </map>
   ```

2. The following map will generate a separate chunk for every topic in every document referenced by the map. In this case, it will result in the topics `P1.xxxx`, `N1.xxxx`, and `N1a.xxxx`.

   ```
   <map chunk="by-topic">
      <topicref href="parent1.dita">
         <topicref href="nested1.dita"/>
      </topicref>
   </map>
   ```

3. The following map will generate two chunks: `parent1.xxxx` will contain only topic P1, while `child1.xxxx` will contain topic C1, with topics GC1 and GC2 nested within C1.

   ```
   <map>
      <topicref href="parent1.dita">
         <topicref href="child1.dita" chunk="to-content">
           <topicref href="grandchild1.dita"/>
           <topicref href="grandchild2.dita"/>
         </topicref>
      </topicref>
   </map>
   ```

4. The following map breaks down portions of `ditabase.dita` into three chunks. The first chunk `Y.xxxx` will contain only the single topic Y. The second chunk `Y1.xxxx` will contain the topic Y1 along with its child Y1a. The final chunk `Y2.xxxx` will contain only the topic Y2. For navigation purposes, the chunks for Y1 and Y2 are still nested within the chunk for Y.

   ```
   <map>
     <topicref href="ditabase.dita#Y" copy-to="Y.dita"
               chunk="to-content select-topic">
       <topicref href="ditabase.dita#Y1" copy-to="Y1.dita"
                 chunk="to-content select-branch"/>
       <topicref href="ditabase.dita#Y2" copy-to="Y2.dita"
                 chunk="to-content select-topic"/>
     </topicref>
   </map>
   ```

5. The following map will produce a single output chunk named `parent1.xxxx`, containing topic P1, with topic Y1 nested within P1, but without topic Y1a.

   ```
   <map chunk="by-document">
      <topicref href="parent1.dita" chunk="to-content">
         <topicref href="ditabase.dita#Y1"
            chunk="select-topic"/>
      </topicref>
   </map>
   ```

6. The following map will produce a single output chunk, `parent1.xxxx`, containing topic P1, topic Y1 nested within P1, and topic Y1a nested within Y1.

   ```
   <map chunk="by-document">
      <topicref href="parent1.dita" chunk="to-content">
         <topicref href="ditabase.dita#Y1"
            chunk="select-branch"/>
      </topicref>
   </map>
   ```

7. The following map will produce a single output chunk, `P1.xxxx`. The topic P1 will be the root topic, and topics X, Y, and Z (together with their descendents) will be nested within topic P1.

```
<map chunk="by-topic">
   <topicref href="parent1.dita" chunk="to-content">
      <topicref href="ditabase.dita#Y1"
         chunk="select-document"/>
   </topicref>
</map>
```

8. The following map will produce a single output chunk named `parentchunk.xxxx` containing topic P1 at the root. Topic N1 will be nested within P1, and N1a will be nested within N1.

```
<map chunk="by-document">
   <topicref href="parent1.dita" chunk="to-content"
copy-to="parentchunk.dita">
     <topicref href="nested1.dita" chunk="select-branch"/>

   </topicref>
</map>
```

9. The following map will produce two output chunks. The first chunk named `parentchunk.xxxx` will contain the topics P1, C1, C3, and GC3. The "to-content" token on the reference to `child2.dita` causes that branch to begin a new chunk named `child2chunk.xxxx`, which will contain topics C2 and GC2.

```
<map chunk="by-document">
   <topicref href="parent1.dita"
      chunk="to-content" copy-to="parentchunk.dita">
      <topicref href="child1.dita" chunk="select-branch"/>

      <topicref href="child2.dita"
         chunk="to-content select-branch"
         copy-to="child2chunk.dita">
         <topicref href="grandchild2.dita"/>
      </topicref>
      <topicref href="child3.dita">
         <topicref href="grandchild3.dita"
            chunk="select-branch"/>
      </topicref>
   </topicref>
 </map>
```

10. The following map produces a single chunk named `nestedchunk.xxxx`, which contains topic N1 with no topics nested within.

```
<map>
   <topicref href="nested1.dita#N1"
             copy-to="nestedchunk.dita"
             chunk="to-content select-topic"/>
</map>
```

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 parent 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 reference a more detailed definition of the subject, for example, another DITA topic or an external resource..

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 the hierarchies of attribute values that are defined in subject scheme maps for purposes of filtering, flagging, 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.) If the controlled values are part of a named key scope, the scope name is ignored for the purpose of validating the controlled values.
- 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-therapist" since it is a type of therapist.

ul>

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">
      <subjectdef keys="la"/>
      <subjectdef keys="nyc"/>
      <subjectdef keys="san-francisco"/>
    </subjectdef>
    <subjectdef keys="state">
      <subjectdef keys="ca"/>
      <subjectdef keys="ny"/>
    </subjectdef>
  </hasInstance>
  <hasPart>
    <subjectdef keys="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.

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.

A 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.

```
<subjectScheme>
    <subjectdef keys="os">
        <topicmeta>
            <navtitle>Operating systems</navtitle>
        </topicmeta>
        <subjectdef keys="linux">
            <topicmeta>
                <navtitle>Linux</navtitle>
            </topicmeta>
            <subjectdef keys="redhat">
                <topicmeta>
                    <navtitle>RedHat Linux</navtitle>
                </topicmeta>
            </subjectdef>
            <subjectdef keys="suse">
                <topicmeta>
                    <navtitle>SuSE Linux</navtitle>
                </topicmeta>
            </subjectdef>
        </subjectdef>
        <subjectdef keys="windows">
        <topicmeta>
            <navtitle>Windows</navtitle>
        </topicmeta>
    </subjectdef>
    <subjectdef keys="zos">
        <topicmeta>
            <navtitle>z/OS</navtitle>
        </topicmeta>
    </subjectdef>
    </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, "redhat" also is excluded. Applications can recover by generating an error message.	Excluded.
	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.

att="platform" val="linux"	att="platform" val="redhat"	How platform="redhat" is evaluated	How platform="linux" is evaluated
	action="include"	Included.	Included, because all "redhat" content is included, and general 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">
        <topicmeta>
            <navtitle>Operating systems</navtitle>
        </topicmeta>
        <subjectdef keys="linux">
            <topicmeta>
                <navtitle>Linux</navtitle>
            </topicmeta>
            <subjectdef keys="redhat">
                <topicmeta>
                    <navtitle>RedHat Linux</navtitle>
                </topicmeta>
            </subjectdef>
            <subjectdef keys="suse">
                <topicmeta>
                    <navtitle>SuSE Linux</navtitle>
                </topicmeta>
            </subjectdef>
        </subjectdef>
        <subjectdef keys="windows">
        <topicmeta>
            <navtitle>Windows</navtitle>
        </topicmeta>
    </subjectdef>
    <subjectdef keys="zos">
        <topicmeta>
            <navtitle>z/OS</navtitle>
        </topicmeta>
    </subjectdef>
    </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"/>
    <subjectdef keyref="windows">
      <subjectdef keys="winxp"/>
      <subjectdef keys="winvis"/>
    </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">
    <subjectdef keys="linux">
      <subjectdef keys="redhat"/>
      <subjectdef keys="suse"/>
    </subjectdef>
    <subjectdef keys="macos">
    <subjectdef keys="windows">
      <subjectdef keys="winxp"/>
      <subjectdef keys="winvis"/>
    </subjectdef>
    <subjectdef keys="zos"/>
  </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="software">
    <subjectdef keyref="os"/>
    <subjectdef keys="applications">
      <subjectdef keys="apache-web-server""/>
      <subjectdef keys="my-sql"/>
    </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>
```

Cascading of metadata attributes in a DITA map

Certain map-level attributes cascade throughout a map, which facilitates attribute and metadata management. When attributes *cascade*, they apply to the elements that are children of the element where the attributes were specified. Cascading applies to a containment hierarchy, as opposed to a element-type hierarchy.

The following attributes cascade when set on the map element or when set within a map:

- audience, platform, product, otherprops, rev
- props and any attribute specialized from props
- linking, toc, search
- format, scope, type
- xml:lang, dir, translate
- processing-role
- cascade

Cascading is additive for attributes that accept multiple values, except when the cascade attribute is set to avoid adding values to attributes. For attributes that take a single value, the closest value defined on a containing element takes effect. In a relationship table, row-level metadata is considered more specific than column-level metadata, as shown in the following containment hierarchy:

- map (most general)
  - topicref container (more specific)
    - topicref (most specific)
  - reltable (more specific)
    - relcolspec (more specific)
      - relrow (more specific)
        - topicref (most specific)

Merging of cascading attributes

The cascade attribute can be used to modify the additive nature of attribute cascading (though it does not turn off cascading altogether). The attribute has two predefined values: "merge" and "nomerge".

cascade="merge" The metadata attributes cascade; the values of the metadata attributes are additive. This is the processing default for the cascade attribute and was the only defined behavior for DITA 1.2 and earlier.

cascade="nomerge" The metadata attributes cascade; however, they are not additive for topicref elements that specify a different value for a specific metadata attribute. If the cascading value for an attribute is already merged based on multiple ancestor elements, that merged value continues to cascade until a new value is encountered (that is, setting `cascade="nomerge"` does not undo merging that took place on ancestors).

Implementers *MAY* define their own custom, implementation-specific tokens. To avoid name conflicts between implementations or with future additions to the standard, implementation-specific tokens *SHOULD* consist of a prefix that gives the name or an abbreviation for the implementation followed by a colon followed by the token or method name.

For example, a processor might define the token "appToken:audience" in order to specify cascading and merging behaviors for **only** the audience attribute. The following rules apply:

- The predefined values for the cascade attribute *MUST* precede any implementation-specific tokens, for example, `cascade="merge appToken:audience"`.
- Tokens can apply to a set of attributes, specified as part of the cascade value. In that case, the syntax for specifying those values consists of the implementation-specific token, followed by a parenthetical group that uses the same syntax as groups within the audience, platform, product, and otherprops attributes. For example, a token that applies to only platform and product could be specified as `cascade="appname:token(platform product)"`.

Examples of the cascade attribute in use

Consider the following code examples:

```
<map audience="a b" cascade="merge">
     <topicref href="topic.dita" audience="c"/>
</map>
```

Figure 1: Map A

```
<map audience="a b" cascade="nomerge">
     <topicref href="topic.dita" audience="c"/>
</map>
```

Figure 2: Map B

For map A, the values for the attribute are merged, and the effective value of the audience attribute for `topic.dita` is "a b c". For map B, the values for the attribute are not additive, and the effective value of the audience attribute for `topic.dita` is "c".

In the following example, merging is active at the map level but turned off below:

```
<map platform="a" product="x" cascade="merge">
  <topicref href="one.dita" platform="b" product="y">
   <topicref href="two.dita" cascade="nomerge" product="z"/>

  </topicref>
</map>
```

Figure 3: Map C

In map C, the reference to `one.dita` has effective merged values of "a b" for platform and "x y" for product.

The reference to `two.dita` turns off merging, so the explicit product value of "z" is used (it does not merge with ancestor values). The platform attribute is not present, so the already-merged value of "a b" continues to cascade and is the effective value of platform on this reference.

Order for processing cascading attributes in a map

When determining the value of an attribute, processors *MUST* evaluate each attribute on each individual element in a specific order; this order is specified in the following list. Applications *MUST* continue through the list until a value is established or until the end of the list is reached (at which point no value is established for the attribute). In essence, the list provides instructions on how processors can construct a map where all attribute values are set and all cascading is complete.

For example, in the case of `<topicref toc="yes">`, applications *MUST* stop at item *List item.* on page 28 in the list; a value is specified for toc in the document instance, so toc values from containing elements will not cascade to that specific topicref element. The `toc="yes"` setting on that topicref element will cascade to contained elements, provided those elements reach item *List item.* on page 28 below when evaluating the toc attribute.

For attributes within a map, the following processing order *MUST* occur:

1. The conref and keyref attributes are evaluated.
2. The explicit values specified in the document instance are evaluated. For example, a topicref element with the toc attribute set to "no" will use that value.
3. The default or fixed attribute values are evaluated. For example, the toc attribute on the reltable element has a default value of "no".
4. The default values that are supplied by a controlled values file are evaluated.
5. The attributes cascade.
6. The processing-supplied default values are applied.
7. After the attributes are resolved within the map, they cascade to referenced maps.

> **Note:** The processing-supplied default values do not cascade to other maps. For example, most processors will supply a default value of `toc="yes"` when no toc attribute is specified. However, a processor-supplied default of `toc="yes"` *MUST* not override a value of `toc="no"` that is set on a referenced map. If the `toc="yes"` value is explicitly specified, is given as a default through a DTD, XSD, RNG, or controlled values file, or cascades from a containing element in the map, it *MUST* override a `toc="no"` setting on the referenced map. See *Map-to-map cascading behaviorsWhen a DITA map (or branch of a DITA map) is referenced by another DITA map, by default, certain rules apply. These rules pertain to the cascading behaviors of attributes, metadata elements, and roles assigned to content (for example, the role of "Chapter" assigned by a chapter element). Attributes and elements that cascade within a map generally follow the same rules when cascading from one map to another map, but there are some exceptions and additional rules that apply.* for more details.

8. Repeat steps *List item.* on page 28 to *List item.* on page 28 for each referenced map.
9. The attributes cascade within each referenced map.
10. The processing-supplied default values are applied within each referenced map.
11. Repeat the process for maps referenced within the referenced maps.

Reconciling topic and map metadata elements

The topicmeta element in maps contains numerous elements that can be used to declare metadata. These metadata elements have an effect on the parent topicref element, any child topicref elements, and – if a direct child of the map element – on the map as a whole.

For each element that can be contained in the topicmeta element, the following table addresses the following questions:

How does it apply to the topic?
This column describes how the metadata specified within the topicmeta element interacts with the metadata specified in the topic. In most cases, the properties are additive. For example, when the audience element is set to "user" at the map level, the value "user" is added during processing to any audience metadata that is specified within the topic.

Does it cascade to other topics in the map?
This column indicates whether the specified metadata value cascades to nested topicref elements. For example, when an audience element is set to "user" at the map level, all child topicref elements implicitly have an audience element set to "user" also. Elements that can apply only to the specific topicref element, such as linktext, do not cascade.

What is the purpose when specified on the map element?
The map element allows metadata to be specified for the entire map. This column describes what effect, if any, an element has when specified at this level.

Table 1: Topicmeta elements and their properties

Element	How does it apply to the topic?	Does it cascade to child topicref elements?	What is the purpose when set on the map element?
audience	Add to the topic	Yes	Specify an audience for the entire map
author	Add to the topic	Yes	Specify an author for the entire map
category	Add to the topic	Yes	Specify a category for the entire map
copyright	Add to the topic	Yes	Specify a copyright for the entire map
critdates	Add to the topic	Yes	Specify critical dates for the entire map
data	Add to the topic	No, unless specialized for a purpose that cascades	No stated purpose, until the element is specialized
data-about	Add the property to the specified target	No, unless specialized for a purpose that cascades	No stated purpose, until the element is specified
foreign	Add to the topic	No, unless specialized for a purpose that cascades	No stated purpose, until the element is specified
keywords	Add to the topic	No	No stated purpose
linktext	Not added to the topic; applies only to links created based on this occurrence in the map	No	No stated purpose

Element	How does it apply to the topic?	Does it cascade to child topicref elements?	What is the purpose when set on the map element?
metadata	Add to the topic	Yes	Specify metadata for the entire map
navtitle	Not added to the topic; applies only to navigation that is created based on this occurrence in the map. The navigation title will be used whenever the locktitle attribute on the containing topicref element is set to "yes".	No	No stated purpose
othermeta	Add to the topic	No	Define metadata for the entire map
permissions	Add to the topic	Yes	Specify permissions for the entire map
prodinfo	Add to the topic	Yes	Specify product info for the entire map
publisher	Add to the topic	Yes	Specify a publisher for the map
resourceid	Add to the topic	No	Specify a resource ID for the map
searchtitle	Replace the one in the topic. If multiple searchtitle elements are specified for a single target, processors can choose to issue a warning.	No	No stated purpose
shortdesc	Only added to the topic when the topicref element specifies a copy-to attribute. Otherwise, it applies only to links created based on this occurrence in the map. **Note:** Processors *MAY* or *MAY NOT* implement this behavior.	No	Provide a description of the map
source	Add to the topic	No	Specify a source for the map
unknown	Add to the topic	No, unless specialized for a purpose that cascades	No stated purpose, until the element is specified
ux-window	Not added to the topic	No	Definitions are global, so setting at map level is equivalent to setting anywhere else.

Example of metadata elements cascading in a DITA map

The following code sample illustrates how an information architect can apply certain metadata to all the DITA topics in a map:

```
<map title="DITA maps" xml:lang="en-us">
 <topicmeta>
  <author>Kristen James Eberlein</author>
  <copyright>
   <copyryear year="2009"/>
   <copyrholder>OASIS</copyrholder>
  </copyright>
 </topicmeta>
 <topicref href="dita_maps.dita">
```

```
  <topicref href="definition_ditamaps.dita"/>
  <topicref href="purpose_ditamaps.dita"/>
  <!-- ... -->
        </topicref>
</map>
```

The author and copyright information cascades to each of the DITA topics referenced in the DITA map. When the DITA map is processed to XHTML, for example, each XHTML file contains the metadata information.

Cascading of attributes from map to map

Certain elements cascade from map to map, although some of the attributes that cascade within a map do not cascade from map to map.

The following attributes cascade from map to map:

- audience, platform, product, otherprops, rev
- props and any attribute specialized from props
- linking, toc, print, search
- type
- translate
- processing-role
- cascade

Note that the above list excludes the following attributes:

format The format attribute must be set to "ditamap" in order to reference a map or a branch of a map, so it cannot cascade through to the referenced map.

xml:lang and dir Cascading behavior for xml:lang is defined in *The xml:lang attribute The xml:lang attribute specifies the language and (optional) locale of the element content. The xml:lang attribute applies to all attributes and content of the element where it is specified, unless it is overridden with xml:lang on another element within that content.*. The dir attribute work the same way.

scope The value of the scope attribute describes the map itself, rather than the content. When the scope attribute is set to "external", it indicates that the referenced map itself is external and unavailable, so the value cannot cascade into that referenced map.

The class attribute is used to determine the processing roles that cascade from map to map. See *Cascading of roles from map to map* on page 36 for more information.

As with values that cascade within a map, the cascading is additive if the attribute permits multiple values (such as audience). When the attribute only permits one value, the cascading value overrides the top-level element.

Example of attributes cascading between maps

For example, assume the following references in `test.ditamap`:

```
<map>
  <topicref href="a.ditamap" format="ditamap" toc="no"/>
  <mapref   href="b.ditamap" audience="developer"/>
  <mapref   href="c.ditamap#branch2" platform="myPlatform"/>
</map>
```

- The map `a.ditamap` is treated as if `toc="no"` is specified on the root map element. This means that the topics that are referenced by `a.ditamap` do not appear in the navigation generated by `test.ditamap` (except for branches within the map that explicitly set `toc="yes"`).
- The map `b.ditamap` is treated as if `audience="developer"` is set on the root map element. If the audience attribute is already set on the root map element within `b.ditamap`, the value "developer" is added to any existing values.
- The element with `id="branch2"` within the map `c.ditamap` is treated as if `platform="myPlatform"` is specified on that element. If the platform attribute is

already specified on the element with `id="branch"`, the value "myPlatform" is added to existing values.

Cascading of metadata elements from map to map

Elements that are contained within topicmeta or metadata elements follow the same rules for cascading from map to map as the rules that apply within a single DITA map.

For a complete list of which elements cascade within a map, see the column "Does it cascade to child topicref elements?" in the topic *Reconciling topic and map metadata elements* on page 29.

Note: It is possible that a specialization might define metadata that should replace rather than add to metadata in the referenced map, but DITA (by default) does not currently support this behavior.

For example, consider the following code examples:

```
<map>
    <topicref href="a.ditamap" format="ditamap">
        <topicmeta>
            <shortdesc>This map contains information about
Acme defects.</shortdesc>
        </topicmeta>
    </topicref>
    <topicref href="b.ditamap" format="ditamap">
        <topicmeta>
            <audience type="programmer"/>
        </topicmeta>
    </topicref>
    <mapref href="c.ditamap" format="ditamap"/>
    <mapref href="d.ditamap" format="ditamap"/>
    </map>
```

Figure 4: `test-2.ditamap`

```
<map>
    <topicmeta>
        <audience type="writer"/>
    </topicmeta>
    <topicref href="b-1.dita"/>
    <topicref href="b-2.dita"/>
</map>
```

Figure 5: `b.ditamap`

When `test-2.ditamap` is processed, the following behavior occurs:

- Because the shortdesc element does not cascade, it does not apply to the DITA topics that are referenced in `a.ditamap`.
- Because the audience element cascades, the audience element in the reference to `b.ditamap` combines with the audience element that is specified at the top level of `b.ditamap`. The result is that the `b-1.dita` topic and `b-2.dita` topic are processed as though hey each contained the following child topicmeta element:

  ```
  <topicmeta>
      <audience type="programmer"/>
  ```

```
    <audience type="writer"/>
</topicmeta>
```

Cascading of roles from map to map

When specialized topicref elements (such as chapter or mapref) reference a map, they typically imply a semantic role for the referenced content.

The semantic role reflects the class hierarchy of the referencing topicref element; it is equivalent to having the class attribute from the referencing topicref cascade to the top-level topicref elements in the referenced map. Although this cascade behavior is not universal, there are general guidelines for when class values should be replaced.

When a topicref element or a specialization of a topicref element references a DITA resource, it defines a role for that resource. In some cases this role is straightforward, such as when a topicref element references a DITA topic (giving it the already known role of "topic"), or when a mapref element references a DITA map (giving it the role of "DITA map").

Unless otherwise instructed, a specialized topicref element that references a map supplies a role for the referenced content. This means that, in effect, the class attribute of the referencing element cascades to top-level topicref elements in the referenced map. In situations where this should not happen - such as all elements from the mapgroup domain - the non-default behavior should be clearly specified.

For example, when a chapter element from the bookmap specialization references a map, it supplies a role of "chapter" for each top-level topicref element in the referenced map. When the chapter element references a branch in another map, it supplies a role of "chapter" for that branch. The class attribute for chapter ("- map/topicref bookmap/chapter ") cascades to the top-level topicref element in the nested map, although it does not cascade any further.

Alternatively, the mapref element in the mapgroup domain is a convenience element; the top-level topicref elements in the map referenced by a mapref element *MUST NOT* be processed as if they are mapref elements. The class attribute from the mapref element ("+ map/topicref mapgroup-d/mapref ") does not cascade to the referenced map.

In some cases, preserving the role of the referencing element might result in out-of-context content. For example, a chapter element that references a bookmap might pull in part elements that contain nested chapter elements. Treating the part element as a chapter will result in a chapter that nests other chapters, which is not valid in bookmap and might not be understandable by processors. The result is implementation specific; processors *MAY* choose to treat this as an error, issue a warning, or simply assign new roles to the problematic elements.

Example of cascading roles between maps

Consider the scenario of a chapter element that references a DITA map. This scenario could take several forms:

Referenced map contains a single top-level topicref element	The entire branch functions as if it were included in the bookmap; the top-level topicref element is processed as if it were the chapter element.
Referenced map contains multiple top-level topicref elements	Each top-level topicref element is processed as if it were a chapter element (the referencing element).
Referenced map contains a single appendix element	The appendix element is processed as it were a chapter element.

Referenced map contains a single part element, with nested chapter elements.	The part element is processed as it were a chapter element. Nested chapter elements might not be understandable by processors; applications *MAY* recover as described above.
chapter element references a single topicref element rather than a map	The referenced topicref element is processed as if it were a chapter element.

Key scopes

Key scopes enable map authors to specify different sets of key definitions for different map branches.

A key scope is defined by a map or topicref element that specifies the keyscope attribute. The keyscope attribute specifies the names of the scope, separated by spaces. The legal characters for a key scope name are the same as those for keys.

A key scope includes the following components:

- The scope-defining element
- The elements that are contained by the scope-defining element, minus the elements that are contained by child key scopes
- The elements that are referenced by the scope-defining element or its descendants, minus the elements that are contained by child key scopes

If the keyscope attribute is specified on both a reference to a DITA map and the root element of the referenced map, only one scope is created; the submap does not create another level of scope hierarchy. The single key scope that results from this scenario has multiple names; its names are the union of the values of the keyscope attribute on the map reference and the root element of the submap. This means that processors can resolve references to both the key scopes specified on the map reference and the key scopes specified on the root element of the submap.

The root element of a root map always defines a key scope, regardless of whether a keyscope attribute is present. All key definitions and key references exist within a key scope, even if it is an unnamed, implicit key scope that is defined by the root element in the root map.

Each key scope has its own key space that is used to resolve the key references that occur within the scope. The key space that is associated with a key scope includes all of the key definitions within the key scope. This means that different key scopes can have different effective key definitions:

- A given key can be defined in one scope, but not another.
- A given key also can be defined differently in different key scopes.

Key references in each key scope are resolved using the effective key definition that is specified within its own key scope.

Example: Key scopes specified on both the map reference and the root element of the submap

Consider the following scenario:

```
<map>
  <mapref keyscope="A" href="installation.ditamap"/>
  <!-- ... -->
</map>
```

Figure 6: Root map

```
<map keyscope="B">
  <!-- ... -->
</map>
```

Figure 7: `installation.ditamap`

Only one key scope is created; it has key scope names of "A" and "B".

Using keys for addressing

For topic references, image references, and other link relationships, resources can be indirectly addressed by using the keyref attribute. For content reference relationships, resources can be indirectly addressed by using the conkeyref attribute.

Syntax

For references to topics, maps, and non-DITA resources, the value of the keyref attribute is simply a key name (for example, `keyref="topic-key"`).

For references to non-topic elements within topics, the value of the keyref attribute is a key name, a slash ("/"), and the ID of the target element (for example, `keyref="topic-key/some-element-id"`.)

Example

For example, consider this topic in the document `file.dita`:

```
<topic id="topicid">
 <title>Example referenced topic</title>
 <body>
  <section id="section-01">Some content.</section>
 </body>
</topic>
```

and this key definition:

```
<map>
  <topicref keys="myexample"
    href="file.dita"
  />
</map>
```

A cross reference of the form `keyref="myexample/section-01"` resolves to the section element in the topic. The key reference is equivalent to the URI reference `xref="file.dita#topicid/section-01"`.

Cross-deliverable addressing and linking

A map can use scoped keys to reference keys that are defined in a different root map. This cross-deliverable addressing can support the production of deliverables that contain working links to other deliverables.

When maps are referenced and the value of the scope attribute is set to "peer", the implications are that the two maps are managed in tandem, and that the author of the referencing map might have access to the referenced map. Adding a key scope to the reference indicates that the peer map should be treated as a separate deliverable for the purposes of linking.

The keys that are defined by the peer map belong to any key scopes that are declared on the topicref element that references that map. Such keys can be referenced from content in the referencing map by using scope-qualified key names. However, processors handle references to keys that are defined in peer maps differently from how they handle references to keys that are defined in submaps.

DITA processors are not required to resolve key references to peer maps. However, if all resources are available in the same processing or management context, processors have the potential to resolve key references to peer maps. There might be performance, scale, and user interface challenges in implementing such systems, but the ability to resolve any given reference is ensured when the source files are physically accessible.

Note the inverse implication; if the peer map is not available, then it is impossible to resolve the key reference. Processors that resolve key references to peer maps should provide appropriate messages when a reference to a peer map cannot be resolved. Depending on how DITA resources are authored, managed, and processed, references to peer maps might not be resolvable at certain points in the content life cycle.

The peer map might specify keyscope on its root element. In that case, the keyscope on the peer map is ignored for the purpose of resolving scoped key references from the referencing map. This avoids the need for processors to have access to the peer map in order to determine whether a given key definition comes from the peer map.

Example: A root map that declares a peer map

Consider the DITA maps `map-a.ditamap` and `map-b.ditamap`. Map A designates Map B as a peer map by using the following markup:

```
<map>
  <title>Map A</title>
  <topicref
    scope="peer"
    format="ditamap"
    keyscope="map-b"
    href="../map-b/map-b.ditamap"
    processing-role="resource-only"
  />
  <!-- ... -->
</map>
```

In this example, `map-b.ditamap` is not a submap of Map A; it is a peer map.

Example: Key resolution in a peer map that contains a keyscope attribute on the root element

Consider the map reference in map Map A:

```
<mapref
  keyscope="scope-b"
  scope="peer"
  href="map-b.ditamap"
/>
```

where `map-b.ditamap` contains the following markup:

```
<map keyscope="product-x">
 <!-- ... -->
</map>
```

From the context of Map A, key references of the form "scope-b.somekey" are resolved to keys that are defined in the global scope of map B, but key references of the form "product-x.somekey" are not. The presence of a keyscope attribute on the map element in Map B has no effect. A key reference to the scope "scope-b.somekey" is equivalent to the unscoped reference "somekey" when processed in the context of Map B as the root map. In both cases, the presence of keyscope on the root element of Map B has no effect; in the first case it is explicitly ignored, and in the second case the key reference is within the scope "product-x" and so does not need to be scope qualified.

Processing xrefs and conrefs within a conref

When referenced content contains a content reference or cross reference, the effective target of the reference depends on the form of address that is used in the referenced content. It also might depend on the map context, especially when key scopes are present.

Direct URI reference (but not a same-topic fragment identifier)
When the address is a direct URI reference of any form other than a same-topic fragment identifier, processors *MUST* resolve it relative to the source document that contains the original URI reference.

Same-topic fragment identifier
When the address is a same-topic fragment identifier, processors *MUST* resolve it relative to the location of the content reference (referencing context).

Key reference
When the address is a key reference, processors *MUST* resolve it relative to the location of the content reference (referencing context).

When resolving key references or same-topic fragment identifiers, the phrase *location of the content reference* means the final resolved context. For example, in a case where content references are chained (topic A pulls from topic B, which in turn pulls a reference from topic C), the reference is resolved relative to the topic that is rendered. When topic B is rendered, the reference is resolved relative to the content reference in topic B; when topic A is rendered, the reference is resolved relative to topic A. If content is pushed from topic A to topic B to topic C, then the same-topic fragment identifier is resolved in the context of topic C.

The implication is that a content reference or cross reference can resolve to different targets in different use contexts. This is because a URI reference that contains a same-topic fragment identifier is resolved in the context of the topic that contains the content reference, and a key reference is resolved in the context of the key scope that is in effect for each use of the topic that contains the content reference.

Note: In the case of same-topic fragment identifiers, it is the responsibility of the author of the content reference to ensure that any element IDs that are specified in same-topic fragment identifiers in the referenced content will also be available in the referencing topic at resolution time.

Example: Resolving conrefs to elements that contain cross references

Consider the following paragraphs in `paras-01.dita` that are intended to be used by reference from other topics:

```
<topic id="paras-01"><title>Reusable paragraphs</title>
    <body>
        <p id="p1">See <xref href="#paras-01/p5"/>.</p>
        <p id="p2">See <xref
href="topic-02.dita#topic02/fig-01"/>.</p>
        <p id="p3">See <xref href="#./p5"/>.</p>
       <p id="p4">See <xref keyref="task-remove-cover"/>.</p>

        <p id="p5">Paragraph 5 in paras-01.</p>
    </body>
</topic>
```

The paragraphs are used by content reference from other topics, including the `using-topic-01.dita` topic:

```
<topic id="using-topic-01"><title>Using topic one</title>
    <body>
        <p id="A" conref="paras-01.dita#paras-01/p1"/>
        <p id="B" conref="paras-01.dita#paras-01/p2"/>
        <p id="C" conref="paras-01.dita#paras-01/p3"/>
        <p id="D" conref="paras-01.dita#paras-01/p4"/>
        <p id="p5">Paragraph 5 in using-topic-01</p>
    </body>
</topic>
```

Following resolution of the content references and processing of the xref elements in the referenced paragraphs, the rendered cross references in `using-topic-01.dita` are shown in the following table.

Paragraph	Value of id attribute on conrefed paragraph	xref within conrefed paragraph	Resolution
A	p1	`<xref href="#paras-01/p5"/>`	The cross reference in paragraph p1 is a direct URI reference that does not contain a same-topic fragment identifier. It can be resolved only to paragraph p5 in `paras-01.dita`, which contains the content "Paragraph 5 in paras-01".
B	p2	`<xref href="topic-02.dita#topic02/fig-01"/>`	The cross reference in paragraph p2 is a direct URI reference. It can be resolved only to the element with `id="fig-01"` in `topic-02.dita`.
C	p3	`<xref href="#./p5"/>`	The cross reference in paragraph p3 is a direct URI reference that contains a same-topic fragment identifier. Because the URI reference contains a same-topic fragment identifier, the reference is resolved in the context of the referencing topic (`using-topic-01.dita`). If `using-topic-01.dita` did not contain an element with `id="p5"`, then the conref to paragraph p3 would result in a link resolution failure.
D	p4	`<xref keyref="task-remove-cover"/>`	The cross reference in paragraph p4 is a key reference. It is resolved to whatever resource is bound to the key name "task-remove-cover" in the applicable map context.

Example: Resolving conrefs to elements that contain key-based cross references

Consider the following map, which uses the topics from the previous example:

```
<map>
  <topicgroup keyscope="product-1">
    <topicref keys="task-remove-cover"
href="prod-1-task-remove-cover.dita"/>
    <topicref href="using-topic-01.dita"/>
  </topicgroup>
  <topicgroup keyscope="product-2">
    <topicref keys="task-remove-cover"
href="prod-2-task-remove-cover.dita"/>
    <topicref href="using-topic-01.dita"/>
  </topicgroup>
</map>
```

The map establishes two key scopes: "product-1" and "product-2". Within the map branches, the key name "task-remove-cover" is bound to a different topic. The topic `using-topic-01.dita`, which includes a conref to a paragraph that includes a cross reference to the key name "task-remove-cover", is also referenced in each branch. When each branch is rendered, the target of the cross reference is different.

In the first branch with the key scope set to "product-1", the cross reference from paragraph p4 is resolved to `prod-1-task-remove-cover.dita`. In the second branch with the key scope set to "product-2", the cross reference from paragraph p4 is resolved to `prod-2-task-remove-cover.dita`.

domains attribute rules and syntax

The domains attribute enables processors to determine whether two elements or two documents use compatible domains. The attribute is declared on the root element for each topic or map type. Each structural, domain, and constraint module defines its ancestry as a parenthesized sequence of space-separated module names; the effective value of the domains attribute is composed of these parenthesized sequences.

Document type shells collect the values that are provided by each module to construct the effective value of the domains attribute. Processors can examine the collected values when content from one document is used in another, in order to determine whether the content is compatible.

For example, when an author pastes content from one topic into another topic within an XML editor, the application can use the domains attribute to determine if the two topics use compatible domains. If not, copied content from the first topic might need to be generalized before it can be placed in the other topic.

The domains attribute serves the same function when an element uses the conref attribute to reference a more specialized version of the element. For example, a note element in a concept topic conrefs a hazardstatement element in a reference document. If the hazard statement domain is not available in the concept topic, the hazardstatement element is generalized to a note element when the content reference is resolved.

Syntax and rules

Each domain and constraint module *MUST* provide a value for use by the domains attribute. Each structural vocabulary module *SHOULD* provide a value for use by the domains attribute, and it *MUST* do so when it has a dependency on elements from any module that is not part of its specialization ancestry.

Values provided for the domains attribute values are specified from root module (map or topic) to the provided module.

structural modules

The value of the domains attribute includes each module in the specialization ancestry:

```
 '(', topic-or-map, (' ', module)+, ')'
```

For example, consider the glossentry specialization, in which the topic type is specialized to the concept type, and the concept type is specialized to glossentry. The structural module contribution to the value of the domains attribute for the glossentry structural module is `(topic concept glossentry)`.

structural modules with dependencies

Structural modules can directly reference or specialize elements from modules that are outside of their specialization ancestry. They also can define specialized elements that reference specialized attributes. In these cases the structural module has a dependency on the non-ancestor module, and the structural module contribution to the value of the domains attribute *MUST* include the names of each dependent, non-ancestor module.

Dependencies are included in the value of the domains attribute following the name of the structural module with the dependency on the non-ancestor module. Domain or attribute modules are appended to the name of the structural module with the dependency on the non-ancestor module, or to previous dependencies, separated by "+". Dependencies on structural specialization modules are appended to the name of the structural module with the dependency on the non-ancestor module, or to previous dependencies, separated by "++". The syntax is the same as for other structural modules, except that added modules can include these dependencies:

```
 '(', topic-or-map, (' ',
module-plus-optional-dependency-list)+, ')'
```

When the structural module is included in a document-type shell, all dependency modules also are included along with their own domains values.

For example, the cppAPIRef structural module is specialized from reference, which is specialized from topic. The cppAPIRef module has a dependency on the cpp-d element domain and on the compilerTypeAtt-d attribute domain. The dependencies are listed after the name of `cppApiref`:

```
(topic reference cppApiRef+cpp-d+compilerTypeAtt-d)
```

Similarly, a codeChecklist structural module is specialized from reference, which is specialized from topic. The codeChecklist module has a dependency on the pr-d domain and on the task structural specialization. Again, the dependencies are listed after the name of `codeChecklist`. The pr-d domain and the task module each contribute their own values, so taken together these modules contribute the following values:

```
(topic reference codeChecklist+pr-d++task) (topic pr-d) (topic
 task)
```

element domains

The value includes the structural type ancestry and, if applicable, the domain module ancestry from which the domain is specialized:

```
 '(', topic-or-map, (' ', domain-module)+, ')'
```

For example, the highlighting domain (specialized from topic) supplies the following value: `(topic hi-d)`. A CPP domain that is specialized from the programming domain, which in turn is specialized from topic, supplies the following value: `(topic pr-d cpp-d)`.

structural constraint modules

The value includes the structural type ancestry followed by the name of the constraint domain:

```
 '(', inheritance-hierarchy qualifierTagname-c, ')'
```

where:

- *inheritance-hierarchy* is the specialization hierarchy, for example, `topic task`.
- *qualifier* is a string that is specific to the constraints module and characterizes it, for example, "strict" or "requiredTitle" or "myCompany-".
- *Tagname* is the element type name with an initial capital, for example, "Taskbody" or "Topic".
- The literal "-c" indicates that the name is the name of a constraint.

For example, the strictTaskbody constraint applies to the task module, which is specialized from topic, resulting in the following value: `(topic task strictTaskbody-c)`.

Optionally, a domains contribution can indicate a strong constraint by preceding the domains contribution with the letter "s". For example, `s(topic task strictTaskbody-c)` indicates a strong constraint.

domain constraint modules

The value includes the specialization ancestry followed by the name of the constraint domain:

```
 '(', inheritance-hierachy qualifierdomainDomain-c ')'
```

where:

- *inheritance-hierarchy* is the specialization hierarchy, for example, `topic hi-d`.
- *qualifier* is a string that is specific to the constraints module and characterizes it, for example, "noSyntaxDiagram" or "myCompany-".
- *domain* is the name of the domain to which the constraints apply, for example, "Highlighting" or "Programming".
- The literal "-c" indicates that the name is the name of a constraint.

For example, a domain constraint module that restricts the highlighting domain includes a value like the following: `(topic hi-d basic-HighlightingDomain-c)`

attribute domains

The value uses an "a" before the initial parenthesis to indicate an attribute domain. Within the parenthesis, the value includes the attribute specialization hierarchy, starting with props or base:

```
 'a(', props-or-base, (' ', attname)+, ')'
```

For example, the mySelectAttribute specialized from props results in the following value: `a(props mySelectAttribute)`

Example: Task with multiple domains

In this example, a document-type shell integrates the task structural module and the following domain modules:

Domain	Domain short name
User interface	ui-d
Software	sw-d
Programming	pr-d

The value of the domains attribute includes one value from each module; the effective value is the following:

```
domains="(topic task) (topic ui-d) (topic sw-d) (topic pr-d)"
```

If the document-type shell also used a specialization of the programming domain that describes C++ programming (with a short name of "cpp-d"), the new C++ programming domain would add an additional value to the domains attribute:

```
domains="(topic task) (topic ui-d) (topic sw-d) (topic pr-d)
 (topic pr-d cpp-d)"
```

Note that the value for the domains attribute is not authored; Instead, the value is defaulted based on the modules that are included in the document type shell.

domains attribute rules and syntax

The domains attribute enables processors to determine whether two elements or two documents use compatible domains. The attribute is declared on the root element for each topic or map type. Each structural, domain, and constraint module defines its ancestry as a parenthesized sequence of space-separated module names; the effective value of the domains attribute is composed of these parenthesized sequences.

Document type shells collect the values that are provided by each module to construct the effective value of the domains attribute. Processors can examine the collected values when content from one document is used in another, in order to determine whether the content is compatible.

For example, when an author pastes content from one topic into another topic within an XML editor, the application can use the domains attribute to determine if the two topics use compatible domains. If not, copied content from the first topic might need to be generalized before it can be placed in the other topic.

The domains attribute serves the same function when an element uses the conref attribute to reference a more specialized version of the element. For example, a note element in a concept topic conrefs a hazardstatement element in a reference document. If the hazard statement domain is not available in the concept topic, the hazardstatement element is generalized to a note element when the content reference is resolved.

Syntax and rules

Each domain and constraint module *MUST* provide a value for use by the domains attribute. Each structural vocabulary module *SHOULD* provide a value for use by the domains attribute, and it *MUST* do so when it has a dependency on elements from any module that is not part of its specialization ancestry.

Values provided for the domains attribute values are specified from root module (map or topic) to the provided module.

structural modules

The value of the domains attribute includes each module in the specialization ancestry:

```
 '(', topic-or-map, (' ', module)+, ')'
```

For example, consider the glossentry specialization, in which the topic type is specialized to the concept type, and the concept type is specialized to glossentry. The structural module contribution to the value of the domains attribute for the glossentry structural module is `(topic concept glossentry)`.

structural modules with dependencies

Structural modules can directly reference or specialize elements from modules that are outside of their specialization ancestry. They also can define specialized elements that reference specialized attributes. In these cases the structural module has a dependency on the non-ancestor module, and the structural module contribution to the value of the domains attribute *MUST* include the names of each dependent, non-ancestor module.

Dependencies are included in the value of the domains attribute following the name of the structural module with the dependency on the non-ancestor module. Domain or attribute modules are appended to the name of the structural module with the dependency on the non-ancestor module, or to previous dependencies, separated by "+". Dependencies on structural specialization modules are appended to the name of the structural module with the dependency on the non-ancestor module, or to previous dependencies, separated by "++". The syntax is the same as for other structural modules, except that added modules can include these dependencies:

```
 '(', topic-or-map, (' ',
module-plus-optional-dependency-list)+, ')'
```

When the structural module is included in a document-type shell, all dependency modules also are included along with their own domains values.

For example, the cppAPIRef structural module is specialized from reference, which is specialized from topic. The cppAPIRef module has a dependency on the cpp-d element domain and on the compilerTypeAtt-d attribute domain. The dependencies are listed after the name of `cppApiref`:

```
(topic reference cppApiRef+cpp-d+compilerTypeAtt-d)
```

Similarly, a codeChecklist structural module is specialized from reference, which is specialized from topic. The codeChecklist module has a dependency on the pr-d domain and on the task structural specialization. Again, the dependencies are listed after the name of `codeChecklist`. The pr-d domain and the task module each contribute their own values, so taken together these modules contribute the following values:

```
(topic reference codeChecklist+pr-d++task) (topic pr-d) (topic
 task)
```

element domains

The value includes the structural type ancestry and, if applicable, the domain module ancestry from which the domain is specialized:

```
 '(', topic-or-map, (' ', domain-module)+, ')'
```

For example, the highlighting domain (specialized from topic) supplies the following value: `(topic hi-d)`. A CPP domain that is specialized from the programming domain, which in turn is specialized from topic, supplies the following value: `(topic pr-d cpp-d)`.

structural constraint modules

The value includes the structural type ancestry followed by the name of the constraint domain:

```
 '(', inheritance-hierarchy qualifierTagname-c, ')'
```

where:

- *inheritance-hierarchy* is the specialization hierarchy, for example, `topic task`.
- *qualifier* is a string that is specific to the constraints module and characterizes it, for example, "strict" or "requiredTitle" or "myCompany-".
- *Tagname* is the element type name with an initial capital, for example, "Taskbody" or "Topic".
- The literal "-c" indicates that the name is the name of a constraint.

For example, the strictTaskbody constraint applies to the task module, which is specialized from topic, resulting in the following value: `(topic task strictTaskbody-c)`.

Optionally, a domains contribution can indicate a strong constraint by preceding the domains contribution with the letter "s". For example, `s(topic task strictTaskbody-c)` indicates a strong constraint.

domain constraint modules

The value includes the specialization ancestry followed by the name of the constraint domain:

```
 '(', inheritance-hierachy qualifierdomainDomain-c ')'
```

where:

- *inheritance-hierarchy* is the specialization hierarchy, for example, `topic hi-d`.
- *qualifier* is a string that is specific to the constraints module and characterizes it, for example, "noSyntaxDiagram" or "myCompany-".
- *domain* is the name of the domain to which the constraints apply, for example, "Highlighting" or "Programming".
- The literal "-c" indicates that the name is the name of a constraint.

For example, a domain constraint module that restricts the highlighting domain includes a value like the following: `(topic hi-d basic-HighlightingDomain-c)`

attribute domains

The value uses an "a" before the initial parenthesis to indicate an attribute domain. Within the parenthesis, the value includes the attribute specialization hierarchy, starting with props or base:

```
 'a(', props-or-base, (' ', attname)+, ')'
```

For example, the mySelectAttribute specialized from props results in the following value: `a(props mySelectAttribute)`

Example: Task with multiple domains

In this example, a document-type shell integrates the task structural module and the following domain modules:

Domain	Domain short name
User interface	ui-d
Software	sw-d
Programming	pr-d

The value of the domains attribute includes one value from each module; the effective value is the following:

```
domains="(topic task) (topic ui-d) (topic sw-d) (topic pr-d)"
```

If the document-type shell also used a specialization of the programming domain that describes C++ programming (with a short name of "cpp-d"), the new C++ programming domain would add an additional value to the domains attribute:

```
domains="(topic task) (topic ui-d) (topic sw-d) (topic pr-d)
 (topic pr-d cpp-d)"
```

Note that the value for the domains attribute is not authored; Instead, the value is defaulted based on the modules that are included in the document type shell.

Example: Setting conditional processing values and groups

Conditional processing attributes can be used to classify content using either individual values or using groups.

Example: Simple product values

In the following example, the first configuration option applies only to the "extendedprod" product, while the second option applies to both "extendedprod" and to "baseprod". The entire p element containing the list applies to an audience of "administrator".

```
<p audience="administrator">Set the configuration options:
  <ul>
    <li product="extendedprod">Set foo to bar</li>
    <li product="basicprod extendedprod">Set your blink
rate</li>
    <li>Do some other stuff</li>
    <li>Do a special thing for Linux</li>
  </ul>
</p>
```

Example: Grouped values on an attribute

The following example indicates that a step applies to one application server and two databases. Specifically, this step only applies when it is taken on the server "mySERVER"; likewise, it only applies when used with the databases "ABC" or "dbOtherName".

```
<steps>
  <step><cmd>Common step</cmd></step>
  <step product="appserver(mySERVER) database(ABC
dbOtherName)">
    <cmd>Do something special for databases ABC or OtherName
 when installing on mySERVER</cmd>
  </step>
  <!-- additional steps -->
</steps>
```

Example: Single ditavalref on a branch

A single ditavalref element can be used to supply filtering conditions for a branch.

Consider the following DITA map and the DITAVAL file that is referenced from the ditavalref element:

```
<map>
  <topicref href="intro.dita"/>
  <topicref href="install.dita">
    <ditavalref href="novice.ditaval"/>
    <topicref href="do-stuff.dita"/>
    <topicref href="advanced-stuff.dita" audience="admin"/>
    <!-- more topics -->
  </topicref>
  <!-- Several chapters worth of other material -->
</map>
```

Figure 8: `input.ditamap`:

```
<val>
  <prop att="audience" val="novice" action="include"/>
  <prop att="audience" val="admin" action="exclude"/>
</val>
```

Figure 9: Contents of `novice.ditaval`

When this content is published, the following processing occurs:

- The first topic (`intro.dita`) does not use any of the conditions that are specified in `novice.ditaval`. It is published normally, potentially using other DITAVAL conditions that are specified externally.
- The second topic (`install.dita`) is filtered using any external conditions as well as the conditions that are specified in `novice.ditaval`.
- The third topic (`do-stuff.dita`) is filtered using any external conditions as well as the conditions that are specified in `novice.ditaval`.
- The fourth topic (`advanced-stuff.dita`) is removed from the map entirely, because it is filtered out with the conditions that are specified for the branch.

In this example, no resources are renamed based on the ditavalref processing.

Note: In cases where the original resource names map directly to names or anchors in a deliverable, the absence of renaming ensures that external links to those topics are stable regardless of whether a DITAVAL document is used.

Example: Multiple ditavalref elements on a branch

Multiple ditavalref elements can be used on a single map branch to create multiple distinct copies of the branch.

Consider the following DITA map that contains a branch with three peer ditavalref elements. Because topics in the branch are filtered in three different ways, processors are effectively required to handle three copies of the entire branch. Sub-elements within the ditavalref elements are used to control how new resource names are constructed for two copies of the branch; one copy (based on the conditions in `win.ditaval`) is left with the original file names.

```
<map>
  <topicref href="intro.dita"/>
  <!-- Begining of installing branch -->
  <topicref href="install.dita">
    <ditavalref href="win.ditaval"/>
    <ditavalref href="mac.ditaval">
      <ditavalmeta>
        <dvrResourceSuffix>-apple</dvrResourceSuffix>
      </ditavalmeta>
    </ditavalref>
    <ditavalref href="linux.ditaval">
      <ditavalmeta>
        <dvrResourceSuffix>-linux</dvrResourceSuffix>
      </ditavalmeta>
    </ditavalref>
    <topicref href="do-stuff.dita">
    <!-- more topics and nested branches -->
      <topicref href="mac-specific-stuff.dita"
platform="mac"/>
    </topicref>
    <!-- End of installing branch -->
    <topicref href="cleanup.dita"/>
  </topicref>
</map>
```

Figure 10: `input.ditamap`

```
<val>
  <prop att="platform" val="win" action="include"/>
  <prop att="platform" action="exclude"/>
</val>
```

Figure 11: Contents of `win.ditaval`

```
<val>
  <prop att="platform" val="mac" action="include"/>
  <prop att="platform" action="exclude"/>
</val>
```

Figure 12: Contents of `mac.ditaval`

```
<val>
  <prop att="platform" val="linux" action="include"/>
  <prop att="platform" action="exclude"/>
</val>
```

Figure 13: Contents of `linux.ditaval`

When a processor evaluates this markup, it results in three copies of the installing branch. The following processing takes place:

- The first topic (`intro.dita`) is published normally, potentially using any other DITAVAL conditions that are specified externally.
- The installing branch appears three times, once for each DITAVAL document. The branches are created as follows:
  - The first branch uses the first DITAVAL document (`win.ditaval`). Resources use their original names as specified in the map. The `mac-specific-stuff.dita` topic is removed. The resulting branch, with indenting to show the hierarchy, matches the original without the mac topic:

    ```
    install.dita
       do-stuff.dita
         ...more topics and nested branches...
       cleanup.dita
    ```

  - The second branch uses the second DITAVAL document (`mac.ditaval`). Resources are renamed based on the dvrResourceSuffix element. The `mac-specific-stuff.dita` topic is included. The resulting branch, with indenting to show the hierarchy, is as follows:

    ```
    install-apple.dita
       do-stuff-apple.dita
          mac-specific-stuff-apple.dita
          ...more topics and nested branches...
       cleanup-apple.dita
    ```

  - The third branch uses the last DITAVAL document (`linux.ditaval`). Resources are renamed based on the dvrResourceSuffix element. The `mac-specific-stuff.dita` topic is removed. The resulting branch, with indenting to show the hierarchy, is as follows:

    ```
    install-linux.dita
       do-stuff-linux.dita
         ...more topics and nested branches...
       cleanup-linux.dita
    ```

The example used three DITAVAL documents to avoid triple maintenance of the installing branch in a map; the following map is functionally equivalent, but it requires parallel maintenance of each branch.

```
<map>
  <topicref href="intro.dita"/>
  <!-- Windows installing branch -->
  <topicref href="install.dita">
    <ditavalref href="win.ditaval"/>
    <topicref href="do-stuff.dita">
      <!-- more topics and nested branches -->
    </topicref>
    <topicref href="cleanup.dita"/>
  </topicref>
  <!-- Mac installing branch -->
  <topicref href="install.dita">
```

```
    <ditavalref href="mac.ditaval">

<ditavalmeta><dvrResourceSuffix>-apple</dvrResourceSuffix></ditavalmeta>

    </ditavalref>
    <topicref href="do-stuff.dita">
      <topicref href="mac-specific-stuff.dita"
platform="mac"/>
      <!-- more topics and nested branches -->
    </topicref>
    <topicref href="cleanup.dita"/>
  </topicref>
  <!-- Linux installing branch -->
  <topicref href="install.dita">
    <ditavalref href="linux.ditaval">

<ditavalmeta><dvrResourceSuffix>-linux</dvrResourceSuffix></ditavalmeta>

    </ditavalref>
    <topicref href="do-stuff.dita">
      <!-- more topics and nested branches -->
    </topicref>
    <topicref href="cleanup.dita"/>
  </topicref>
  <!-- Several chapters worth of other material -->
</map>
```

Figure 14: `input.ditamap`

Example: Single ditavalref as a child of map

Using a ditavalref element as a direct child of the map element is equivalent to setting global filtering conditions for the map.

The following map is equivalent to processing all the contents of the map with the conditions in the `novice.ditaval` document. If additional conditions are provided externally (for example, as a parameter to the publishing process), those conditions take precedence.

```
<map>
  <title>Sample map</title>
  <ditavalref href="novice.ditaval"/>
  <!-- lots of content -->
</map>
```

Example: Single ditavalref in a reference to a map

Using a ditavalref element in a reference to a map is equivalent to setting filtering conditions for the referenced map.

In the following example, `other.ditamap` is referenced by a root map. The ditavalref element indicates that all of the content in `other.ditamap` should be filtered using the conditions specified in the `some.ditaval` document.

```
<topicref href="parent.dita">
  <topicref href="other.ditamap" format="ditamap">
    <ditavalref href="some.ditaval"/>
  </topicref>
</topicref>
```

Figure 15: Map fragment

```
<map>
  <topicref href="nestedTopic1.dita">
    <topicref href="nestedTopic2.dita"/>
  </topicref>
  <topicref href="nestedTopic3.dita"/>
</map>
```

Figure 16: Contents of `other.ditamap`

This markup is functionally equivalent to applying the conditions in `some.ditaval` to the topics that are referenced in the nested map. For the purposes of filtering, it could be rewritten in the following way. The extra topicgroup container is used here to ensure filtering is not applied to `parent.dita`, as it would not be in the original example:

```
<topicref href="parent.dita">
  <topicgroup>
    <ditavalref href="some.ditaval"/>
    <topicref href="nestedTopic1.dita">
      <topicref href="nestedTopic2.dita"/>
    </topicref>
    <topicref href="nestedTopic3.dita"/>
  </topicgroup>
</topicref>
```

For the purposes of filtering, this map also could be rewritten as follows.

```
<topicref href="parent.dita">
  <topicref href="nestedTopic1.dita">
    <ditavalref href="some.ditaval"/>
    <topicref href="nestedTopic2.dita"/>
  </topicref>
  <topicref href="nestedTopic3.dita">
    <ditavalref href="some.ditaval"/>
  </topicref>
</topicref>
```

Filtering based on the ditavalref element applies to the containing element and its children, so in each case, the files `nestedTopic1.dita`, `nestedTopic2.dita`, and nestedTopic3.dita are filtered against the conditions specified in some.ditaval. In each version, parent.dita is not a parent for the ditavalref, so it is not filtered.

Example: Multiple ditavalref elements as children of map in a root map

Using multiple instances of the ditavalref element as direct children of the map element in a root map is equivalent to setting multiple sets of global filtering conditions for the root map.

Note: Unlike most other examples of branch filtering, this example cannot be rewritten using a single valid map with alternate markup that avoids having multiple ditavalref elements as children of the same grouping element.

Processing the following root map is equivalent to processing all the contents of the map with the conditions in the `mac.ditaval` file and again with the `linux.ditaval` file. If additional conditions are provided externally (for example, as a parameter to the publishing process), those global conditions take precedence.

```
<map>
  <title>Setting up my product
on <keyword platform="mac">Mac</keyword><keyword
platform="linux">Linux</keyword></title>
  <topicmeta>
    <othermeta platform="mac"   name="ProductID"
content="1234M"/>
    <othermeta platform="linux" name="ProductID"
content="1234L"/>
  </topicmeta>
  <ditavalref href="mac.ditaval"/>
  <ditavalref href="linux.ditaval"/>
  <!-- lots of content, including relationship tables -->
</map>
```

Figure 17: `input.ditamap`

```
<val>
  <prop att="platform" val="mac"   action="include"/>
  <prop att="platform" val="linux" action="exclude"/>
</val>
```

Figure 18: Contents of `mac.ditaval`

```
<val>
  <prop att="platform" val="mac"   action="exclude"/>
  <prop att="platform" val="linux" action="include"/>
</val>
```

Figure 19: Contents of `linux.ditaval`

Because the title and metadata each contain filterable content, processing using the conditions that are referenced by the ditavalref element results in two variants of the title and common metadata. While this cannot be expressed using valid DITA markup, it is conceptually similar to something like the following.

```
<!-- The following wrapperElement is not a real DITA element.

     It is used here purely as an example to illustrate one
```

```
possible
     way of picturing the conditions. -->
<wrapperElement>
  <map>
    <title>Setting up my product on <keyword
platform="mac">Mac</keyword></title>
    <topicmeta>
      <othermeta platform="mac"   name="ProductID"
content="1234M"/>
    </topicmeta>
    <ditavalref href="mac.ditaval"/>
    <!-- lots of content, including relationship tables -->
  </map>
  <map>
    <title>Setting up my product on <keyword
platform="linux">Linux</keyword></title>
    <topicmeta>
      <othermeta platform="linux" name="ProductID"
content="1234L"/>
    </topicmeta>
    <ditavalref href="linux.ditaval"/>
    <!-- lots of content, including relationship tables -->
  </map>
</wrapperElement>
```

How this map is rendered is implementation dependent. If this root map is rendered as a PDF, possible renditions might include the following:

- Two PDFs, with one using the conditions from `mac.ditaval` and another using the conditions from `linux.ditaval`
- One PDF, with a title page that includes each filtered variant of the title and product ID, followed by Mac-specific and Linux-specific renderings of the content as chapters in the PDF
- One PDF, with the first set of filter conditions used to set book level titles and metadata, followed by content filtered with those conditions, followed by content filtered with conditions from the remaining ditavalref element.

Example: Multiple ditavalref elements in a reference to a map

Using multiple instances of the ditavalref element in a reference to a map is equivalent to referencing that map multiple times, with each reference nesting one of the ditavalref elements.

In the following example, `other.ditamap` is referenced by a root map. The ditavalref elements provide conflicting sets of filter conditions.

```
<topicref href="parent.dita">
  <topicref href="other.ditamap" format="ditamap">
    <ditavalref href="audienceA.ditaval"/>
    <ditavalref href="audienceB.ditaval"/>
    <ditavalref href="audienceC.ditaval"/>
  </topicref>
</topicref>
```

Figure 20: Map fragment

This markup is functionally equivalent to referencing `other.ditamap` three times, with each reference including a single ditavalref elements. The fragment could be rewritten as:

```
<topicref href="parent.dita">
  <topicref href="other.ditamap" format="ditamap">
    <ditavalref href="audienceA.ditaval"/>
  </topicref>
  <topicref href="other.ditamap" format="ditamap">
    <ditavalref href="audienceB.ditaval"/>
  </topicref>
  <topicref href="other.ditamap" format="ditamap">
    <ditavalref href="audienceC.ditaval"/>
  </topicref>
</topicref>
```

Figure 21: Map fragment

Example: ditavalref within a branch that already uses ditavalref

When a branch is filtered because a ditavalref element is present, another ditavalref deeper within that branch can supply additional conditions for a subset of the branch.

In the following map fragment, a set of operating system conditions applies to installation instructions. Within that common branch, a subset of content applies to different audiences.

```
<topicref href="install.dita">
  <ditavalref href="linux.ditaval"/>
  <ditavalref href="mac.ditaval">
    <ditavalmeta>
      <dvrResourceSuffix>-mac</dvrResourceSuffix>
    </ditavalmeta>
  </ditavalref>
  <ditavalref href="win.ditaval">
    <ditavalmeta>
      <dvrResourceSuffix>-win</dvrResourceSuffix>
    </ditavalmeta>
  </ditavalref>
  <topicref href="perform-install.dita">
    <!-- other topics-->
  </topicref>
  <!-- Begin configuration sub-branch -->
  <topicref href="configure.dita">
    <ditavalref href="novice.ditaval">
      <ditavalmeta>
        <dvrResourceSuffix>-novice</dvrResourceSuffix>
      </ditavalmeta>
    </ditavalref>
    <ditavalref href="advanced.ditaval">
      <ditavalmeta>
        <dvrResourceSuffix>-admin</dvrResourceSuffix>
      </ditavalmeta>
    </ditavalref>
    <!-- Other config topics -->
  </topicref>
  <!-- End configuration sub-branch -->
</topicref>
```

In this case, the effective map contains three copies of the complete branch. The branches are filtered by operating system. Because topics in the branch are filtered in different ways, processors are effectively required to handle three copies of the entire branch. The map author uses the dvrResourceSuffix elements to control naming for each copy. The Linux branch does not specify a dvrResourceSuffix element, because it is the default copy of the branch; this allows documents such as `install.dita` to retain their original names.

Within each operating system instance, the configuration sub-branch is repeated; it is filtered once for novice users and then again for advanced users. As a result, there are actually six instances of the configuration sub-branch. Additional dvrResourceSuffix elements are used to control naming for each instance.

1. The first instance is filtered using the conditions in `linux.ditaval` and `novice.ditaval`. For this instance, the resource `configure.dita` is treated as the resource `configure-novice.dita`. There is no renaming based on `linux.ditaval`, and the ditavalref the references `novice.ditaval` adds the `suffix -novice`.
2. The second instance is filtered using the conditions in `linux.ditaval` and `advanced.ditaval`. For this instance, the resource `configure.dita` is treated as the resource `configure-admin.dita`. There is no renaming based on `linux.ditaval`, and the ditavalref that references `advanced.ditaval` adds the suffix `-admin`.
3. The third instance is filtered using the conditions in `mac.ditaval` and `novice.ditaval`. For this instance, the resource `configure.dita` is treated as the resource `configure-novice-mac.dita`. The ditavalref that references `novice.ditaval` adds the suffix `-novice`, resulting in `configure-novice.dita`, and then the ditavalref that references `mac.ditaval` adds the additional suffix `-mac`.
4. The fourth instance is filtered using the conditions in `mac.ditaval` and `advanced.ditaval`. For this instance, the resource `configure.dita` is treated as the resource `configure-admin-mac.dita`. The ditavalref that references `admin.ditaval` adds the suffix `-admin`, resulting in `configure-admin.dita`, and then the ditavalref that references `mac.ditaval` adds the additional suffix `-mac`.
5. The fifth instance is filtered using the conditions in `win.ditaval` and `novice.ditaval`. For this instance, the resource `configure.dita` is treated as the resource `configure-novice-win.dita`. The ditavalref that references `novice.ditaval` adds the suffix `-novice`, resulting in `configure-novice.dita`, and then the ditavalref that references `win.ditaval` adds the additional suffix `-win`.
6. The sixth instance is filtered using the conditions in `win.ditaval` and `advanced.ditaval`. For this instance, the resource `configure.dita` is treated as the resource `configure-admin-win.dita`. The ditavalref that references `admin.ditaval` adds the suffix `-admin`, resulting in `configure-admin.dita`, and then the ditavalref that references `win.ditaval` adds the additional suffix `-win`.

Example: ditavalref error conditions

It is an error condition when multiple, non-equivalent copies of the same file are created with the same resource name.

The following map fragment contains several error conditions that result in name clashes:

```
<topicref href="a.dita" keys="a">
  <ditavalref href="one.ditaval"/>
  <ditavalref href="two.ditaval"/>
  <topicref href="b.dita" keys="b"/>
</topicref>
<topicref href="a.dita"/>
<topicref href="c.dita" keys="c">
  <ditavalref href="one.ditaval">
    <ditavalmeta>
      <dvrResourceSuffix>-token</dvrResourceSuffix>
    </ditavalmeta>
  </ditavalref>
  <ditavalref href="two.ditaval">
    <ditavalmeta>
      <dvrResourceSuffix>-token</dvrResourceSuffix>
    </ditavalmeta>
  </ditavalref>
</topicref>
```

In this sample, the effective map that results from evaluating the filter conditions has several clashes. In some cases the same document must be processed with conflicting conditions, using the same URI. In addition, because no key scope is added or modified, keys in the branch are duplicated in such a way that only one version is available for use. When the branches are evaluated to create distinct copies, the filtered branches result in the following equivalent map:

```
<topicref href="a.dita" keys="a"> <!-- a.dita to be filtered
 by one.ditaval -->
  <topicref href="b.dita" keys="b"/>  <!-- b.dita to be
filtered by one.ditaval -->
</topicref>
<topicref href="a.dita" keys="a"> <!-- a.dita to be filtered
 by two.ditaval; key "a" ignored -->
  <topicref href="b.dita" keys="b"/>  <!-- b.dita to be
filtered by two.ditaval; key "b" ignored -->
</topicref>
<topicref href="a.dita"/>
<topicref href="c-token.dita" keys="c">
  <!-- c-token.ditaval to be filtered by one.ditaval -->
</topicref>
<topicref href="c-token.dita" keys="c">
  <!-- c-token.ditaval to be filtered by two.ditaval, key "c"
 ignored -->
</topicref>
```

The equivalent map highlights several problems with the original source:

- The key names "a" and "b" are present in a branch that will be duplicated. No key scope is introduced for either version of the branch, meaning that the keys will be duplicated.

Because there can only be one effective key definition for "a" or "b", it only is possible to reference one version of the topic using keys.

- The key name "c" is present on another branch that will be duplicated, resulting in the same problem.
- The file `c.dita` is filtered with two sets of conditions, each of which explicitly maps the filtered resource to `c-token.dita`. This is an error condition that should be reported by processors.
- In situations where resource names map directly to output file names, such as an HTML5 rendering that creates files based on the original resource name, the following name conflicts also occur. In this case a processor would need to report an error, use an alternate naming scheme, or both:

  1. `a.dita` generates `a.html` using three alternate set of conditions. One version uses `one.ditaval`, one version uses `two.ditaval`, and the third version uses no filtering.
  2. `b.dita` generates `b.html` using two alternate set of conditions. One version uses `one.ditaval`, and the other version uses `two.ditaval`.

class attribute rules and syntax

The specialization hierarchy of each DITA element is declared as the value of the class attribute. The class attribute provides a mapping from the current name of the element to its more general equivalents, but it also can provide a mapping from the current name to more specialized equivalents. All specialization-aware processing can be defined in terms of class attribute values.

The class attribute tells a processor what general classes of elements the current element belongs to. DITA scopes elements by module type (for example topic type, domain type, or map type) instead of document type, which lets document type developers combine multiple module types in a single document without complicating transformation logic.

The sequence of values in the class attribute is important because it tells processors which value is the most general and which is most specific. This sequence is what enables both specialization aware processing and generalization.

Syntax

Values for the class attribute have the following syntax requirements:

- An initial "-" or "+" character followed by one or more spaces. Use "-" for element types that are defined in structural vocabulary modules, and use "+" for element types that are defined in domain modules.
- A sequence of one or more tokens of the form `"`*`modulename/typename`*`",` with each token separated by one or more spaces, where *modulename* is the short name of the vocabulary module and *typename* is the element type name. Tokens are ordered left to right from most general to most specialized.

  These tokens provide a mapping for every structural type or domain in the ancestry of the specialized element. The specialization hierarchy for a given element type must reflect any intermediate modules between the base type and the specialization type, even those in which no element renaming occurs.
- At least one trailing space character (" "). The trailing space ensures that string matches on the tokens can always include a leading and trailing space in order to reliably match full tokens.

Rules

When the class attribute is declared in an XML grammar, it *MUST* be declared with a default value. In order to support generalization round-tripping (generalizing specialized content into a generic form and then returning it to the specialized form) the default value *MUST NOT* be fixed. This allows a generalization process to overwrite the default values that are defined by a general document type with specialized values taken from the document being generalized.

A vocabulary module *MUST NOT* change the class attribute for elements that it does not specialize, but simply reuses by reference from more generic levels. For example, if task, bctask, and guitask use the p element without specializing it, they *MUST NOT* declare mappings for it.

Authors *SHOULD NOT* modify the class attribute.

Example: DTD declaration for class attribute for the step element

The following code sample lists the DTD declaration for the class attribute for the step element:

```
<!ATTLIST step         class  CDATA "- topic/li task/step ">
```

This indicates that the step element is specialized from the li element in a generic topic. It also indicates explicitly that the step element is available in a task topic; this enables round-trip migration between upper level and lower level types without the loss of information.

Example: Element with class attribute made explicit

The following code sample shows the value of the class attribute for the wintitle element:

```
<wintitle class="+ topic/keyword ui-d/wintitle ">A specialized
 keyword</wintitle>
```

The class attribute and its value is generally not surfaced in authored DITA topics, although it might be made explicit as part of a processing operation.

Example: class attribute with intermediate value

The following code sample shows the value of a class attribute for an element in the guitask module, which is specialized from task. The element is specialized from keyword in the base topic vocabulary, rather than from an element in the task module:

```
<windowname class="- topic/keyword task/keyword
guitask/windowname ">...</windowname>
```

The intermediate values are necessary so that generalizing and specializing transformations can map the values simply and accurately. For example, if `task/keyword` was missing as a value, and a user decided to generalize this guitask up to a task topic, then the transformation would have to guess whether to map to keyword (appropriate if task is more general than guitask, which it is) or leave it as windowname (appropriate if task were more specialized, which it isn't). By always providing mappings for more general values, processors can then apply the simple rule that missing mappings must by default be to more specialized values than the one we are generalizing to, which means the last value in the list is appropriate. For example, when generalizing guitask to task, if a p element has no target value for task, we can safely assume that p does not specialize from task and should not be generalized.

Specializing to include non-DITA content

You can extend DITA to incorporate standard vocabularies for non-textual content, such as MathML and SVG, as markup within DITA documents. This is done by specializing the foreign or unknown elements.

There are three methods of incorporating foreign content into DITA.

- A domain specialization of the foreign or unknown element. This is the usual implementation.
- A structural specialization using the foreign or unknown element. This affords more control over the content.
- Directly embedding the non-DITA content within foreign or unknown elements. If the non-DITA content has interoperability or vocabulary naming issues such as those that are addressed by specialization in DITA, they must be addressed by means that are appropriate to the non-DITA content.

The foreign or unknown elements should not be used to include textual content or metadata in DITA documents, except where such content acts as an example or display, rather than as the primary content of a topic.

Example: Creating an element domain specialization for SVG

The following code sample, which is from the `svgDomain.ent` file, shows the domain declaration for the SVG domain.

```
<!--
============================================================
 -->
<!--                    SVG DOMAIN ENTITIES
       -->
<!--
============================================================
 -->

<!-- SVG elements must be prefixed, otherwise they conflict
with
     existing DITA elements (e.g., <desc> and <title>.
  -->
<!ENTITY % NS.prefixed "INCLUDE" >
<!ENTITY % SVG.prefix "svg" >

<!ENTITY % svg-d-foreign
   "svg-container
   "
>

<!ENTITY   svg-d-att
  "(topic svg-d)"
>
```

Note that the SVG-specific SVG.prefix parameter entity is declared. This establishes the default namespace prefix to be used for the SVG content embedded with this domain. The namespace can be overridden in a document-type shell by declaring the parameter entity before the reference to the `svgDomain.ent` file. Other foreign domains might need similar entities when required by the new vocabulary.

For more information, see the `svgDomain.mod` file that is shipped with the OASIS DITA distributions. For an example of including the SVG domain in a document type shell, see `task.dtd`.

Sharing elements across specializations

Specialization enables easy reuse of elements from ancestor specializations. However, it is also possible to reuse elements from non-ancestor specializations, as long as the dependency is properly declared in order to prevent invalid generalization or conref processing.

A structural specialization can incorporate elements from unrelated domains or other structural specializations by referencing them in the content model of a specialized element. The elements included in this manner must be specialized from ancestor content that is valid in the new context. If the reusing and reused specializations share common ancestry, the reused elements must be valid in the reusing context at every level they share in common.

Although a well-designed structural specialization hierarchy with controlled use of domains is still the primary means of sharing and reusing elements in DITA, the ability to also share elements declared elsewhere in the hierarchy allows for situations where relevant markup comes from multiple sources and would otherwise be developed redundantly.

Example: A specialization of concept reuses an element from the task module

A specialized concept topic could declare a specialized process section that contains the steps element that is defined in the task module. This is possible because of the following factors:

- The steps element is specialized from ol.
- The process element is specialized from section, and the content model of section includes ol.

The steps element in process always can be generalized back to ol in section.

Example: A specialization of reference reuses an element from the programming domain

A specialized reference topic could declare a specialized list (apilist) in which each apilistitem contains an apiname element that is borrowed from the programming domain.

Conref compatibility with constraints

To determine compatibility between two document instances, a conref processor checks the domains attribute to confirm whether the referencing document has a superset of the vocabulary modules in the referenced document. If one or both of the document instances are constrained, the conref processor checks to confirm the compatibility of the constraints.

Conref processors take into account whether constraints are specified as strong. For strong constraints, the following rules apply:

Conref pull
For each vocabulary module used by both document types, the module in the document type that contains the referencing element must be less (or equally) constrained than the same module in the document type that contains the referenced element. For example, if each document type uses the highlighting domain module, that module must be less (or equally) constrained in the document type that contains the referencing element.

Conref push
For each vocabulary module used by both document types, the module in the document type that contains the referencing element must be more (or equally) constrained than the same module in the document type that contains the referenced element. For example, if each document type uses the highlighting domain module, that module must be more (or equally) constrained in the document type that contains the referencing element.

Example: Conref pull and constraint compatibility

The following table contains scenarios where conref pull occurs between constrained and unconstrained document instances. It assumes that the processor is **not** configured to treat all constraints as strong constraints.

Values of domains attribute in document type that contains the referencing element	Values of domains attribute in document type that contains the referenced element	Resolution	Comments
`(topic)`	`(topic shortdescReq-c)`	Allowed	The content model of the referenced topic is more constrained than the referencing topic.
`s(topic shortdescReq-c)`	`(topic)`	Prevented	The constraint is specified as a strong constraint, and the content model of the referenced topic is less constrained than the referencing topic.
`(topic shortdescReq-c)`	`(topic)`	Allowed	Although the content model of referenced topic is less constrained than the referencing topic, this is a weak constraint and so permitted.

Values of domains attribute in document type that contains the referencing element	Values of domains attribute in document type that contains the referenced element	Resolution	Comments
`(topic task) (topic hi-d) (topic hi-d basicHighlightingDomain-c)`	`(topic simpleSection-c) (topic task) (topic task simpleStep-c)`	Allowed	The referenced topic has a subset of the vocabulary modules that are integrated into the document-type shell for the referencing topic. Both document types integrate constraints, but for modules used in both document types, the referencing topic is less constrained than the referenced topic.
`(topic hi-d) (topic simpleSection-c) s(topic simpleP-c)`	`(topic simpleSection-c) (topic task) (topic hi-d) (topic hi-d basicHighlightingDomain-c)`	Prevented	The referencing document has constraints that are not present in the referenced document, including a strong constraint applied to the p element.

Example: Conref push and constraint compatibility

The following table contains scenarios where conref push occurs between constrained and unconstrained document instances. It assumes that the processor has **not** been configured to treat all constraints as strong constraints.

Values of domains attribute in document type that contains the referencing element	Values of domains attribute in document type that contains the referenced element	Resolution	Comments
`(topic)`	`(topic shortdescReq-c)`	Allowed	Although the content model of the referenced topic is more constrained than the referencing topic, this is a weak constraint and so permitted.
`(topic)`	`s(topic shortdescReq-c)`	Prevented	The constraint is specified as a strong constraint, and the content model of the referenced topic is more constrained than the referencing topic.

Values of domains attribute in document type that contains the referencing element	Values of domains attribute in document type that contains the referenced element	Resolution	Comments
`(topic shortdescReq-c)`	`(topic)`	Allowed	The content model of the referencing topic is more constrained than the referenced topic.
`(topic task) (topic hi-d) (topic hi-d basicHighlightingDomain-c)`	`(topic simpleSection-c) (topic task) (topic task simpleStep-c)`	Allowed	The referenced topic has a subset of the vocabulary modules that are integrated into the document-type shell for the referencing topic. For modules used in both document types, the referenced topic is more constrained than the referencing topic, but this is a weak constraint and so permitted.
`(topic simpleSection-c) (topic task) (topic hi-d) (topic hi-d basicHighlightingDomain-c)`	`(topic hi-d) (topic simpleSection-c) s(topic simpleP-c)`	Prevented	For the common topic module, the referenced document has more constraints than the referencing document, including a strong constraint applied to the p element.

DTD: Coding requirements for element domain modules

The vocabulary modules that define element domains have an additional coding requirement. The entity declaration file must include a parameter entity for each element that the domain extends.

Parameter entity name

The name of the parameter entity is the abbreviation for the domain, followed by a hyphen ("-"), and the name of the element that is extended.

Parameter entity value

The value of the parameter entity is a list of the specialized elements that can occur in the same locations as the extended element. Each element must be separated by the vertical line (|) symbol.

Example

Because the highlighting domain extends the ph element, the entity declaration file for that domain must include a parameter entity corresponding to the ph element. The name of the entity uses the short name of the domain (`hi-d`) followed by the name of the base element. The value includes each specialization of ph in the domain.

```
<!ENTITY % hi-d-ph "b | u | i | line-through | overline | tt
 | sup | sub">
```

Example: Redefine the content model for the topic element

In this scenario, an information architect for Acme, Incorporated wants to redefine the content model for the topic document type. She wants to omit the abstract element and make the shortdesc element required; she also wants to omit the related-links element and disallow topic nesting.

1. She creates a .mod file using the following naming conventions: *`qualiferTagname`*`Constraint.mod`, where *qualifer* is a string the describes the constraint, and *Tagname* is the element type name with an initial capital. Her contraint module is named `acme-TopicConstraint.mod`.
2. She adds the following content to `acme-TopicConstraint.mod`:

```
<!--
==========================================================
 -->
<!--                    CONSTRAINED TOPIC ENTITIES
         -->
<!--
==========================================================
 -->

<!-- Declares the entity for the constraint module and
defines     -->
<!-- its contribution to the @domains attribute.
         -->

<!ENTITY topic-constraints
  "(topic basic-Topic-c)"
>

<!-- Declares the entities referenced in the constrained
content  -->
<!-- model.
        -->

<!ENTITY % title            "title">
<!ENTITY % titlealts        "titlealts">
<!ENTITY % shortdesc        "shortdesc">
<!ENTITY % prolog           "prolog">
<!ENTITY % body             "body">

<!-- Defines the constrained content model for <topic>.
         -->

<!ENTITY % topic.content
                       "((%title;),
                         (%titlealts;)?,
                         (%shortdesc;),
                         (%prolog;)?,
                         (%body;)?)"
>
```

3. She then integrates the constraint module into her document-type shell for topic by adding the following section above the "TOPIC ELEMENT INTEGRATION" comment:

```
<!--
==========================================================
 -->
<!--                    CONTENT CONSTRAINT INTEGRATION
         -->
<!--
==========================================================
 -->

<!ENTITY % topic-constraints-c-def
  PUBLIC "-//ACME//ELEMENTS DITA Topic Constraint//EN"
  "acme-TopicConstraint.mod">
%topic-constraints-c-def;
```

4. She then adds the constraint to the list of domains and constraints that need to be included in the value of the domains attribute for topic:

```
<!--
==========================================================
 -->
<!--                    DOMAINS ATTRIBUTE OVERRIDE
         -->
<!--
==========================================================
 -->

<!ENTITY included-domains
                          "&hi-d-att;
                           &ut-d-att;
                           &indexing-d-att;
                           &topic-constraints;
  "
>
```

5. After updating the `catalog.xml` file to include the new constraints file, her work is done.

Example: Constrain attributes for the section element

In this scenario, an information architect wants to redefine the attributes for the section element. He wants to make the id attribute required and omit the spectitle attribute.

1. He creates a .mod file named `idRequiredSectionContraint.mod`, where "idRequired" is a string that characterizes the constraint.
2. He adds the following content to `idRequiredSectionContraint.mod`:

```
<!--
==========================================================
 -->
<!--                    CONSTRAINED TOPIC ENTITIES
         -->
<!--
==========================================================
 -->

<!ENTITY section-constraints
  "(topic idRequired-section-c)"
>

<!-- Declares the entities referenced in the constrained
content  -->
<!-- model.
        -->
<!ENTITY % conref-atts
             'conref    CDATA #IMPLIED
              conrefend CDATA #IMPLIED
              conaction
(mark|pushafter|pushbefore|pushreplace|-dita-use-conref-target)
 #IMPLIED
              conkeyref CDATA #IMPLIED' >
<!ENTITY % filter-atts
             'props      CDATA #IMPLIED
              platform   CDATA #IMPLIED
              product    CDATA #IMPLIED
              audience   CDATA #IMPLIED
              otherprops CDATA #IMPLIED
              %props-attribute-extensions;' >
<!ENTITY % select-atts
             '%filter-atts;
              base      CDATA #IMPLIED
              %base-attribute-extensions;
              importance
(default|deprecated|high|low|normal|obsolete|optional|

recommended|required|urgent|-dita-use-conref-target)
#IMPLIED
              rev       CDATA #IMPLIED
              status
(changed|deleted|unchanged|-dita-use-conref-target)
#IMPLIED' >
```

```
<!ENTITY % localization-atts
             'translate (no|yes|-dita-use-conref-target)
#IMPLIED
              xml:lang CDATA #IMPLIED
              dir
(lro|ltr|rlo|rtl|-dita-use-conref-target) #IMPLIED' >

<!-- Declares the constrained content model. Original
definition   -->
<!-- included %univ-atts; and spectitle; now includes-->
<!-- individual pieces of univ-atts, to make ID required.
         -->

<!ENTITY % section.attributes
          "id          CDATA   #REQUIRED
           %conref-atts;
           %select-atts;
           %localization-atts;
           outputclass CDATA   #IMPLIED">
```

Note: The information architect had to declare all the parameter entities that are referenced in the redefined attributes for section. If he did not do so, none of the attributes that are declared in the conref-atts, select-atts, or localization-atts parameter entities would be available on the section element. Furthermore, since the select-atts parameter entity references the filter-atts parameter entity, the filter-atts must be declared and it must precede the declaration for the select-atts parameter entity. The props-attribute-extensions and base-attribute-extensions parameter entities do not need to be declared in the constraint module, because they are declared in the document-type shells before the inclusion of the constraint module.

3. He then integrates the constraint module into the applicable document-type shells and adds it to his `catalog.xml` file.

Example: Constrain a domain module

In this scenario, an information architect wants to use only a subset of the elements defined in the highlighting domain. She wants to use b and i, but not line-through, overline, sup, sup, tt, or u. She wants to integrate this constraint into the document-type shell for task.

1. She creates `reducedHighlightingDomainConstraint.mod`, where "reduced" is a string that characterizes the constraint.
2. She adds the following content to `reducedHighlightingDomainConstraint.mod`:

```
<!--
==========================================================
 -->
<!--      CONSTRAINED HIGHLIGHTING DOMAIN ENTITIES
         -->
<!--
==========================================================
 -->

<!ENTITY HighlightingDomain-constraints
  "(topic hi-d basic-HighlightingDomain-c)"
>

<!ENTITY % HighlightingDomain-c-ph     "b | i"
     >
```

3. She then integrates the constraint module into her company-specific, document-type shell for the task topic by adding the following section directly before the "DOMAIN ENTITY DECLARATIONS" comment:

```
<!--
==========================================================
 -->
<!--                    DOMAIN CONSTRAINT INTEGRATION
         -->
<!--
==========================================================
 -->

<!ENTITY % HighlightingDomain-c-dec
  PUBLIC "-//ACME//ENTITIES DITA Highlighting Domain
Constraint//EN"
  "acme-HighlightingDomainConstraint.mod"
>%basic-HighlightingDomain-c-dec;
```

4. In the "DOMAIN EXTENSIONS" section, she replaces the parameter entity for the highlighting domain with the parameter entity for the constrained highlighting domain:

```
<!ENTITY % ph           "ph |
                         %HighlightingDomain-c-ph; |
                         %sw-d-ph; |
                         %ui-d-ph;
                        ">
```

5. She then adds the constraint to the list of domains and constraints that need to be included in the value of the domains attribute for task:

```
<!--
==========================================================
 -->
<!--                    DOMAINS ATTRIBUTE OVERRIDE
         -->
<!--
==========================================================
 -->

<!ENTITY included-domains
                          "&task-att;
                           &hi-d-att;
                           &indexing-d-att;
                           &pr-d-att;
                           &sw-d-att;
                           &ui-d-att;
                           &taskbody-constraints;
                          &HighlightingDomain-constraints;

  "
>
```

6. After updating the `catalog.xml` file to include the new constraints file, her work is done.

Example: Replace a base element with the domain extensions

In this scenario, an information architect wants to remove the ph element but allow the extensions of ph that exist in the highlighting, programming, software, and user interface domains.

1. The information architect creates an entities file named `noPhConstraint.ent`, where "no" is a qualifier string that characterizes the constraint.
2. The information architect adds the following content to `noPhConstraint.ent`:

   ```
   <!--
   ==========================================================
    -->
   <!--      CONSTRAINED HIGHLIGHTING DOMAIN ENTITIES
            -->
   <!--
   ==========================================================
    -->

   <!ENTITY ph-constraints
     "(topic noPh-ph-c)"
   >
   ```

   Note: Because the highlighting and programming domains cannot be generalized without the ph element, this entity must be defined so that there is a separate parenthetical expression that can be included in the domains attribute for the topic.

3. The information architect then integrates the constraint module into a document-type shell for concept by adding the following section above the "TOPIC ELEMENT INTEGRATION" comment:

   ```
   <!--
   ==========================================================
    -->
   <!--                    CONTENT CONSTRAINT INTEGRATION
            -->
   <!--
   ==========================================================
    -->

   <!ENTITY % noPh-ph-c-def
     PUBLIC "-//ACME//ELEMENTS DITA Ph Constraint//EN"
     "acme-PhConstraint-constraints" "noPhConstraint.ent">
   %noPh-ph-c-def;
   ```

4. In the "DOMAIN EXTENSIONS" section, the information architect removes the reference to the ph element:

   ```
   <!-- Removed "ph | " so as to make <ph> not available, only
    the domain extensions. -->
   <!ENTITY % ph           "%pr-d-ph; |
                            %sw-d-ph; |
                            %ui-d-ph;
                           ">
   ```

5. She then adds the constraint to the list of domains and constraints that need to be included in the value of the domains attribute:

   ```
   <!--
   ==========================================================
    -->
   <!--                    DOMAINS ATTRIBUTE OVERRIDE
            -->
   <!--
   ==========================================================
    -->

   <!ENTITY included-domains
                             "&concept-att;
                              &hi-d-att;
                              &indexing-d-att;
                              &pr-d-att;
                              &sw-d-att;
                              &ui-d-att;
                              &ph-constraint;
     "
   >
   ```

6. After updating the `catalog.xml` file to include the new constraints file, the information architect's work is done.

Example: Apply multiple constraints to a single document-type shell

You can apply multiple constraints to a single document-type shell. However, there can be only one constraint for a given element or domain.

Here is a list of constraint modules and what they do:

File name	What it constrains	Details	Contribution to the domains attribute
example-TopicConstraint.mod	topic	• Removes abstract • Makes shortdesc required • Removes related-links • Disallows topic nesting	(topic basic-Topic-c)
example-SectionConstraint.mod	section	• Makes id required • Removes spectitle attribute	(topic idRequired-section-c)
example-HighlightingDomainConstraint.mod	Highlighting domain	Reduces the highlighting domain elements to b and i	(topic hi-d basic-HighlightingDomain-c)
example-PhConstraint.ent	ph	Removes the ph element	(topic noPh-ph-c)

All of these constraints can be integrated into a single document-type shell for topic, since they constrain distinct element types and domains. The constraint for the highlighting domain must be integrated before the "DOMAIN ENTITIES" section, but the order in which the other three constraints are listed does not matter.

Each constraint module provides a unique contribution to the domains attribute. When integrated into the document-type shell for topic, the effective value of the domains attribute will include the following values, as well as values for any other modules that are integrated into the document-type shell:

```
(topic basic-Topic-c) (topic idRequired-section-c) (topic
hi-d basic-HighlightingDomain-c) (topic noPh-ph-c)
```

Example: Correct the constraint for the machinery task

For DITA 1.3, the OASIS DITA TC failed to update the constraint for the machinery task. In this scenario, an information architect corrects that oversight; she makes the (new for DITA 1.3) tasktroubleshooting element available in the body of the machinery task.

Note: This example assumes that the information architect has already implemented her own document-type shell for the machinery task information type.

1. She makes a copy of the `machineryTaskbodyConstraint.mod` file and renames it `correctedMachineryTaskbodyConstraint.mod`.
2. She modifies the `correctedMachineryTaskbodyConstraint.mod` file to declare the entity for the tasktroubleshooting element and make it available at the correct place within the task body. (Her modifications are highlighted in bold below.)

```
<!ENTITY taskbody-constraints
  "(topic task+taskreq-d machineryTaskbody-c)"
>
<!ENTITY % prelreqs
 "prelreqs">
<!ENTITY % context
 "context">
<!ENTITY % section
 "section">
<!ENTITY % steps
 "steps">
<!ENTITY % steps-unordered
 "steps-unordered">
<!ENTITY % steps-informal
 "steps-informal">
<!ENTITY % result
 "result">
<!ENTITY % tasktroubleshooting
 "tasktroubleshooting">
<!ENTITY % example
 "example">
<!ENTITY % closereqs
 "closereqs">


<!ENTITY % taskbody.content
              "((%prelreqs; |
                 %context; |
                 %section;)*,
                (%steps; |
                 %steps-unordered; |
                 %steps-informal;)?,
                (%result;)?,
                (%tasktroubleshooting;)?,
                (%example;)*,
                (%closereqs;)?)"
```

3. She updates her company-specific document-type shell to integrate the updated constraint.
4. After updating the `catalog.xml` file to include the new constraints file, her work is done.

Index