DITA Technical Committee

Darwin Information Typing Architecture (DITA) Version 2.0

"Working Draft 12"

"29 August 2020"

Table of contents

1 Introduction

The Darwin Information Typing Architecture (DITA) specification defines a set of document types for authoring and aggregating topic-oriented information, as well as a set of mechanisms for combining, extending, and constraining document types.

1.1 About the DITA 2.0 specification

The DITA specification includes grammar files and the written specification.

1.1.1 XML grammar files

The XML grammar files are available in RELAX NG (RNG), and XML Document-Type Definitions (DTD).

While the files should define the same DITA elements, the RELAX NG grammars are normative if there is a discrepancy.

1.1.2 Written specification

The specification is written for implementers of the DITA standard, including tool developers and XML architects who develop specializations.

The specification contains several parts:

- Introduction
- Architectural specification
- Language reference
- Conformance statement
- Appendices

The specification is available in the following formats:

- DITA source
- PDF
- HTML5 (available from the OASIS Web site, authoritative)
- ZIP of HTML5 (optimized for local use)

1.2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT, "RECOMMEND", "MAY", and "OPTIONAL" in this document are to be interpreted as described in **[RFC 2119]** and **[RFC8174]** when, and only when, they appear in all capitals, as shown here.

1.3 IPR policy

This specification is provided under the RF on Limited Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/dita/ipr.php).

1.4 Normative references

Normative references are references to external documents or resources to which implementers of DITA *MUST* comply.

[RFC 2119]
Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC 3986]
Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC 5646]
Phillips, A., Ed., and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <http://www.rfc-editor.org/info/rfc5646>.

[RFC8174]
Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <http://www.rfc-editor.org/info/rfc8174>.

[XML 1.0]
Extensible Markup Language (XML) 1.0 (Fifth Edition), T Bray, J. Paoli, M. E. Maler, F. Yergeau, Editors, W3C Recommendation, 26 November 2008, http://www.w3.org/TR/2008/REC-xml-20081126/. Latest version available at http://www.w3.org/TR/xml.

[XML 1.1]
Extensible Markup Language (XML) 1.1 (Second Edition), T. Bray, J. Paoli, M. E. Maler, F. Yergeau, J. Cowan, Editors, W3C Recommendation, 16 August 2006, http://www.w3.org/TR/2006/REC-xml11-20060816/. Latest version available at http://www.w3.org/TR/xml11/.

1.5 Non-normative references

Non-normative references are references to external documents or resources that implementers of DITA might find useful.

Comment by Robert
Need to add a reference for HTML5. Not sure if those will replace, or supplement, the XHTML references.

[ISO 8601]
ISO/TC 154, *Data elements and interchange formats—Information interchange—Representation of dates and times*, 3rd edition, http://www.iso.org/iso/catalogue_detail?csnumber=40874, 12 December 2004.

[ISO/IEC 19757-3]
ISO/IEC JTC 1/SC 34 Document description and processing languages, *Information technology—Document Schema Definition Languages (DSDL)—Part 3: Rule-based validation—Schematron*, http://www.iso.org/iso/catalogue_detail.htm?csnumber=40833, 1 June 2006.

[Namespaces in XML 1.0]
Namespaces in XML 1.0 (Third Edition), T. Bray, D. Hollander, A. Layman, R. Tobin, H. S. Thompson, Editors, W3C Recommendation, 8 December 2009, http://www.w3.org/TR/2009/REC-xml-names-20091208/. Latest version available at http://www.w3.org/TR/xml-names.

[Namespaces in XML 1.1]
Namespaces in XML 1.1 (Second Edition), T. Bray, D. Hollander, A. Layman, R. Tobin, Editors, W3C Recommendation, 16 August 2006, http://www.w3.org/TR/2006/REC-xml-names11-20060816/. Latest version available at http://www.w3.org/TR/xml-names11/.

[OASIS Table Model]
XML Exchange Table Model Document Type Definition. Edited by Norman Walsh, 1999. Technical Memorandum TR 9901:1999. https://www.oasis-open.org/specs/tm9901.htm.

[RELAX NG]
J. Clark and M. Murata, editors, *RELAX NG Specification*, http://www.oasis-open.org/committees/relax-ng/spec-20011203.html, OASIS Committee Specification, 3 December 2001.

[RELAX NG Compact Syntax]
J. Clark, editor, *RELAX NG Compact Syntax*, http://www.oasis-open.org/committees/relax-ng/compact-20021121.html, OASIS Committee Specification, 21 November 2002.

[RELAX NG DTD Compatibility]
J. Clark and M. Murata, editors, *RELAX NG DTD Compatibility*, http://www.oasis-open.org/committees/relax-ng/compatibility-20011203.html, OASIS Committee Specification, 3 December 2001.

[XHTML 1.0]
XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition), S. Pemberton, Editor, W3C Recommendation, 1 August 2002, http://www.w3.org/TR/2002/REC-xhtml1-20020801. Latest version available at http://www.w3.org/TR/xhtml1.

[XHTML 1.1]
XHTML™ 1.1 – Module-based XHTML – Second Edition, S. McCarron, M. Ishikawa, Editors, W3C Recommendation, 23 November 2010, http://www.w3.org/TR/2010/REC-xhtml11-20101123. Latest version available at http://www.w3.org/TR/xhtml11/.

[XPointer 1.0]
XML Pointer Language (XPointer), S. J. DeRose, R. Daniel, P. Grosso, E. Maler, J. Marsh, N. Walsh, Editors, W3C Working Draft (work in progress), 16 August 2002, http://www.w3.org/TR/2002/WD-xptr-20020816/. Latest version available at http://www.w3.org/TR/xptr/.

[XML Catalogs 1.1]
OASIS Standard, *XML Catalogs Version 1.1*, 7 October 2005, https://www.oasis-open.org/committees/download.php/14809/xml-catalogs.html.

[xml:tm 1.0]
A. Zydroń, R. Raya, and B. Bogacki, editors, *XML Text Memory (xml:tm) 1.0 Specification*, http://www.gala-global.org/oscarStandards/xml-tm/, The Localization Industry Standards Association (LISA) xml:tm 1.0, 26 February 2007.

[XSL 1.0]
Extensible Stylesheet Language (XSL) Version 1.0, S. Adler, A. Berglund, J. S. Deach, T. Graham, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, S. Zilles, Editors, W3C Recommendation, 15 October 2001, http://www.w3.org/TR/2001/REC-xsl-20011015/. Latest version available at http://www.w3.org/TR/xsl/.

[XSL 1.1]
Extensible Stylesheet Language (XSL) Version 1.1, A. Berglund, Editor, W3C Recommendation, 5 December 2006, http://www.w3.org/TR/2006/REC-xsl11-20061205/. Latest version available at http://www.w3.org/TR/xsl11/.

[XSLT 2.0]

XSL Transformations (XSLT) Version 2.0, M. Kay, Editor, W3C Recommendation, 23 January 2007, http://www.w3.org/TR/2007/REC-xslt20-20070123/. Latest version available at http://www.w3.org/TR/xslt20.

[XTM 1.0]

S. Pepper and G. Moore, editors, *XML Topic Maps (XTM) 1.0*, http://www.topicmaps.org/xtm/index.html, TopicMaps.Org XTM 1.0, 2001.

1.6 Formatting conventions in the HTML5 version of the specification

Given the size and complexity of the specification, it is not generated as a single HTML5 file. Instead, each DITA topic is rendered as a separate HTML5 file.

The HTML5 version of the specification uses certain formatting conventions to aid readers in navigating through the specification and locating material easily: Link previews and navigation links.

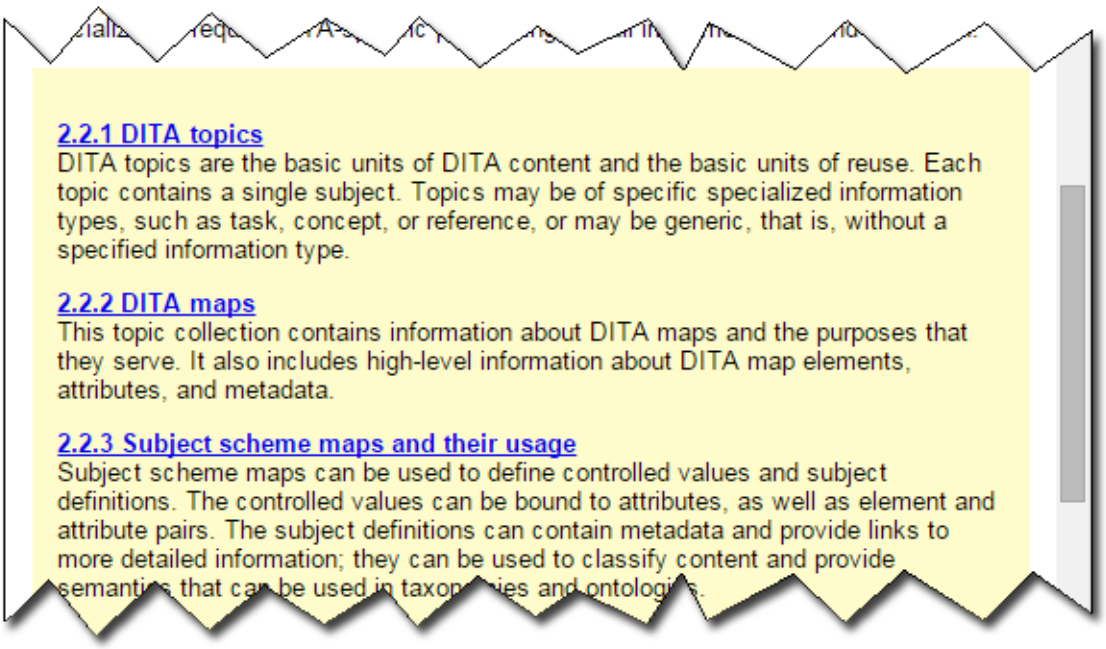
1.6.1 Link previews

The DITA specification uses the content of the DITA `<shortdesc>` element to provide link previews for its readers. These link previews are visually highlighted by a border and a colored background.

The link previews serve as enhanced navigation aids, enabling readers to more easily locate content. This usability enhancement is one of the ways in which the specification illustrates the capabilities of DITA and exemplifies DITA best practices.

The following screen capture illustrates how link previews are displayed in the HTML5 version of the specification:

Figure 1: Link previews



1.6.2 Navigation links

To ease readers in navigating from one topic to another, each HTML5 file generated by a DITA topic contains navigation links at the bottom.

Parent topic
: Takes readers to the parent topic, which is the topic referenced by the closest topic in the containment hierarchy

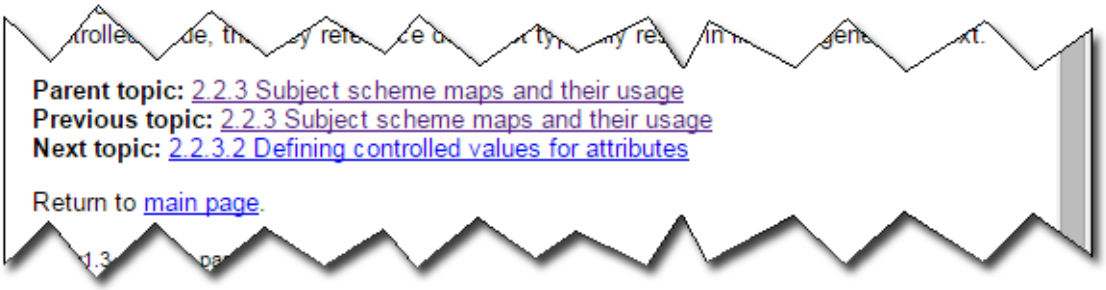
Previous topic
: Takes readers to the previous topic in the reading sequence

Next topic
: Takes readers to the next topic in the reading sequence

Return to main page
: Takes readers to the place in the table of contents for the current topic in the reading sequence

The following screen capture illustrates how navigation links are displayed in the HTML5 version of the specification:

Figure 2: Navigation links



When readers hover over the navigation links, the short description of the DITA topic also is displayed.

2 DITA terminology, notation, and conventions

The DITA specification uses specific notation and terms to define the components of the DITA standard.

2.1 Normative and non-normative information

The DITA specification contains normative and non-normative information.

Normative information
: Normative information is the formal portion of the specification that describes the rules and requirements that make up the DITA standard and which must be followed.

Non-normative information
: Non-normative information includes descriptions that provide background, examples, notes, and other useful information that are not formal requirements or rules that must be followed.

All information in the specification is normative unless it is an example, a note, an appendix, or is explicitly labeled as non-normative.

The DITA specification contains examples to help clarify or illustrate specific aspects of the specification. Because examples are specific rather than general, they might not illustrate all aspects or be the only way to accomplish or implement an aspect of the specification. Therefore all examples are non-normative.

2.2 Notation

Certain conventions are used throughout the specification to identify attributes and elements.

attribute types
: Attribute names are preceded by @ to distinguish them from elements or surrounding text, for example, the `@props` or the `@class` attribute.

element types
: Element names are delimited with angle brackets (< and >) to distinguish them from surrounding text, for example, the `<keyword>` or the `<prolog>` element.

In general, the unqualified use of the term *map* or *topic* can be interpreted to mean "a `<map>` element and any specialization of a `<map>` element " or "a `<topic>` element or any specialization of a `<topic>` element." Similarly, the unqualified use of an element type name (for example, `<p>`) can be interpreted to mean the element type or any specialization of the element type.

2.3 Basic DITA terminology

Certain terminology is used for basic DITA components.

DITA document

An XML document that conforms to the requirements of this specification.

A DITA document *MUST* have as its root element one of the following elements:

- `<map>` or a specialization of the `<map>` element
- `<topic>` or a specialization of the `<topic>` element
- `<dita>`, which cannot be specialized, but which allows documents with multiple sibling topics

DITA document type
A unique set of structural modules, domain modules, and constraint modules that taken together provide the XML element and attribute declarations that define the structure of DITA documents.

DITA document-type shell
A set of DTD or RELAX NG declarations that implement a DITA document type by using the rules and design patterns that are included in the DITA specification. A DITA document-type shell includes and configures one or more structural modules, zero or more domain modules, and zero or more constraint modules. With the exception of the optional declarations for the `<dita>` element and its attributes, DITA document-type shells do not declare any element or attribute types directly.

DITA element
An XML element instance whose type is a DITA element type. DITA elements must exhibit a `@class` attribute that has a value that conforms to the rules for specialization hierarchy specifications.

DITA element type
An element type that is either one of the base element types that are defined by the DITA specification, or a specialization of one of the base element types.

map instance
An occurrence of a map type in a DITA document.

map type
A map or a specialization of map that defines a set of relationships among topic instances.

structural type instance
An occurrence of a topic type or a map type in a DITA document.

topic instance
An occurrence of a topic type in a DITA document.

topic type
A topic or a specialization of topic that defines a complete unit of content.

2.4 Specialization terminology

Certain terminology is used to discuss DITA specialization.

base type
An element or attribute type that is not a specialization. All base types are defined by the DITA specification.

extension element
Within a vocabulary module, an element type that can be extended, replaced, or constrained for use in a DITA document type.

generalization
The process by which a specialized element is transformed into a less-specialized ancestor element or a specialized attribute is transformed into a less-specialized ancestor attribute. The original specialization-hierarchy information can be preserved in the generalized instance; this allows the original specialized type to be recreated from the generalized instance.

specialization
(1) The act of defining new element or attribute types as a semantic refinement of existing element or attribute types
(2) An element or attribute type that is a specialization of a base type
(3) A process by which a generalized element is transformed into one of its more specialized element types or a generalized attribute is transformed into a more specialized attribute.

specialization hierarchy
The sequence of element or attribute types, from the most general to most specialized, from which a given element or attribute type is specialized. The specialization hierarchy for a DITA element is formally declared through its `@class` attribute.

structural type
A topic type or map type.

2.5 DITA module terminology

Certain terminology is used to discuss DITA modules.

attribute domain module
A domain module that defines a specialization of either the `@base` or `@props` attribute.

constraint module
A set of declarations that imposes additional constraints onto the element or attribute types that are defined in a specific vocabulary module.

domain module
A vocabulary module that defines a set of element types or an attribute type that supports a specific subject or functional area.

element domain module
A domain module that defines one or more element types for use within maps or topics.

structural module
A vocabulary module that defines a top-level map type or topic type.

vocabulary module
A set of element or attribute declarations.

2.6 Linking and addressing terminology

Certain terminology is used for discussing linking and addressing.

referenced element
An element that is referenced by another DITA element. See also *referencing element*.
Example

Consider the following code sample from a `installation-reuse.dita` topic. The `<step>` element that it contains is a referenced element; other DITA topics reference the `<step>` element by using the `@conref` attribute.

```
<step id="run-startcmd-script">
    <cmd>Run the startcmd script that is applicable to your operating-system
environment.</cmd>
</step>
```

referencing element
An element that references another DITA element by specifying an addressing attribute. See also *referenced element* and *addressing attribute*
Example

The following `<step>` element is a referencing element. It uses the `@conref` attribute to reference a `<step>` element in the `installation-reuse.dita` topic.

```
<step conref="installation-reuse.dita#reuse/run-startcmd-script">
    <cmd/>
</step>
```

addressing attribute
An attribute, such as `@conref`, `@conkeyref`, `@keyref`, and `@href`, that specifies an address.

2.7 Key terminology

Certain terminology is used to discuss keys.

resource
For the purposes of keys and key resolution, one of the following:

- An object addressed by URI
- Metadata specified on a resource, such as a `@scope` or `@format` attribute
- Text or metadata located within a `<topicmeta>` element

key
A name for a resource. See 6.4.3 Using keys for addressing (80) for more information.

key definition
A `<topicref>` element that binds one or more key names to zero or more resources.

key reference
An attribute that references a key, such as `@keyref` or `@conkeyref`.

key space
A list of key definitions that are used to resolve key references.

effective key definition
The definition for a key within a key space that is used to resolve references to that key. A key might have multiple definitions within a key space, but only one of those definitions is effective.

key scope
A map or section of a map that defines its own key space and serves as the resolution context for its key references.

2.8 Map terminology

Certain terminology is used for DITA maps.

root map
The DITA map that is provided as input for a processor.

submap
A DITA map that is referenced with a `@scope` attribute that evaluates as "local". The value of the scope attribute might be explicitly set, be defaulted, or cascade from another element.

peer map
A DITA map that is referenced with a `@scope` attribute that evaluates as "peer". The value of the scope attribute might be explicitly set, be defaulted, or cascade from another element.

map branch
A `<topicref>` element or a specialization of `<topicref>`, along with any child elements and all resources that are referenced by the original element or its children.

2.9 File extensions

DITA uses certain file extensions for topics, maps, and conditional processing profiles.

Files that contain DITA content *SHOULD* use the following file extensions:

DITA topics
`*.dita` (preferred)

```
*.xml
```

DITA maps

```
*.ditamap
```

Conditional processing profiles

```
*.ditaval
```

3 Overview of DITA

The Darwin Information Typing Architecture (DITA) is an XML-based architecture for authoring, producing, and delivering topic-oriented, information-typed content that can be reused and single-sourced in a variety of ways. While DITA historically has been driven by the requirements of large-scale technical documentation authoring, management, and delivery, it is a standard that is applicable to any kind of publication or information that might be presented to readers, including interactive training and educational materials, standards, reports, business documents, trade books, travel and nature guides, and more.

DITA is designed for creating new document types and describing new information domains based on existing types and domains. The process for creating new types and domains is called specialization. Specialization enables the creation of specific, targeted XML grammars that can still use tools and design rules that were developed for more general types and domains; this is similar to how classes in an object-oriented system can inherit the methods of ancestor classes.

Because DITA topics are conforming XML documents, they can be readily viewed, edited, and validated using standard XML tools, although realizing the full potential of DITA requires using DITA-aware tools.

Comment by Kristen J Eberlein on 03 June 2019

This section of the spec now contains material about topics, maps, and metadata that was previously in the "DITA markup" section."

We need to carefully consider what of this content is appropriate. Some of it – information about map elements and attributes, metadata – is duplicated elsewhere. If we think it is useful to have a high-level overview here, we should mark it as non-normative – and point users to the normative coverage of the topic.

In a parallel move, I think we'll need to move coverage of critical DITA attributes into a more prominent place in the spec.

3.1 Basic concepts

DITA has been designed to satisfy requirements for information typing, semantic markup, modularity, reuse, interchange, and production of different deliverable forms from a single source. These topics provide an overview of the key DITA features and facilities that serve to satisfy these requirements.

DITA topics

In DITA, a topic is the basic unit of authoring and reuse. All DITA topics have the same basic structure: a title and, optionally, a body of content. Topics can be generic or more specialized; specialized topics represent more specific information types or semantic roles, for example, `<concept>`, `<task>`, or `<reference>` See DITA topics (19) for more information.

DITA maps

DITA maps are documents that organize topics and other resources into structured collections of information. DITA maps specify hierarchy and the relationships among the topics; they also provide the contexts in which keys are defined and resolved. See DITA maps (24) for more information.

Information typing

Information typing is the practice of identifying types of topics, such as concept, reference, and task, to clearly distinguish between different types of information. Topics that answer different reader questions (How do I? What is?) can be categorized with different information types. The base information types provided by DITA specializations (for example, technical content, machine industry,

and learning and training) provide starter sets of information types that can be adopted immediately by many technical and business-related organizations. See Information typing (21) for more information.

DITA addressing
DITA provides two addressing mechanisms. DITA addresses either are direct URI-based addresses, or they are indirect key-based addresses. Within DITA documents, individual elements are addressed by unique identifiers specified on the `@id` attribute. DITA defines two fragment-identifier syntaxes; one is the full fragment-identifier syntax, and the other is an abbreviated fragment-identifier syntax that can be used when addressing non-topic elements from within the same topic. See DITA addressing (73) for more information.

Content reuse
The DITA `@conref`, `@conkeyref`, `@conrefend`, and `@conaction` attributes provide mechanisms for reusing content within DITA topics or maps. These mechanisms can be used both to pull and push content. See Content reuse (112) for more information

Conditional processing
Conditional processing, also known as profiling, is the filtering or flagging of information based on processing-time criteria. See Conditional processing (117) for more information.

Configuration
A document-type shell is an XML grammar file that specifies the elements and attributes that are allowed in a DITA document. The document-type shell integrates structural modules, domain modules, and element-configuration modules. In addition, a document-type shell specifies whether and how topics can nest. See 8.2 Document-type configuration (142) for more information.

Specialization
The specialization feature of DITA allows for the creation of new element types and attributes that are explicitly and formally derived from existing types. This facilitates interchange of conforming DITA content and ensures a minimum level of common processing for all DITA content. It also allows specialization-aware processors to add specialization-specific processing to existing base processing. See Specialization (144) for more information.

Constraints
Constraint modules define additional constraints for vocabulary modules in order to restrict content models or attribute lists for specific element types, remove certain extension elements from an integrated domain module, or replace base element types with domain-provided, extension element types. See Constraints (155) for more information.

3.2 Producing different deliverables from a single source

DITA is designed to enable the production of multiple deliverable formats from a single set of DITA content. This means that many rendition details are specified neither in the DITA specification nor in the DITA content; the rendition details are defined and controlled by the processors.

Like many XML-based applications for human-readable documentation, DITA supports the separation of content from presentation. This is necessary when content is used in different contexts, since authors cannot predict how or where the material that they author will be used. The following features and mechanisms enable users to produce different deliverable formats from a single source:

DITA maps
Different DITA maps can be optimized for different delivery formats. For example, you might have a book map for printed output and another DITA map to generate online help; each map uses the same content set.

Specialization
The DITA specialization facility enables users to create XML elements that can provide appropriate rendition distinctions. Because the use of specializations does not impede interchange or

interoperability, DITA users can safely create the specializations that are demanded by their local delivery and rendition requirements, with a minimum of additional impact on the systems and business processes that depend on or use the content. While general XML practices suggest that element types should be semantic, specialization can be used to define element types that are purely presentational in nature. The highlighting domain is an example of such a specialization.

Conditional processing
Conditional processing makes it possible to have a DITA topic or map that contains delivery-specific content.

Content referencing
The conref mechanism makes it possible to construct delivery-specific maps or topics from a combination of generic components and delivery-context-specific components.

Key referencing
The keyref mechanism makes it possible to have key words be displayed differently in different deliverables. It also allows a single link to resolve to different targets in different deliverables.

@outputclass attribute
The `@outputclass` attribute provides a mechanism whereby authors can indicate specific rendition intent where necessary. Note that the DITA specification does not define any values for the `@outputclass` attribute; the use of the `@outputclass` attribute is processor specific.

While DITA is independent of any particular delivery format, it is a standard that supports the creation of human-readable content. As such, it defines some fundamental document components including paragraphs, lists, and tables. When there is a reasonable expectation that such basic document components be rendered consistently, the DITA specification defines default or suggested renderings.

3.3 DITA topics

DITA topics are the basic units of DITA content and the basic units of reuse. Each topic contains a single subject.

3.3.1 The topic as the basic unit of information

In DITA, a topic is the basic unit of authoring and reuse. All DITA topics have the same basic structure: a title and, optionally, a body of content. Topics can be generic or more specialized; specialized topics represent more specific information types or semantic roles, for example, `<concept>`, `<task>`, or `<reference>`

DITA topics consist of content units that can be as generic as sets of paragraphs and unordered lists or as specific as sets of instructional steps in a procedure or cautions to be considered before a procedure is performed. Content units in DITA are expressed using XML elements and can be conditionally processed using metadata attributes.

Classically, a DITA topic is a titled unit of information that can be understood in isolation and used in multiple contexts. It is short enough to address a single subject or answer a single question but long enough to make sense on its own and be authored as a self-contained unit. However, DITA topics also can be less self-contained units of information, such as topics that contain only titles and short descriptions and serve primarily to organize subtopics or links or topics that are designed to be nested for the purposes of information management, authoring convenience, or interchange.

DITA topics are used by reference from DITA maps. DITA maps enable topics to be organized in a hierarchy for publication. Large units of content, such as complex reference documents or book chapters, are created by nesting topic references in a DITA map. The same set of DITA topics can be used in any number of maps.

DITA topics also can be used and published individually; for example, one can represent an entire deliverable as a single DITA document that consists of a root topic and nested topics. This strategy can

accommodate the migration of legacy content that is not topic-oriented; it also can accommodate information that is not meaningful outside the context of a parent topic. However, the power of DITA is most fully realized by storing each DITA topic in a separate XML document and using DITA maps to organize how topics are combined for delivery. This enables a clear separation between how topics are authored and stored and how topics are organized for delivery.

3.3.2 The benefits of a topic-based architecture

Topics enable the development of usable and reusable content.

While DITA does not require the use of any particular writing practice, the DITA architecture is designed to support authoring, managing, and processing of content that is designed to be reused. Although DITA provides significant value even when reuse is not a primary requirement, the full value of DITA is realized when content is authored with reuse in mind. To develop topic-based information means creating units of standalone information that are meaningful with little or no surrounding context.

By organizing content into topics that are written to be reusable, authors can achieve several goals:

- Content is readable when accessed from an index or search, not just when read in sequence as part of an extended narrative. Since most readers do not read technical and business-related information from beginning to end, topic-oriented information design ensures that each unit of information can be read independently.
- Content can be organized differently for online and print delivery. Authors can create task flows and concept hierarchies for online delivery and create a print-oriented hierarchy to support a narrative content flow.
- Content can be reused in different collections. Since a topic is written to support random access (as by search), it should be understandable when included as part of various product deliverables. Topics permit authors to refactor information as needed, including only the topics that apply to each unique scenario.
- Content is more manageable in topic form whether managed as individual files in a traditional file system or as objects in a content management system.
- Content authored in topics can be translated and updated more efficiently and less expensively than information authored in larger or more sequential units.
- Content authored in topics can be filtered more efficiently, encouraging the assembly and deployment of information subsets from shared information repositories.

Topics written for reuse should be small enough to provide opportunities for reuse but large enough to be coherently authored and read. When each topic is written to address a single subject, authors can organize a set of topics logically and achieve an acceptable narrative content flow.

3.3.3 Disciplined, topic-oriented writing

Topic-oriented writing is a disciplined approach to writing that emphasizes modularity and reuse of concise units of information: topics. Well-designed DITA topics can be reused in many contexts, as long as writers are careful to avoid unnecessary transitional text.

Conciseness and appropriateness

Readers who are trying to learn or do something quickly appreciate information that is written in a structure that is easy to follow and contains only the information needed to complete that task or grasp a fact. Recipes, encyclopedia entries, car repair procedures; all serve up a uniquely focused unit of information. The topic contains everything required by the reader.

Locational independence

A well-designed topic is reusable in other contexts to the extent that it is context free, meaning that it can be inserted into a new document without revision of its content. A context-free topic avoids transitional text. Phrases like "As we considered earlier" or "Now that you have completed the initial step" make little sense if a topic is reused in a new context in which the relationships are different or

no longer exist. A well-designed topic reads appropriately in any new context because the text does not refer the reader outside the topic.

Navigational independence

Most print publications or web pages are a mixture of content and navigation. Internal links lead a reader through a sequence of choices as he or she navigates through a website. DITA supports the separation of navigation from content by assembling independent topics into DITA maps. Nonetheless, writers might want to provide links within a topic to additional topics or external resources. DITA does not prohibit such linking within individual topics. The DITA relationship table enables links between topics and to external content. Since it is defined in the DITA map, it is managed independently of the topic content.

Links in the content are best used for cross-references within a topic. Links from within a topic to additional topics or external resources are best avoided because they limit reuse of the topic. To link from a term or keyword to its definition, use the DITA keyref facility to avoid creating topic-to-topic dependencies that are difficult to maintain. See Key-based addressing (77)

3.3.4 Information typing

Information typing is the practice of identifying types of topics, such as concept, reference, and task, to clearly distinguish between different types of information. Topics that answer different reader questions (How do I? What is?) can be categorized with different information types. The base information types provided by DITA specializations (for example, technical content, machine industry, and learning and training) provide starter sets of information types that can be adopted immediately by many technical and business-related organizations.

Information typing has a long history of use in the technical documentation field to improve information quality. It is based on extensive research and experience, including Robert Horn's Information Mapping and Hughes Aircraft's STOP (Sequential Thematic Organization of Proposals) technique. Note that many DITA topic types are not necessarily closely connected with traditional Information Mapping.

Information typing is a practice designed to keep documentation focused and modular, thus making it clearer to readers, easier to search and navigate, and more suitable for reuse. Classifying information by type helps authors perform the following tasks:

- Develop new information more consistently
- Ensure that the correct structure is used for closely related kinds of information (retrieval-oriented structures like tables for reference information and simple sequences of steps for task information)
- Avoid mixing content types, thereby losing reader focus
- Separate supporting concept and reference information from tasks, so that users can read the supporting information if needed and ignore if it is not needed
- Eliminate unimportant or redundant detail
- Identify common and reusable subject matter

DITA currently defines a small set of well-established information types that reflects common practices in certain business domains, for example, technical communication and instruction and assessment. However, the set of possible information types is unbounded. Through the mechanism of specialization, new information types can be defined as specializations of the base topic type (`<topic>`) or as refinements of existing topics types, for example, `<concept>`, `<task>`, `<reference>`, or `<learningContent>`.

You need not use any of the currently-defined information types. However, where a currently-defined information type matches the information type of your content, use the currently-defined information type, either directly, or as a base for specialization. For example, for information that is procedural in nature,

use the task information type or a specialization of task. Consistent use of established information types helps ensure smooth interchange and interoperability of DITA content.

3.3.5 Generic topics

The element type `<topic>` is the base topic type from which all other topic types are specialized. All topics have the same basic structure.

For authors, typed content is preferred to support consistency in writing and presentation to readers. The generic topic type is best used only if authors are not trained in information typing or when a specialized topic type is inappropriate. The OASIS DITA standard provides several specialized topic types, including concept, task, and reference that are critical for technical content development.

For those pursuing specialization, specialize new topic types from appropriate ancestors to meet authoring and output requirements.

3.3.6 Topic structure

All topics have the same basic structure, regardless of topic type: title, description or abstract, prolog, body, related links, and nested topics.

All DITA topics must have an XML identifier (the `@id` attribute) and a title. The basic topic structure consists of the following parts, some of which are optional:

Topic element
: The topic element holds the required `@id` attribute and contains all other elements.

Title
: The title contains the subject of the topic.

Alternate titles
: Titles specifically for use in navigation or search. When not provided, the base title is used for all contexts.

Short description or abstract
: A short description of the topic or a longer abstract with an embedded short description. The short description might be used both in topic content (as the first paragraph), in generated summaries that include the topic, and in links to the topic. Alternatively, the abstract lets you create more complex introductory content and uses an embedded short description element to define the part of the abstract that is suitable for summaries and link previews.
While short descriptions are not required, they can make a dramatic difference to the usability of an information set and should generally be provided for all topics.

Prolog
: The prolog is the container for topic metadata, such as change history, audience, product, and so on.

Body
: The topic body contains the topic content: paragraphs, lists, sections, and other content that the information type permits.

Related links
: Related links connect to other topics. When an author creates a link as part of a topic, the topic becomes dependent on the other topic being available. To reduce dependencies between topics and thereby increase the ability to reuse each topic, authors can use DITA maps to define and manage links between topics, instead of embedding links directly in each related topic.

Nested topics
: Topics can be defined inside other topics. However, nesting requires special care because it can result in complex documents that are less usable and less reusable. Nesting might be appropriate for

information that is first converted from desktop publishing or word processing files or for topics that are unusable independent from their parent or sibling topics.
The rules for topic nesting can be configured in a document-type shells. For example, the standard DITA configuration for concept topics only allows nested concept topics. However, local configuration of the concept topic type could allow other topic types to nest or disallow topic nesting entirely. In addition, the `@chunk` attribute enables topics to be equally re-usable regardless of whether they are separate or nested. The standard DITA configuration for ditabase document-type documents allows unrestricted topic nesting and can be used for holding sets of otherwise unrelated topics that hold re-usable content. It can also be used to convert DITA topics from non-DITA legacy source without first determining how individual topics should be organized into separate XML documents.

3.3.7 Topic content

The content of all topics, regardless of topic type, is built on the same common structures.

Topic body

The topic body contains all content except for that contained in the title or the short description/abstract. The topic body can be constrained to remove specific elements from the content model; it also can be specialized to add additional specialized elements to the content model. The topic body can be generic while the topic title and prolog are specialized.

Sections and examples

The body of a topic might contain divisions, such as sections and examples. They might contain block-level elements like titles and paragraphs and phrase-level elements like API names or text. It is recommend that sections have titles, whether they are entered directly into the `<title>` element or rendered using a fixed or default title.
Either body divisions or untitled sections or examples can be used to delimit arbitrary structures within a topic body. However, body divisions can nest, but sections and examples cannot contain sections.

<sectiondiv>

The `<sectiondiv>` element enables the arbitrary grouping of content within a section for the purpose of content reuse. The `<sectiondiv>` element does not include a title. For content that requires a title, use `<section>` or `<example>`.

<bodydiv>

The `<bodydiv>` element enables the arbitrary grouping of content within the body of a topic for the purpose of content reuse. The `<bodydiv>` element does not include a title. For content that requires a title, use `<section>` or `<example>`.

** **

The `<div>` element enables the arbitrary grouping of content within a topic. The `<div>` element does not include a title. For content that requires a title, use `<section>` or `<example>` or, possibly, `<fig>`.

Block-level elements

Paragraphs, lists, figures, and tables are types of "block" elements. As a class of content, they can contain other blocks, phrases, or text, though the rules vary for each structure.

Phrases and keywords

Phrase level elements can contain markup to label parts of a paragraph or parts of a sentence as having special semantic meaning or presentation characteristics, such as `<uicontrol>` or `<b>`. Phrases can usually contain other phrases and keywords as well as text. Keywords can only contain text.

Images

Images can be inserted to display photographs, illustrations, screen captures, diagrams, and more. At the phrase level, they can display trademark characters, icons, toolbar buttons, and so forth.

Multimedia

The `<object>` element enables authors to include multimedia, such as diagrams that can be rotated and expanded. The `<foreign>` element enables authors to include media within topic content, for example, SVG graphics, MathML equations, and so on.

3.4 DITA maps

This topic collection contains information about DITA maps and the purposes that they serve. It also includes high-level information about DITA map elements, attributes, and metadata.

3.4.1 Definition of DITA maps

DITA maps are documents that organize topics and other resources into structured collections of information. DITA maps specify hierarchy and the relationships among the topics; they also provide the contexts in which keys are defined and resolved.

Maps draw on a rich set of existing best practices and standards for defining information models, such as hierarchical task analysis. They also support the definition of non-hierarchical relationships, such as matrices and groups, which provide a set of capabilities that has similarities to Resource Description Framework (RDF) and ISO topic maps.

DITA maps use `<topicref>` elements to reference DITA topics, DITA maps, and non-DITA resources, for example, HTML and TXT files. The `<topicref>` elements can be nested or grouped to create relationships among the referenced topics, maps, and non-DITA files; the `<topicref>` elements can be organized into hierarchies in order to represent a specific order of navigation or presentation.

DITA maps impose an architecture on a set of topics. Information architects can use DITA maps to specify what DITA topics are needed to support a given set of user goals and requirements; the sequential order of the topics; and the relationships that exist among those topics. Because DITA maps provide this context for topics, the topics themselves can be relatively context-free; they can be used and reused in multiple different contexts.

DITA maps often represent a single deliverable, for example, a specific Web site, a printed publication, or the online help for a product. DITA maps also can be subcomponents for a single deliverable, for example, a DITA map might contain the content for a chapter in a printed publication or the troubleshooting information for an online help system. The DITA specification provides specialized map types; book maps represent printed publications, subject scheme maps represent taxonomic or ontological classifications, and learning maps represent formal units of instruction and assessment. However, these map types are only a starter set of map types reflecting well-defined requirements.

DITA maps establish relationships through the nesting of `<topicref>` elements and the application of the `@collection-type` attribute. Relationship tables also can be used to associate topics with each other based on membership in the same row; for example, task topics can be associated with supporting concept and reference topics by placing each group in cells of the same row. During processing, these relationships can be rendered in different ways, although they typically result in lists of "Related topics" or "For more information" links. Like many aspects of DITA, the details about how such linking relationships are presented is determined by the DITA processor.

DITA maps also define keys and organize the contexts (*key scopes*) in which key references are resolved.

3.4.2 Purpose of DITA maps

DITA maps enable the scalable reuse of content across multiple contexts. They can be used by information architects, writers, and publishers to plan, develop, and deliver content.

DITA maps support the following uses:

Defining an information architecture
Maps can be used to define the topics that are required for a particular audience, even before the topics themselves exist. DITA maps can aggregate multiple topics for a single deliverable.

Defining what topics to build for a particular output
Maps reference topics that are included in output processing. Information architects, authors, and publishers can use maps to specify a set of topics that are processed at the same time, instead of processing each topic individually. In this way, a DITA map can serve as a manifest or bill of materials.

Defining navigation
Maps can define the online navigation or table of contents for a deliverable.

Defining related links
Maps define relationships among the topics they reference. These relationships are defined by the nesting of elements in the DITA map, relationship tables, and the use of elements on which the `@collection-type` attribute is set. On output, these relationships might be expressed as related links or the hierarchy of a table of contents (TOC).

Defining an authoring context
The DITA map can define the authoring framework, providing a starting point for authoring new topics and integrating existing ones.

Defining keys and key scopes
Maps can define keys, which provide an indirect addressing mechanism that enhances portability of content. The keys are defined by `<topicref>` elements or specializations of `<topicref>` elements, such as `<keydef>`. The `<keydef>` element is a convenience element; it is a specialized type of a `<topicref>` element with the following attributes:

- A required `@keys` attribute
- A `@processing-role` attribute with a default value of "resource-only".

Maps also define the context or contexts for resolving key-based references, such as elements that specify the `@keyref` or `@conkeyref` attribute. Elements within a map structure that specify a `@keyscope` attribute create a new context for key reference resolution. Key references within such elements are resolved against the set of effective key definitions for that scope.

Specialized maps can provide additional semantics beyond those of organization, linking, and indirection. For example, the subjectScheme map specialization adds the semantics of taxonomy and ontology definition.

3.4.3 DITA map attributes

DITA maps have unique attributes that are designed to control the way that relationships are interpreted for different output purposes. In addition, DITA maps share many metadata and linking attributes with DITA topics.

DITA maps often encode structures that are specific to a particular medium or output, for example, Web pages or a PDF document. Attributes, such as `@deliveryTarget` and `@toc`, are designed to help processors interpret the DITA map for each kind of output.

Many of the following attributes are not available in DITA topics; individual topics, once separated from the high-level structures and dependencies associated with a particular kind of output, should be entirely reusable regardless of the intended output format.

@collection-type
The `@collection-type` attribute specifies how the children of a `<topicref>` element relate to their parent and to each other. This attribute, which is set on the parent element, typically is used by processors to determine how to generate navigation links in the rendered topics. For example, a `@collection-type` value of "sequence" indicates that children of the specifying `<topicref>` element represent an ordered sequence of topics; processors might add numbers to the list of child topics or generate next/previous links for online presentation. This attribute is available in topics on the `<linklist>` and `<linkpool>` elements, where it has the same behavior. Where the `@collection-type` attribute is available on elements that cannot directly contain elements, the behavior of the attribute is undefined.

@linking

By default, the relationships between the topics that are referenced in a map are reciprocal:

- Child topics link to parent topics and vice versa.
- Next and previous topics in a sequence link to each other.
- Topics in a family link to their sibling topics.
- Topics referenced in the table cells of the same row in a relationship table link to each other. A topic referenced within a table cell does not (by default) link to other topics referenced in the same table cell.

This behavior can be modified by using the `@linking` attribute, which enables an author or information architect to specify how a topic participates in a relationship. The following values are valid:

linking="none"
Specifies that the topic does not exist in the map for the purposes of calculating links.

linking="sourceonly"
Specifies that the topic will link to its related topics but not vice versa.

linking="targetonly"
Specifies that the related topics will link to it but not vice versa.

linking="normal"
Default value. It specifies that linking will be reciprocal (the topic will link to related topics, and they will link back to it).

Authors also can create links directly in a topic by using the `<xref>` or `<link>` elements, but in most cases map-based linking is preferable, because links in topics create dependencies between topics that can hinder reuse.

Note that while the relationships between the topics that are referenced in a map are reciprocal, the relationships merely *imply* reciprocal links in generated output that includes links. The rendered navigation links are a function of the presentation style that is determined by the processor.

@toc
Specifies whether topics are excluded from navigation output, such as a Web site map or an online table of contents. By default, `<topicref>` hierarchies are included in navigation output; relationship tables are excluded.

@search
Specifies whether the topic is included in search indexes.

@chunk

Specifies that the processor generates an interim set of DITA topics that are used as the input for the final processing. This can produce the following output results:

- Multi-topic files are transformed into smaller files, for example, individual HTML files for each DITA topic.
- Individual DITA topics are combined into a single file.

Specifying a value for the `@chunk` attribute on a `<map>` element establishes chunking behavior that applies to the entire map, unless overridden by `@chunk` attributes that are set on more specific elements in the DITA map. For a detailed description of the `@chunk` attribute and its usage, see 5.4 Chunking (57).

@copy-to

In most situations, specifies whether a duplicate version of the topic is created when it is transformed. This duplicate version can be either literal or virtual. The value of the `@copy-to` attribute specifies the uniform resource identifier (URI) by which the topic can be referenced by a `@conref` attribute, `<topicref>` element, or `<xref>` element. The duplication is a convenience for output processors that use the URI of the topic to generate the base address of the output. The `@keys` and `@keyref` attributes provide an alternative mechanism; they enable references to topics in specific-use contexts.

The `@copy-to` attribute also can be used to specify the name of a new chunk when topics are being chunked; it also can be used to determine the name of the stub topic that is generated from a `<topicref>` element that contains a title but does not specify a target. In both of those cases, no duplicate version of the topic is generated.

For information on how the `@copy-to` attribute can be used with the `@chunk` attribute, see 5.4 Chunking (57).

@processing-role

Specifies whether the topic or map referenced is processed normally or treated as a resource that is only included in order to resolve key or content references.

processing-role="normal"

The topic is a readable part of the information set. It is included in navigation and search results. This is the default value for the `<topicref>` element.

processing-role="resource-only"

The topic is used only as a resource for processing. It is not included in navigation or search results, nor is it rendered as a topic. This is the default value for the `<keydef>` element.

If the `@processing-role` attribute is not specified locally, the value cascades from the closest element in the containment hierarchy.

@cascade

Specifies whether the default rules for the cascading of metadata attributes in a DITA map apply. The following values are specified:

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).

Processors also *MAY* define additional values for the `@cascade` attribute.

For more information, see 3.4.4.4 Example: How the cascade attribute functions (31).

@keys
Specifies one or more key names.

@keyscope
Defines a new scope for key definition and resolution, and gives the scope one or more names. For more information about key scopes, see 6.4 Indirect key-based addressing (77).

Attributes in the list above are used exclusively or primarily in maps, but many important map attributes are shared with elements in topics. DITA maps also use many of the following attributes that are used with linking elements in DITA topics, such as `<link>` and `<xref>`:

- `@format`
- `@href`
- `@keyref`
- `@scope`
- `@type`

The following metadata and reuse attributes are used by both DITA maps and DITA topics:

- `@rev`, `@status`, `@importance`
- `@dir`, `@xml:lang`, `@translate`
- `@id`, `@conref`, `@conrefend`, `@conkeyref`, `@conaction`
- `@props` and any attribute specialized from `@props` (including those integrated by default in OASIS shells: `@audience`, `@deliveryTarget`, `@platform`, `@product`, `@otherprops`)
- `@search`

When new attributes are specialized from `@props` or `@base` as a domain, they can be incorporated into both map and topic structural types.

3.4.4 Examples of DITA maps

This section of the specification contains simple examples of DITA maps. The examples illustrate a few of the ways that DITA maps are used.

3.4.4.1 Example: DITA map that references a subordinate map

This example illustrates how one map can reference a subordinate map using either `<mapref>` or the basic `<topicref>` element.

The following code sample illustrates how a DITA map can use the specialized `<mapref>` element to reference another DITA map:

```
<map>
  <title>DITA work at OASIS</title>
  <topicref href="oasis-dita-technical-committees.dita">
    <topicref href="dita_technical_committee.dita"/>
    <topicref href="dita_adoption_technical_committee.dita"/>
  </topicref>
  <mapref href="oasis-processes.ditamap"/>
  <!-- ... -->
</map>
```

The <mapref> element is a specialized <topicref> intended to make it easier to reference another map; use of <mapref> is not required for this task. This map also could be tagged in the following way:

```
<map>
  <title>DITA work at OASIS</title>
  <topicref href="oasis-dita-technical-committees.dita">
    <topicref href="dita_technical_committee.dita"/>
    <topicref href="dita_adoption_technical_committee.dita"/>
  </topicref>
<topicref href="oasis-processes.ditamap" format="ditamap"/>
<!-- ... -->
</map>
```

With either of the above examples, during processing, the map is resolved in the following way:

```
<map>
  <title>DITA work at OASIS</title>
  <topicref href="oasis-dita-technical-committees.dita">
    <topicref href="dita_technical_committee.dita"/>
    <topicref href="dita_adoption_technical_committee.dita"/>
  </topicref>
  <!-- Contents of the oasis-processes.ditamap file -->
  <topicref href="oasis-processes.dita">
    <!-- ... -->
  </topicref>
  <!-- ... -->
</map>
```

3.4.4.2 Example: DITA map with a simple relationship table

This example illustrates how to interpret a basic three-column relationship table used to maintain links between concept, task, and reference material.

The following example contains the markup for a simple relationship table:

```
<map>
<!-- ... -->
<reltable>
  <relheader>
    <relcolspec type="concept"/>
    <relcolspec type="task"/>
    <relcolspec type="reference"/>
  </relheader>
  <relrow>
    <relcell>
      <topicref href="A.dita"/>
    </relcell>
    <relcell>
      <topicref href="B.dita"/>
    </relcell>
    <relcell>
      <topicref href="C1.dita"/>
      <topicref href="C2.dita"/>
    </relcell>
  </relrow>
</reltable>
</map>
```

A DITA-aware tool might represent the relationship table graphically:

type="concept"	type="task"	type="reference"
A	B	C1 C2

When the output is generated, the topics contain the following linkage:

A
Links to B, C1, and C2

B
Links to A, C1, and C2

C1, C2
Links to A and B

3.4.4.3 Example: How the @collection-type and @linking attributes determine links

In this scenario, a simple map establishes basic hierarchical and relationship table links. The `@collection-type` and `@linking` attributes are then added to modify how links are generated.

The following example illustrates how linkage is defined in a DITA map:

Figure 3: Simple linking example

```
<topicref href="A.dita" collection-type="sequence">
  <topicref href="A1.dita"/>
  <topicref href="A2.dita"/>
</topicref>
<reltable>
  <relrow>
    <relcell><topicref href="A.dita"/></relcell>
    <relcell><topicref href="B.dita"/></relcell>
  </relrow>
</reltable>
```

When the output is generated, the topics contain the following linkage. Sequential (next/previous) links between A1 and A2 are present because of the `@collection-type` attribute on the parent:

A
Links to A1, A2 as children
Links to B as related

A1
Links to A as a parent
Links to A2 as next in the sequence

A2
Links to A as a parent
Links to A1 as previous in the sequence

B
Links to A as related

The following example illustrates how setting the `@linking` attribute can change the default behavior:

Figure 4: Linking example with the @linking attribute

```
<topicref href="A.dita" collection-type="sequence">
  <topicref href="B.dita" linking="none"/>
  <topicref href="A1.dita"/>
  <topicref href="A2.dita"/>
</topicref>
<reltable>
  <relrow>
    <relcell><topicref href="A.dita"/></relcell>
    <relcell linking="sourceonly"><topicref href="B.dita"/></relcell>
```

```
  </relrow>
</reltable>
```

When the output is generated, the topics contain the following linkage:

A
Links to A1, A2 as children
Does not link to B as a child or related topic

A1
Links to A as a parent
Links to A2 as next in the sequence
Does not link to B as previous in the sequence

A2
Links to A as a parent
Links to A1 as previous in the sequence

B
Links to A as a related topic

3.4.4.4 Example: How the @cascade attribute functions

The following example illustrates how the `@cascade` attribute can be used to fine tune how the values for the `@platform` attribute apply to topics referenced in a DITA map.

Here a DITA map contains a collection of topics that apply to Windows, Linux, and Macintosh OS; it also contains a topic that is only applicable to users running the application on Linux.

```
<map product="PuffinTracker" platform="win linux mac" cascade="nomerge">
  <title>Puffin Tracking Software</title>
  <topicref href="introduction.dita"/>
  <topicref href="setting-up-the-product.dita"/>
  <topicref href="linux-instructions.dita" platform="linux"/>
</map>
```

The values of the `@platform` attribute set at the map level cascade throughout the map and apply to the `introduction.dita` and `setting-up-the-product.dita` topics. However, since the value of the `@cascade` attribute is set to "nomerge", the value of the `@platform` attribute for the `linux-instructions.dita` topic does not merge with the values that cascade from above in the DITA map. The effective value of the `@platform` attribute for `linux-instructions.dita` is "linux".

The same results are produced by the following mark-up:

```
<map product="PuffinTracker" platform="win linux mac">
  <title>Puffin Tracking Software</title>
  <topicref href="introduction.dita"/>
  <topicref href="setting-up-the-product.dita"/>
  <topicref href="linux-instructions.dita" platform="linux" cascade="nomerge"/>
</map>
```

3.5 DITA metadata

Metadata can be applied in both DITA topics and DITA maps. Metadata that is assigned in DITA topics can be supplemented or overridden by metadata that is assigned in a DITA map; this design facilitates the reuse of DITA topics in different DITA maps and use-specific contexts.

3.5.1 Metadata elements

The metadata elements, many of which map to Dublin core metadata, are available in topics and DITA maps. This design enables authors and information architects to use identical metadata markup in both topics and maps.

The `<metadata>` element is a wrapper element that contains many of the metadata elements. In topics, the `<metadata>` element is available in the `<prolog>` element. In maps, the `<metadata>` element is available in the `<topicmeta>` element.

Note Not all metadata elements are available in the `<metadata>` element. However, they are available in either the topic `<prolog>` element or the map `<topicmeta>` element.

In DITA maps, the metadata elements also are available directly in the `<topicmeta>` element. Collections of metadata can be shared between DITA maps and topics by using the conref or keyref mechanism.

In general, specifying metadata in a `<topicmeta>` element is equivalent to specifying it in the `<prolog>` element of a referenced topic. The value of specifying the metadata at the map level is that the topic then can be reused in other maps where different metadata might apply. Many items in the `<topicmeta>` element also cascade to nested `<topicref>` elements within the map.

Related information

Dublin Core Metadata Initiative (DCMI)

3.5.2 Metadata attributes

The metadata attributes specify properties of the content that can be used to determine how the content is processed. Specialized metadata attributes can be defined to enable specific business-processing needs, such as semantic processing and data mining.

Metadata attributes typically are used for the following purposes:

- Filtering content based on the attribute values, for example, to suppress or publish profiled content
- Flagging content based on the attribute values, for example, to highlight specific content on output
- Performing custom processing, for example, to extract business-critical data and store it in a database

Typically `@audience`, `@platform`, `@product`, `@otherprops`, `@props`, `@deliveryTarget`, and specializations of the `@props` attributes are used for filtering; the same attributes plus the `@rev` attribute are used for flagging. The `@status` and `@importance` attributes, as well as custom attributes specialized from `@base`, are used for application-specific behavior, such as identifying metadata to aid in search and retrieval.

3.5.2.1 Filtering and flagging attributes

Conditional-processing attributes are available on most elements.

@product
The product that is the subject of the discussion.

@platform
The platform on which the product is deployed.

@audience
The intended audience of the content.

@deliveryTarget
The intended delivery target of the content, for example, "html", "pdf", or "epub".

The `@deliveryTarget` attribute is specialized from the `@props` attribute. It is defined in the deliveryTargetAttDomain, which is integrated into all OASIS-provided document-type shells. If this domain is not integrated into a given document-type shell, the `@deliveryTarget` attribute will not be available.

@rev
The revision or draft number of the current document. (This is used only for flagging.)

@otherprops
Other properties that do not require semantic identification.

@props
A generic conditional processing attribute that can be specialized to create new semantic conditional-processing attributes.

Related concepts

Conditional processing (profiling) (117)
Conditional processing, also known as profiling, is the filtering or flagging of information based on processing-time criteria.

Related reference

DITAVAL elements (356)
A conditional processing profile (DITAVAL file) is used to identify which values are to be used for conditional processing during a particular output, build, or some other purpose. The profile should have an extension of `.ditaval`.

3.5.2.2 Other processing attributes

Other attributes are still considered metadata on an element, but they are not designed for filtering or flagging.

@importance
The degree of priority of the content. This attribute takes a single value from an enumeration.

@status
The current state of the content. This attribute takes a single value from an enumeration.

@base
A generic attribute that has no specific purpose, but is intended to act as the basis for specialized attributes that have a simple value syntax like the conditional processing attributes (one or more alphanumeric values separated by whitespace or parenthesized groups of values).

@outputclass
Provides a label on one or more element instances, typically to specify a role or other semantic distinction. As the `@outputclass` attribute does not provide a formal type declaration or the structural consistency of specialization, use it sparingly, usually only as a temporary measure while a

specialization is developed. For example, `<uicontrol>` elements that define button labels could be distinguished by adding an `@outputclass` attribute:

```
<uicontrol outputclass="button">Cancel</uicontrol>
```

The value of the `@outputclass` attribute can be used to trigger XSLT or CSS rules, while providing a mapping to be used for future migration to a more specialized set of user interface elements.

3.5.2.3 Translation and localization attributes

DITA elements have several attributes that support localization and translation.

@xml:lang
Identifies the language of the content, using the standard language and country codes. For instance, French Canadian is identified by the value fr-CA. The `@xml:lang` attribute asserts that all content and attribute values within the element bearing the attribute are in the specified language, except for contained elements that declare a different language.

@translate
Determines whether the element requires translation. A default value can often be inferred from the element type. For example, `<apiname>` might be untranslated by default, whereas `<p>` might be translated by default.

@dir
Determines the direction in which the content is rendered.

3.5.2.4 Architectural attributes

The architectural attributes specify the version of DITA that the content supports; they also identify the DITA domains, structural types, and specializations that are in use by the content.

The architectural attributes should not be marked up in the source DITA map and topics. Instead, the values of the architectural attributes are handled by the processor when the content is processed, preferably through defaults set in the XML grammar. This practice ensures that the DITA content instances do not specify invalid values for the architectural attributes.

The architectural attributes are as follows:

@class
This attribute identifies the specialization hierarchy for the element type.

@specializations
This attribute identifies the specialized attributes that are used in a map or topic.

@DITAArchVersion
This attribute identifies the version of the DITA architecture that is used by the XML grammar. The attribute is declared in a DITA namespace to allow namespace-sensitive tools to detect DITA markup.

To make the document instance usable in the absence of an XML grammar, a normalization process can set the architectural attributes in the document instance.

3.5.3 Metadata in maps and topics

Information about topics can be specified as metadata on the map, as attributes on the `<topicref>` element, or as metadata attributes or elements in the topic itself. By default, metadata in the map supplements or overrides metadata that is specified at the topic level.

DITA map: Metadata elements

At the map level, properties can be set by using metadata elements. They can be set for an individual topic, for a set of topics, or globally for the entire document. The metadata elements are

authored within a <topicmeta> element, which associates metadata with the parent element and its children. Because the topics in a branch of the hierarchy typically have some common subjects or properties, this is a convenient mechanism to define properties for a set of topics. For example, the <topicmeta> element in a <relcolspec> can associate metadata with all the topics that are referenced in the <reltable> column.

A map can override or supplement everything about a topic except its primary title and body content. All the metadata elements that are available in a topic also are available in a map. In addition, a map can use @copy-to provide alternate titles and a short description. The alternate titles can override their equivalent titles in the topic.

DITA map: Attributes of the <topicref> element
At the map level, properties can be set as attributes of the <topicref> element.

DITA topic
Within a topic, authors can either set metadata attributes on the root element or add metadata elements in the <prolog> element.

When the same metadata element or attribute is specified in both a map and a topic, by default the value in the map takes precedence; the assumption here is that the author of the map has more knowledge of the reusing context than the author of the topic.

3.5.4 Context hooks and window metadata for user assistance

Context hook information specified in the <resourceid> element in the DITA map or in a DITA topic enables processors to generate the header, map, alias and other types of support files that are required to integrate the user assistance with the application. Some user assistance topics might need to be displayed in a specific window or viewport, and this windowing metadata can be defined in the DITA map within the <ux-window> element.

Context hook and windowing information is ignored if the processor does not support this metadata.

User interfaces for software application often are linked to user assistance (such as help systems and tool tips) through *context hooks*. Context hooks are identifiers that associate a part of the user interface with the location of a help topic. Context hooks can be direct links to URIs, but more often they are indirect links (numeric context identifiers and context strings) that can processed into external resource files. These external resource and mapping files are then used directly by context-sensitive help systems and other downstream applications.

Context hooks can define either one-to-one or one-to-many relationships between user interface controls and target help content.

The metadata that is available in <resourceid> and <ux-window> provides flexibility for content developers:

- You can overload maps and topics with all the metadata needed to support multiple target help systems. This supports single-sourcing of help content and help metadata.
- You can choose whether to add <resourceid> metadata to <topicref> elements, <prolog> elements, or both. Context-dependent metadata might be best be kept with maps, while persistent, context-independent metadata might best stay with topics in <prolog> elements

Context hook information is defined within DITA topics and DITA maps through attributes of the <resourceid> element.

In some help systems, a topic might need to be displayed in a specifically sized or featured window. For example, a help topic might need to be displayed immediately adjacent to the user interface control that it supports in a window of a specific size that always remains on top, regardless of the focus within the

operating system. Windowing metadata can be defined in the DITA map within the `<ux-window>` element.

The `<ux-window>` element provides the `@top`, `@left`, `@height`, `@width`, `@on-top`, `@features`, `@relative`, and `@full-screen` attributes.

Related reference

resourceid (306)
A resource ID provides an identifier for applications that must use their own identifier scheme, such as context-sensitive help systems and databases.

ux-window

4 Determining effective attribute values

Topic to be moved to more appropriate location: how to determine effective attribute values.

Need to reconcile the two different existing lists, in 5.3.1.2 Processing cascading attributes in a map (51) and 5.2.3 Binding controlled values to an attribute (40).

From "processing cascading attributes"

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 5.3.3 Map-to-map cascading behaviors (54) for more details.
8. Repeat steps 1 (37) to 4 (37) 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.

From "binding controlled values"

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

5 DITA map processing

Introduction to this chapter to be written later, when content is more stable.

5.1 DITA maps and their usage

New topic cluster to hold normative architectural content about DITA maps. Currently holds notes about material that we intend to cover in the new topic cluster.

Topical areas

- How `<topicref>` elements establish hierarchies including parent/child relationships and next/previous relationships.
- Map-group elements
  - Role as convenience elements—in most (all?) cases, the same function can be accomplished with base elements. For example, `<topichead>` is effectively no different than `<topicref>` with nothing but a title.
  - Special role of `<topicgroup>`, which does not contribute to hierarchy
- How relationship tables establish linking relationships between topic references
- Meaning of titles (and navigation titles) on maps, submaps, mapgroup elements, and relationship tables
- Link relationships created by attributes and nesting in DITA maps

Current topics with applicable content

Topic	Applicable content
3.4.5.1 Example: DITA map that references a subordinate map	Resolution of a submap.
3.4.5.2 Example: DITA map with a simple relationship table	How links are generated from a relationship table; how processors might represent a relationship table.
3.4.5.3 Example: How the `@collection-type` and `@linking` determine links	Effect of `@collection-type` and `@linking` attributes on generated links.
6.1 Navigation	Container topic; incorporate into new "DITA maps and their usage" cluster.
6.1.1 Table of contents	All content is applicable and needs to be incorporated into the new "DITA maps and their usage" cluster – Closest thing we currently have to a topic about how maps create hierarchies.
9.3.1.1 `<map>`	Relationships between topics created by map hierarchy or `@collection-type` attribute; role of titles, especially in submaps.
9.3.1.2 `<topicref>`	Role of `<topicref>` nesting in creating containment hierarchies and parent-child relationships.
9.3.1.6 `<reltable>`	Relationship table titles – Processing expectations for relationship tables (not rendered, used to generate links) – “Within a map tree, the effective relationship table is the union of all relationship tables in the map.” – How a DITA-aware tool might represent the `<reltable>` element graphically.

Topic	Applicable content
9.3.1.10 `<relcolspec>`	How labels for related links from a relationship table are generated.
9.3.2.3 `<mapref>`	“The hierarchy of the referenced map is merged into the container map at the position of the reference, and the relationship tables of the child map are added to the parent map.”
9.3.2.4 `<topicgroup>`	How processors handle navigation titles within `<topicgroup>` elements.
9.8.13.10 The `@format` attribute	How processors determine the value of the `@format` attribute when it is not explicitly set.

Possible new topics

- DITA maps
- Relationship tables
- Creating navigational hierarchies
- Defining links between resources

5.2 Subject scheme maps and their usage

Subject scheme maps can be used to define controlled values and subject definitions. The controlled values can be bound to attributes, as well as element and attribute pairs. The subject definitions can contain metadata and provide links to more detailed information; they can be used to classify content and provide semantics that can be used in taxonomies and ontologies.

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

5.2.1 Subject scheme maps

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

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

Subject definitions are classifications and sub-classifications that compose a tree. Subject definitions provide semantics that can be used in conjunction with taxonomies and ontologies. In conjunction with the classification domain, subject definitions can be used for retrieval and traversal of the content at run time when used with information viewing applications that provide such functionality.

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

5.2.2 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:

- A <titlealt> element with a @title-role of navigation (e.g. 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 in regard to subject scheme maps:

- 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.
- Authoring tools *MAY* support accessing and displaying the content of the subject definition resource in order to provide users with a detailed explanation of the subject.
- Authoring 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 expandable links as part of a user interface that implements a progressive disclosure strategy.

5.2.3 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>
```

5.2.4 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 behavior is expected of processors in regard to subject scheme maps:

- 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.

Processors *SHOULD* apply the following algorithm when they 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.

Example: A hierarchy of controlled values and conditional processing

The following code sample shows 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 flag "novice-therapist" since it is a type of therapist.

5.2.5 Extending subject schemes

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

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

the current subject-scheme map; the result is equivalent to specifying all of the values in a single subject scheme map.

5.2.6 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.

5.2.7 Classification maps

A classification map is a DITA map in which the classification domain has been made available.

The classification domain provides elements that enable map authors to indicate information about the subject matter of DITA topics. The subjects are defined in subjectScheme maps, and the map authors reference the subjects using the `@keyref` attribute.

5.2.8 Examples of subject scheme maps

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

5.2.8.1 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 will 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.

att="platform" val="linux"	att="platform" val="redhat"	How platform="redhat" is evaluated	How platform="linux" is evaluated
Unspecified	action="exclude"	Excluded, because all "redhat" content is excluded	If the default for @platform values is "include", this is included. If the default for @platform values is "exclude", this is excluded.
	action="include"	Included.	Included, because all "redhat" content is included, and general 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.

5.2.8.2 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>
```

5.2.8.3 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.

5.2.8.4 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>
```

5.3 Map cascading

5.3.1 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:

- `@rev`
- `@props` and any attribute specialized from `@props` (including those integrated by default in OASIS shells: `@audience`, `@deliveryTarget`, `@platform`, `@product`, `@otherprops`)

- `@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)

Related reference

topicmeta (270)

Topic metadata is metadata that applies to a topic based on its context in a map.

5.3.1.1 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 for the `@merge` attribute. 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.

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)".

The predefined values for the @cascade attribute *MUST* precede any implementation-specific tokens, for example, cascade="merge appToken:audience".

Example: The @cascade attribute in use

Consider the following code examples:

Figure 5: Map A

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

Figure 6: Map B

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

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:

Figure 7: Map C

```
<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>
```

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.

5.3.1.2 Processing cascading attributes in a map

Certain rules apply to processors when they process 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 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 5.3.3 Map-to-map cascading behaviors (54) for more details.
8. Repeat steps 1 (51) to 4 (52) 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.

For example, in the case of `<topicref toc="yes">`, applications must stop at item 2 (51) 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 5 (52) below when evaluating the `@toc` attribute.

5.3.2 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 `<titlealt>` or `<keytext>`, 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
<foreign>	Add to the topic	No, unless specialized for a purpose that cascades	No stated purpose, until the element is specified
<keytext>	Not added to the topic	No	No stated purpose
<keywords>	Add to the topic	No	No stated purpose
<metadata>	Add to the topic	Yes	Specify metadata for the entire map
<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
<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.	No	Provide a description of the map
<source>	Add to the topic	No	Specify a source for the map
<titlealt>	Add to the topic before its <titlealt> elements	No	Specify an alternative title for the map

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?
<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="2020"/>
            <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 HTML5, for example, each HTML5 file contains the metadata information.

Related reference

topicmeta (270)
Topic metadata is metadata that applies to a topic based on its context in a map.

5.3.3 Map-to-map cascading behaviors

When 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.

5.3.3.1 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:

- @rev
- @props and any attribute specialized from @props (including those integrated by default in OASIS shells: @audience, @deliveryTarget, @platform, @product, @otherprops)
- @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 7.6.1 The xml:lang attribute (136). 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 5.3.3.3 Cascading of roles from map to map (56) 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.

5.3.3.2 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 5.3.2 Reconciling topic and map metadata elements (52).

Note It is possible that a specialization might define metadata that is intended to 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:

Figure 8: test-2.ditamap

```
<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 9: b.ditamap

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

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>
  ```

5.3.3.3 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.

Because the <mapref> element 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 can 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.

5.4 Chunking

Content often needs to be delivered in a different granularity than it is authored. The @chunk attribute enables map authors to specify that multiple source documents are combined into a single document for delivery, or that a single source document is split into multiple documents for delivery.

5.4.1 About the @chunk attribute

The @chunk attribute is designed to handle cases where the best organization for authoring DITA topics is not equivalent to the best organization for publishing those topics.

The @chunk attribute is composed of a single token without any white space. DITA defines two tokens for the @chunk attribute: combine and split. Other tokens can be defined by applications, but support for those tokens will vary.

chunk="combine"
: The "combine" token in @chunk is intended for cases where a publishing process normally results in a single output artifact for each source XML document. When some or all of those source XML

documents are better presented as a single output artifact, setting `chunk="combine"` instructs a processor to combine the referenced source documents for rendering purposes.

chunk="split"
The "split" token in `@chunk` is intended for cases where a publishing process normally results in a single output artifact for each single source XML document, regardless of how many DITA topics exist within each source document. When a source XML document containing many topics is better rendered as multiple output artifacts, setting `chunk="split"` instructs a processor to split each topic from the referenced source document into its own document for rendering purposes.

The `@chunk` attribute describes how a processor can split or combine source DITA documents into alternate organizational schemes for rendering purposes. This means that the `@chunk` attribute is only relevant when the organization of source DITA documents has an effect on organization of published documents.

The `@chunk` attribute does not cascade.

The following rules apply to all values of the `@chunk` attribute:

- When the source document organization has no effect on published output, such as when producing a single PDF or EPUB, processors *MAY* ignore the `@chunk` attribute.
- When the `@chunk` attribute results in more or fewer documents based on the `combine` or `split` tokens, the hierarchy of topics within the resulting map and topic organization *SHOULD* match the hierarchy in the original topics and maps.
- When the `@chunk` attribute results in more or fewer documents, processors *MAY* create their own naming schemes for those reorganized documents.
- `@chunk` attribute values apply to DITA topic documents referenced from a map. Processors *MAY* apply equivalent processing to non-DITA documents.

5.4.2 Processing chunk="combine"

Setting `chunk="combine"` instructs a processor to combine the reference source documents for rendering purposes. A single result document is generated.

Comment by Kristen J Eberlein on 25 May 2019

Don't these need to be normative statements?

The following rules apply:

- When specified on the root element of a map, all source DITA documents referenced by the map are treated as one DITA document.
- When specified on a branch of a map, all source DITA documents referenced within that branch are treated as one DITA document.

  Note This is true regardless of whether the element that specifies `@chunk` refers to a topic or specifies a heading. In cases such as `<topicgroup>` where a grouping element specifies `chunk="combine"`, the equivalent DITA document would be a single DITA document with a root element grouping peer topics.
- When specified on a reference to a map, all source DITA documents within the scope of the referenced map are treated as one DITA document.
- Once `chunk="combine"` is specified on a map, branch, or map reference, all source DITA documents grouped by that reference are treated as a single resource. Any additional `@chunk` attributes on elements within the hierarchy are ignored.

5.4.3 Processing chunk="split"

Setting `chunk="split"` instructs a processor to split each topic from the referenced source document into its own document for rendering purposes. Multiple result documents are generated.

Comment by Kristen J Eberlein on 25 May 2019

Don't these need to be normative statements?

The following rules apply:

- When specified on a `<topicref>` element that refers to a source DITA document, it indicates that all topics within the referenced document should be rendered as individual documents.
- When specified on an element such as `<topicgroup>` that does not refer to a topic or result in a published topic, the attribute has no meaning.
- When specified on the root element of a map, `chunk="split"` sets a default operation for all source DITA documents in the map (outside the context of relationship tables). The default `split` value is used except where a `combine` value is encountered, in which case `combine` takes over for that entire branch.

5.4.4 Using the @chunk attribute for other purposes

Applications can define additional tokens for use in the `@chunk` attribute. These tokens are necessarily implementation dependent and might not be supported by other applications.

5.4.5 Examples of the @chunk attribute

These examples illustrate the processing expectations for various scenarios that involve the `@chunk` attribute. The processing examples use either before and after sample markup or expanded syntax that shows the equivalent markup without the `@chunk` attribute.

Note Examples use sample files with modified file names to help illustrate equivalent before and after resolution of `@chunk` attributes. However, there is no requirement for implementations processing `@chunk` to generate files, as long as the rendered result is split or combined as described. If generating files, actual file names are implementation dependent.

5.4.5.1 Example: Using @chunk to combine all documents into one

Where a publishing system typically would render each topic document as an independent result document, the `@chunk` attribute can be used to render all content as a single document.

Consider the following source documents, with root map `input.ditamap`:

Figure 10: Input map without chunking

```
input.ditamap:
<map>
  <title>Lesson plan</title>
  <topicref href="background.dita">
    <!-- more background topics -->
  </topicref>
  <topicref href="goals.dita">
    <!-- more goal topics -->
  </topicref>
  <!-- more topics -->
</map>

background.dita:
<topic id="background">
  <title>Prerequisite concepts</title>
```

```
  <shortdesc>This information is necessary before starting</shortdesc>
  <body><!-- ...background content... --></body>
</topic>

goals.dita:
<topic id="goals">
  <title>Lesson gals</title>
  <shortdesc>After you complete the lesson, ...</shortdesc>
  <body><!-- ...goal content... --></body>
</topic>
```

For many systems or output formats, each document in the map is rendered as an independent document. For example, rendering this map as HTML5 might result in `background.html` and `goals.html` (along with other HTML5 files). In such cases, if output requirements demand only a single result document, adding `chunk="combine"` to the root map element instructs a processor to render one document that combines all topics.

Figure 11: Root map with chunking specified

```
input.ditamap:
<map chunk="combine">
  <title>Lesson plan</title>
  <topicref href="background.dita">
    <!-- more background topics -->
  </topicref>
  <topicref href="goals.dita">
    <!-- more goal topics -->
  </topicref>
  <!-- more topics -->
</map>
```

The result of evaluating this `@chunk` attribute is equivalent to the following map and topic document; content from all topics within the map is combined into a single result, with a topic order and topic nesting structure that match the original map hierarchy.

Figure 12: Equivalent source content

```
input.ditamap:
<map>
  <title>Lesson plan</title>
  <topicref href="input.dita"/>
</map>

input.dita:
<dita>
  <!-- original content of background.dita -->
  <topic id="background">
    <title>Prerequisite concepts</title>
    <shortdesc>This information is necessary before starting</shortdesc>
    <body><!-- ...background content... --></body>
    <!-- more background topics -->
  </topic>
  <!-- original content of goals.dita -->
  <topic id="goals">
    <title>Lesson gals</title>
    <shortdesc>After you complete the lesson, ...</shortdesc>
    <body><!-- ...goal content... --></body>
    <!-- more goal topics -->
  </topic>
  <!-- more topics -->
</dita>
```

5.4.5.2 Example: Using @chunk to render a single document from one branch

Where a publishing system typically would render each topic document as an independent result document, the `@chunk` attribute can be used to render individual branches of a map as single documents.

Consider the following source documents, with root map `input.ditamap`:

Figure 13: Input map without chunking

```
input.ditamap:
<map>
  <title>Lesson plan</title>
  <topicref href="goals.dita">
    <!-- more goal topics -->
  </topicref>
  <topicref href="firstLesson.dita">
    <!-- more tasks in the first lesson -->
  </topicref>
  <topicref href="nextLesson.dita">
    <!-- more tasks in the next lesson -->
  </topicref>
  <!-- More branches -->
</map>

firstLesson.dita:
<task id="firstLesson">
  <title>Starting to work with scissors</title>
  <shortdesc>This lesson will teach ...</shortdesc>
  <taskbody><!-- ... --></taskbody>
</task>

nextLesson.dita:
<task id="nextLesson">
  <title>Advanced cutting</title>
  <shortdesc>This lesson will introduce complicated shapes...</shortdesc>
  <taskbody><!-- ... --></taskbody>
</task>
```

For many systems or output formats, each document in the map is rendered as an independent document. For example, rendering this map as HTML5 might result in `goals.html`, `firstLesson.html`, and `nextLesson.html`, while child documents within each branch would each result in their own HTML files.

When output requirements demand that portions of the map be combined into a single document, adding `chunk="combine"` to a branch of the map instructs a processor to render one document that combines all topics in that branch. This is particularly useful when the topics need to be rendered independently for other contexts, or when the way topics are contributed makes creating a single source document impossible.

In the following sample, the original map is updated with `@chunk` attributes to indicate that each lesson branch is rendered as a single result document; topics in the first branch with `goals.dita` are not affected as a result of the `@chunk` attribute.

Figure 14: Map with chunking specified for one branch

```
input.ditamap:
<map>
  <title>Lesson plan</title>
  <topicref href="goals.dita">
    <!-- more goal topics -->
  </topicref>
  <topicref href="firstLesson.dita" chunk="combine">
    <!-- more tasks in the first lesson -->
  </topicref>
  <topicref href="nextLesson.dita" chunk="combine">
    <!-- more tasks in the next lesson -->
```

```
  </topicref>
  <!-- More branches -->
</map>
```

The result of evaluating this `@chunk` attribute is equivalent to the following map and topic documents. Content from each branch where `@chunk` attribute is set is combined into a single result document, with an order and topic nesting structure that matches the original map hierarchy. Content from outside of those branches remains unchanged.

Figure 15: Equivalent source content

```
input.ditamap:
<map>
  <title>Lesson plan</title>
  <topicref href="goals.dita">
    <!-- more goal topics -->
  </topicref>
  <topicref href="firstLesson.dita"/>
  <topicref href="nextLesson.dita"/>
  <!-- More branches -->
</map>

firstLesson.dita:
<task id="firstLesson">
  <title>Starting to work with scissors</title>
  <shortdesc>This lesson will teach ...</shortdesc>
  <taskbody><!-- ... --></taskbody>
  <!-- more tasks in the first lesson -->
</task>

nextLesson.dita:
<task id="nextLesson">
  <title>Advanced cutting</title>
  <shortdesc>This lesson will introduce complicated shapes...</shortdesc>
  <taskbody><!-- ... --></taskbody>
  <!-- more tasks in the next lesson -->
</task>
```

5.4.5.3 Example: Using @chunk to combine a group of topics

The `@chunk` attribute can be used on grouping elements to combine multiple source documents into one result document.

Assume the following map `input.ditamap`, where `@chunk` is used on both `<topicgroup>` and `<topichead>`.

Figure 16: Input map

```
<map>
  <title>Groups are combined</title>
  <topicgroup chunk="combine">
    <topicref href="ingroup1.dita"/>
    <topicref href="ingroup2.dita"/>
  </topicgroup>
  <topichead chunk="combine">
    <topicmeta>
      <navtitle>Heading for a branch</navtitle>
    </topicmeta>
    <topicref href="inhead1.dita"/>
    <topicref href="inhead2.dita"/>
  </topichead>
</map>
```

The result of evaluating the `@chunk` attribute on `<topicgroup>` is equivalent to a single DITA document with the content of both `ingroup1.dita` and `ingroup2.dita`.

The `@chunk` attribute on `<topichead>` also results in a single DITA document. In many applications, a `<topichead>` is equivalent to a single title-only topic; in that case, the chunked result is equivalent to a root topic with the title "Heading for a branch", containing as child topics the content of both `inhead1.dita` and `inhead2.dita`. If `<topichead>` is ignorable in the current processing context, the chunked result would be equivalent to processing `<topicgroup>` (a single DITA document with the content of both `inhead1.dita` and `inhead2.dita`).

Figure 17: Equivalent source content

```
<map>
  <title>Groups are combined</title>
  <topicref href="chunkgroup-1.dita"/>
  <topicref href="chunkgroup-2.dita"/>
</map>

chunkgroup-1.dita
<dita>
  <!-- content of ingroup1.dita -->
  <!-- content of ingroup2.dita -->
</dita>

chunkgroup-2.dita
<dita>
  <topic id="head">
    <title>Heading for a branch</title>
    <!-- content of inhead1.dita -->
    <!-- content of inhead2.dita -->
  </topic>
</dita>
```

5.4.5.4 Example: Using @chunk to combine nested documents

Special attention is necessary when combining a nested map hierarchy that includes documents with their own nested topics.

Consider the following source map `input.ditamap`:

Figure 18: Input map without chunking

```
input.ditamap:
<map chunk="combine">
  <title>Generation example</title>
  <topicref href="ancestor.dita">
    <topicref href="middle.dita">
      <topicref href="child.dita"/>
    </topicref>
  </topicref>
</map>
```

In this case, the `@chunk` attribute instructs a processor to treat the three topics as a single combined DITA document, while preserving the original map hierarchy. Now consider the following three topic documents, each of which includes nested or peer topics:

Figure 19: Source documents with nested structures

```
ancestor.dita:
<dita>
  <topic id="ancestor-first">
    <title>First major topic in ancestor composite doc</title>
    <!-- ...topic content... -->
  </topic>
  <!-- more topics in ancestor composite doc -->
  <topic id="ancestor-last">
    <title>Last major topic in ancestor composite doc</title>
    <!-- ...topic content... -->
```

```
    <topic id="ancestor-last-child">
      <title>Child of last major topic in ancestor composite doc</title>
      <!-- ...topic content... -->
    </topic>
  </topic>
</dita>

middle.dita:
<topic id="middle-root">
  <title>Root topic in middle doc</title>
  <body><!-- ... --></body>
  <topic id="middle-child">
    <title>Child of root topic in middle doc</title>
    <!-- body content, maybe more children of middle topic's root -->
  </topic>
</topic>

child.dita:
<topic id="child">
  <title>Small child topic</title>
  <!-- small child topic content -->
</topic>
```

When `chunk="combine"` is evaluated, the three source documents are combined into one. Both the ancestor and middle documents have child topics that need to be taken into account.

- `ancestor.dita` has a root `<dita>` element, so content from each nested topic reference is located after any nested topics within the final child of the `<dita>` element.
- `middle.dita` does not have `<dita>` but does have a nested topic, so content from any nested topic references is located after that nested topic.

Figure 20: Equivalent source content

```
input.ditamap:
<map>
  <title>Generation example</title>
  <topicref href="input.dita"/>
</map>

input.dita:
<dita>
  <topic id="ancestor-first">
    <title>First major topic in ancestor composite doc</title>
    <!-- ...topic content... -->
  </topic>
  <!-- more topics in ancestor composite doc -->
  <topic id="ancestor-last">
    <title>Last major topic in ancestor composite doc</title>
    <!-- ...topic content... -->
    <topic id="ancestor-last-child">
      <title>Child of last major topic in ancestor composite doc</title>
      <!-- ...topic content... -->
    </topic>
    <!-- content of middle.dita combined here -->
    <topic id="middle-root">
      <title>Root topic in middle doc</title>
      <body><!-- ... --></body>
      <topic id="middle-child">
        <title>Child of root topic in middle doc</title>
        <!-- body content, maybe more children of middle topic's root -->
      </topic>
      <!-- content of child.dita combined here -->
      <topic id="child">
        <title>Small child topic</title>
        <!-- small child topic content -->
      </topic>
    </topic>
  </topic>
</dita>
```

5.4.5.5 Example: Using @chunk to split documents

When topics are most easily created or generated in a single DITA document, `chunk="split"` will instruct processors to render them individually when possible.

Splitting a single document in the map

Consider the following example, where a map includes generated topics used to document message numbers from an application:

Figure 21: Source map and topics

```
<map>
  <title>Message guide for WidgetAnalyzer</title>
  <topicref href="about.dita">
    <topicref href="messages-install.dita"/>
    <topicref href="messages-run.dita"/>
    <topicref href="messages-other.dita"/>
  </topicref>
</map>

about.dita:
<topic id="about">
  <title>About this guide</title>
  <shortdesc>Warnings or errors will appear if...<shortdesc>
</topic>

messages-install.dita:
<dita>
  <topic id="INS001">
    <title>INS001: Installation failure</title>
    <!-- explanation and recovery... -->
  </topic>
  <!-- more install messages... -->
</dita>

messages-run.dita:
<dita>
  <topic id="RUN001">
    <title>RUN001: Failed to initialize</title>
    <!-- explanation and recovery... -->
  </topic>
  <!-- hundreds of messages... -->
  <topic id="RUN999">
    <title>RUN999: Out of memory</title>
    <!-- explanation and recovery... -->
  </topic>
</dita>

messages-other.dita:
<topic id="othermsg">
  <title>Other messages</title>
  <shortdesc>You could also encounter ...</shortdesc>
  <topic id="OTHER001">
    <title>OTHER001: Analyzer is tired</title>
    <!-- explanation and recovery... -->
  </topic>
  <topic id="OTHER002">
    <title>OTHER002: Analyzer needs to be updated</title>
    <!-- explanation and recovery... -->
  </topic>
</topic>
```

In a normal build to HTML5, this map might result in four result documents `about.html`, `messages-install.html`, `messages-run.html`, and `messages-other.html`. With hundreds of messages in

`messages-run.dita`, it might be better in some situations to render one result document for each message topic in the document. This can be done by setting `chunk="split"` on the topic reference.

Figure 22: Splitting all topics in one document

```
<map>
  <title>Message guide for WidgetAnalyzer</title>
  <topicref href="about.dita">
    <topicref href="messages-install.dita"/>
    <topicref href="messages-run.dita" chunk="split"/>
    <topicref href="messages-other.dita"/>
  </topicref>
</map>
```

The result of evaluating `@chunk` in this case is equivalent to the following map and topics. While `messages-run.dita` now is split into hundreds of topics, other topics in the map are unaffected.

Figure 23: Equivalent source content

```
<map>
  <title>Message guide for WidgetAnalyzer</title>
  <topicref href="about.dita">
    <topicref href="messages-install.dita"/>
    <topicref href="RUN001.dita"/>
    <!-- hundreds of messages... -->
    <topicref href="RUN999.dita"/>
    <topicref href="messages-other.dita"/>
  </topicref>
</map>

RUN001.dita:
<topic id="RUN001">
  <title>RUN001: Failed to initialize</title>
  <!-- explanation and recovery... -->
</topic>

RUN999.dita:
<topic id="RUN999">
  <title>RUN999: Out of memory</title>
  <!-- explanation and recovery... -->
</topic>
```

Note Because the `@chunk` attribute does not cascade, even if the reference to `messages-install.dita` had child topic references, they would be unaffected by the `chunk="split"` value in this example.

Splitting every document in the map

Similarly, because setting `chunk="split"` on the map element sets a default for the entire map, the following change to the original map would result in *every* referenced DITA document being split into one document per topic. The only source document not affected by this split is `about.dita`, because it only contained a single topic to begin with.

Figure 24: Splitting every topic in the map

```
<map chunk="split">
  <title>Message guide for WidgetAnalyzer</title>
  <topicref href="about.dita">
    <topicref href="messages-install.dita"/>
    <topicref href="messages-run.dita"/>
    <topicref href="messages-other.dita"/>
  </topicref>
</map>
```

Using `chunk="split"` on the map is equivalent to the following structure:

- `about.dita` is unchanged.
- `messages-install.dita` is split into one document per message (as in the previous example that split `messages-run.dita`).
- `messages-run.dita` is split exactly as in the previous example.
- `messages-other.dita` contains a root topic and two child topics, so it results in three documents. The hierarchy of those documents is preserved in the map.

Figure 25: Equivalent source content

```
<map>
  <title>Message guide for WidgetAnalyzer</title>
  <topicref href="about.dita">
    <topicref href="INS001.dita"/>
    <!-- more install messages... -->
    <topicref href="RUN001.dita"/>
    <!-- hundreds of messages... -->
    <topicref href="RUN999.dita"/>
    <topicref href="othermsg.dita">
      <topicref href="OTHER001.dita"/>
      <topicref href="OTHER002.dita"/>
    </topicref>
  </topicref>
</map>

INS001.dita:
<topic id="INS001">
  <title>INS001: Installation failure</title>
  <!-- explanation and recovery... -->
</topic>

RUN001.dita:
<topic id="RUN001">
  <title>RUN001: Failed to initialize</title>
  <!-- explanation and recovery... -->
</topic>

RUN999.dita:
<topic id="RUN999">
  <title>RUN999: Out of memory</title>
  <!-- explanation and recovery... -->
</topic>

othermsg.dita:
<topic id="othermsg">
  <title>Other messages</title>
  <shortdesc>You could also encounter ...</shortdesc>
</topic>

OTHER001.dita:
<topic id="OTHER001">
  <title>OTHER001: Analyzer is tired</title>
  <!-- explanation and recovery... -->
</topic>

OTHER002.dita:
<topic id="OTHER002">
  <title>OTHER002: Analyzer needs to be updated</title>
  <!-- explanation and recovery... -->
</topic>
```

5.4.5.6 Example: Using @chunk to split nested documents

Special attention is necessary when evaluating the map hierarchy that results from splitting a documents with their own nested topics.

Consider the following source map `input.ditamap`:

Figure 26: Input map without chunking

```
input.ditamap:
<map chunk="split">
  <title>Generation example</title>
  <topicref href="ancestor.dita">
    <topicref href="middle.dita">
      <topicref href="child.dita"/>
    </topicref>
  </topicref>
</map>
```

In this case, the `@chunk` attribute instructs a processor to render every topic in each of the three documents as its own document, while preserving any hierarchy from those documents. Now consider the following three topic documents, each of which includes nested or peer topics:

Figure 27: Source documents with nested structures

```
ancestor.dita:
<dita>
  <topic id="ancestor-first">
    <title>First major topic in ancestor composite doc</title>
    <!-- ...topic content... -->
  </topic>
  <!-- more topics in ancestor composite doc -->
  <topic id="ancestor-last">
    <title>Last major topic in ancestor composite doc</title>
    <!-- ...topic content... -->
    <topic id="ancestor-last-child">
      <title>Child of last major topic in ancestor composite doc</title>
      <!-- ...topic content... -->
    </topic>
  </topic>
</dita>

middle.dita:
<topic id="middle-root">
  <title>Root topic in middle doc</title>
  <body><!-- ... --></body>
  <topic id="middle-child">
    <title>Child of root topic in middle doc</title>
    <!-- body content -->
  </topic>
</topic>

child.dita:
<topic id="child">
  <title>Small child topic</title>
  <!-- small child topic content -->
</topic>
```

When `chunk="split"` is evaluated, both `ancestor.dita` and `middle.dita` are split and treated as multiple DITA topic documents. `child.dita` is only a single topic and has nothing to split.

- `ancestor.dita` has a root `<dita>` element, so it results in multiple peer topic references (or branches) in the map. Topic references nested within the original reference to `ancestor.dita` are now located within the reference to "ancestor-last" (the last topic child of the `<dita>` element).

- `middle.dita` has nested topics, so results in its own new hierarchy within the map. Content from the nested topic reference is now located within the reference to the root topic from `middle.dita`, but after any references to child topics.

Figure 28: Equivalent source content

```
input.ditamap:
<map chunk="split">
  <title>Generation example</title>
  <topicref href="ancestor-first.dita"/>
  <!-- more topics in ancestor composite doc -->
  <topicref href="ancestor-last.dita">
    <topicref href="ancestor-last-child.dita"/>
    <!-- middle.dita now located here, as final child of
         final topic child of <dita> in ancestor.dita -->
    <topicref href="middle-root.dita">
      <topicref href="middle-child.dita"/>
      <!-- child.dita now located here, as final topic
           child root topic in middle.dita ancestor.dita -->
      <topicref href="child.dita"/>
    </topicref>
  </topicref>
</map>
```

5.4.5.7 Example: When @chunk is ignored

The `@chunk` attribute is ignored in some cases, such as when `chunk="combine"` is already in effect or when `chunk="split"` is specified on a grouping element.

Ignoring @chunk when already combining topics

In the following example, evaluating `@chunk` results in one rendered document for each branch of the map. Any additional `@chunk` values within that branch are ignored (including `@chunk` values within any referenced maps).

Figure 29: Chunk within a combined branch

```
<map>
  <title>Ignoring chunking when already combined</title>

  <topicref href="branchOne.dita" chunk="combine">
    <!-- @chunk ignored for branchOneChild.dita -->
    <topicref href="branchOneChild.dita" chunk="split"/>
  </topicref>

  <topicref href="branchTwo.dita" chunk="combine">
    <!-- Any @chunk within submap.ditamap is ignored -->
    <topicref href="submap.ditamap" format="ditamap"/>
  </topicref>
```

Ignoring @chunk on a grouping element

In the following example, `chunk="split"` is specified on two grouping elements.

Figure 30: Chunk within a combined branch

```
<map>
  <title>Trying to "split" groups</title>
  <topicgroup chunk="split">
    <topicref href="ingroup1.dita">...</topicref>
    <topicref href="ingroup2.dita">...</topicref>
  </topicgroup>
  <topichead chunk="split">
    <topicmeta><navtitle>Heading for a branch</navtitle></topicmeta>
```

```
    <topicref href="inhead1.dita">...</topicref>
    <topicref href="inhead2.dita">...</topicref>
  </topichead>
</map>
```

- The `@chunk` attribute on the `<topicgroup>` is ignored; it does not cascade, and there is no referenced topic, so it has no effect.
- In some cases, an implementation might treat the `<topichead>` element as equivalent to a single title-only topic, while in other cases it might be ignored. In either case the `@chunk` value has no effect. If the `<topichead>` is treated as a title-only topic, it cannot be split further; if it is ignored for the current processing context, it is no different than the `<topicgroup>`.

5.4.5.8 Example: Combining topics within a split context

While `@chunk` attributes are ignored when a "combine" action is already in effect, it is possible to use `chunk="combine"` when `split` is otherwise in effect.

Assume the following map, where `chunk="split"` on the root element means that all topic documents within this map structure are split by default, but a branch within the map sets `chunk="combine"`.

Figure 31: Map with default "split" action, that also uses "combine"

```
<map chunk="split">
  <title>Split most, but not one branch</title>
  <topicref href="splitme.dita">...</topicref>
  <topicref href="exception.dita" chunk="combine">...</topicref>
  <topicref href="splitmetoo.dita">...</topicref>
</topicref>
```

Assume as well that no other `@chunk` attributes are specified in this map. The following points are true when `@chunk` is evaluated:

1. The document `splitme.dita` is treated as multiple split documents when it contains more than one topic. The same is true for any other document within that branch.
2. The second branch (beginning with `exception.dita`) is treated as a single DITA document, combining all topic documents within that branch.
3. The document `splitmetoo.dita` is treated as multiple split documents when it contains more than one topic. The same is true for any other document within that branch.

5.4.5.9 Example: Managing links when chunking

Link management with `@chunk` is often straightforward; in most cases where URI-based linking is ambiguous, using indirect links and `@keyref` will give the correct result.

Input topics for following examples

The following map and topics are used for all examples in this topic.

Figure 32: input.ditamap

```
<map>
  <title>Map with chunks and links</title>

  <keydef href="splitThis.dita" keys="splitThisKey"/>
  <keydef href="splitThis.dita#splitThisChild" keys="splitThisChildKey"/>

  <topicref href="splitThis.dita" chunk="split" keys="explicitSplitKey"/>
  <topicref href="combineThis.dita" keys="combineThisKey">
    <topicref href="combinedChild.dita" keys="combinedChildKey"/>
```

```
  </topicref>
</map>
```

Figure 33: Topics used by input.ditamap

```
splitThis.dita:
<topic id="splitThisRoot">
  <title>Root topic in split document</title>
  <!-- ... -->
  <topic id="splitThisChild">
    <title>Child topic in split document</title>
    <!-- ... -->
  </topic>
</topic>

combineThis.dita:
<topic id="combineThisRoot">
  <title>Root topic in combined document</title>
  <!-- ... -->
  <topic id="combineThisChild">
    <title>Child topic in combined document</title>
    <!-- ... -->
  </topic>
</topic>

combinedChild.dita:
<topic id="combinedChildRoot">
  <title>Topic in child document, combined with parent</title>
  <!-- ... -->
</topic>
```

Topics that are rendered only once when publishing

Assume that the map above is a root map or is used by another map does not otherwise render the three topic documents. In that case, the following is true:

- `splitThis.dita` is rendered as two documents. For this example, assume a processor creates two documents with names taken from the topic ID, so that topic becomes `splitThisRoot.dita` and `splitThisChild.dita`.
- The branch with `combineThis.dita` is rendered as one document together with the content of `combinedChild.dita`. For this example, assume a processor merges the child topic into the file `combineThis.dita`.
- All links using `href="splitThis.dita"`, `keyref="splitThisKey"`, or `keyref="explicitSplitKey"` will resolve to `splitThisRoot.dita` (the only rendered instance of that topic).
- All links using `href="splitThis.dita#splitThisChild"` or `keyref="splitThisChildKey"` will resolve to `splitThisChild.dita` (the only rendered instance of that topic).
- All links using `href="combinedChild.dita"` or `keyref="combinedChildKey"` will resolve to that topic within `combineThis.dita` (the only rendered instance of that topic).

Topics that are rendered twice when publishing

Now assume that the map above is reused in another context that also renders all three topic documents as originally authored. As a result, each of the three documents in this map (`splitThis.dita`, `combineThis.dita`, and `combinedChild.dita`) are rendered more than once.

When each of these documents is rendered twice, the following is true:

- The original source document `splitThis.dita` is rendered twice. Based on the map above, assume a processor creates two documents with names taken from the topic ID, so that topic

becomes `splitThisRoot.dita` and `splitThisChild.dita`. At the same time, `splitThis.dita` is rendered *in another context* as a single document, with a different name.

- Based on the map above, the branch that starts with the original source document `combineThis.dita` is rendered as one document combined with the content of `combinedChild.dita`. At the same time, those two documents are rendered in another context as individual documents. For this example, assume a processor generates the combined document using the generated name `combinThis-2.dita`, while the documents `combineThis.dita` and `combinedChild.dita` retain their names in their other context.
- All links in this map using the direct URI references `href="splitThis.dita"`, `href="splitThis.dita#splitThisChild"`, `href="combineThis.dita"`, or `href="combinedChild.dita"` are now ambiguous. They could go to the chunked instance from this map, or to the individual topics in the other context. Implementations will have to guess which topic to target: the split or combined instances from this map or versions in the alternate context from the root map.
- All links using indirect key-based references `keyref="splitThisKey"` or `keyref="splitThisChildKey"` are also ambiguous, because the key definitions are not associated explicitly with the chunked or not-chunked instance. If key scopes are used, applications might more reliably guess that the intended target is the split copy in this map, but this is not guaranteed.
- All links using `keyref="explicitSplitKey"`, `keyref="combinedThisKey"`, or `keyref="combinedChildKey"` are unambiguous; they can only resolve to the chunked instance from this submap, because they are defined directly within the chunk context.
- There is no way to unambiguously link to the child document that will result from splitting `splitThis.dita`. This is because it is only possible for the element using `@chunk` to associate a key definition with the first or root topic in the document. While other key definition elements can be used to associate keys with other topics in the same document, that can only be done outside of the navigation context that uses `@chunk`; as a result, a processor cannot guarantee whether the intended link target is the split topic from the `@chunk` context, or a use of the same topic in the second context. It is possible for an implementation to define its own way to resolve this ambiguity; however, if a situation requires both multiple instances of split topics and unambiguous cross-implementation links to those split topics, alternate reuse mechanisms need to be considered.

6 DITA addressing

DITA provides two addressing mechanisms. DITA addresses either are direct URI-based addresses, or they are indirect key-based addresses. Within DITA documents, individual elements are addressed by unique identifiers specified on the `@id` attribute. DITA defines two fragment-identifier syntaxes; one is the full fragment-identifier syntax, and the other is an abbreviated fragment-identifier syntax that can be used when addressing non-topic elements from within the same topic.

6.1 ID attribute

The `@id` attribute assigns an identifier to DITA elements so that the elements can be referenced.

The `@id` attribute is available for most elements. An element must have a valid value for the `@id` attribute before it can be referenced using a fragment identifier. The requirements for the `@id` attribute differ depending on whether it is used on a topic element, a map element, or an element within a topic or map.

All values for the `@id` attribute must be XML name tokens.

The `@id` attributes for topic and map elements are declared as XML attribute type ID; therefore, they must be unique with respect to other XML IDs within the XML document that contains the topic or map element. The `@id` attribute for most other elements within topics and maps are not declared to be XML IDs; this means that XML parsers do not require that the values of those attributes be unique. However, the DITA specification requires that all IDs be unique within the context of a topic. For this reason, tools might provide an additional layer of validation to flag violations of this rule.

Within documents that contain multiple topics, the values of the `@id` attribute for all non-topic elements that have the same nearest-ancestor-topic element need to be unique with respect to each other. The values of the `@id` attribute for non-topic elements can be the same as non-topic elements with different nearest-ancestor-topic elements. Therefore, within a single DITA document that contains more than one topic, the values of the `@id` attribute of the non-topic elements need only to be unique within each topic.

Within a map document, the values of the `@id` attributes for all elements *SHOULD* be unique. When two elements within a map have the same value for the `@id` attribute, processors *MUST* resolve references to that ID to the first element with the given ID value in document order.

Figure 34: Summary of requirements for the @id attribute

Element	XML attribute type for `@id`	Must be unique within	Required?
`<map>`	ID	document	No
`<topic>`	ID	document	Yes
sub-map (elements nested within a map)	NMTOKEN	document	Usually no, with some exceptions
sub-topic (elements nested within a topic)	NMTOKEN	individual topic	Usually no, with some exceptions

Note For all elements other than footnote (`<fn>`), the presence of a value for the `@id` attribute has no impact on processing. For `<fn>`, the presence or absence of a valid `@id` attribute affects how the element is processed. This is important for tools that automatically assign `@id` attributes to all elements.

6.2 DITA linking

DITA supports many different linking elements, but they all use the same set of attributes to describe relationships between content.

URI-based addressing

URI-based links are described by the following attributes.

@href
: The `@href` attribute specifies the URI of the resource that is being addressed.

@format
: The `@format` attribute identifies the format of the resource being addressed. For example, references to DITA topics are identified with `format="dita"`, whereas references to DITA maps use `format="ditamap"`. References to other types of content use other values for this attribute. By default, references to non-XML content use the extension of the URI in the `@href` attribute as the effective format.

@scope
: The `@scope` attribute describes the closeness of the relationship between the current document and the target resource. Resources in the same information unit are considered `"local"`; resources in the same system as the referencing content but not part of the same information unit are considered `"peer"`; and resources outside the system, such as Web pages, are considered `"external"`.

@type
: The `@type` attribute is used on cross-references to describe the target of the reference. Most commonly, the `@type` attribute names the element type being referenced when `format="dita"`.

These four attributes act as a unit, describing whatever link is established by the element that carries them.

The `@format` and `@scope` attributes are assigned default values based on the URI that is specified in the `@href` attribute. Thus they rarely need to be explicitly specified in most cases. However, they can be useful in many non-traditional linking scenarios or environments.

Indirect key-based addressing

DITA also supports indirect links and cross-references in which a DITA map assigns unique names, or keys, to the resources being referenced by the publication. This is done using `<topicref>` elements that specify the `@keys` attribute. Using the `@keyref` attribute, individual links, cross-references, and images then reference resources by their keys instead of their URIs. Links defined using `@keyref` thus allow context-specific linking behavior. That is, the links in a topic or map might resolve to one set of resources in one context, and a completely different set of resources in another, without the need for any modifications to the link markup.

When links are defined using `@keyref`, values for the four linking attributes described above are typically all specified (or given default values) on the key defining element.

6.3 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 the`<dita>` 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"`
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)

6.4 Indirect key-based addressing

DITA keys provide an alternative to direct addressing. The key reference mechanism provides a layer of indirection so that resources (for example, URIs, metadata, or variable text strings) can be defined at the DITA map level instead of locally in each topic.

For information about using keys to define and reference controlled values, see 5.2 Subject scheme maps and their usage (39).

Note The material in this section of the DITA specification is exceptionally complex; it is targeted at implementers who build processors and other rendering applications.

6.4.1 Core concepts for working with keys

The concepts described below are critical for a full understanding of keys and key processing.

The use of the phases "`<map>` element" or "`<topicref>` element" should be interpreted as "`<map>` element and any specialization of `<map>` element " or " `<topicref>` element or any specialization of `<topicref>` element."

Definitions related to keys

resource
For the purposes of keys and key resolution, one of the following:

- An object addressed by URI
- Metadata specified on a resource, such as a `@scope` or `@format` attribute
- Text or metadata located within a `<topicmeta>` element

key
A name for a resource. See 6.4.3 Using keys for addressing (80) for more information.

key definition
A `<topicref>` element that binds one or more key names to zero or more resources.

key reference
An attribute that references a key, such as `@keyref` or `@conkeyref`.

key space
A list of key definitions that are used to resolve key references.

effective key definition
The definition for a key within a key space that is used to resolve references to that key. A key might have multiple definitions within a key space, but only one of those definitions is effective.

key scope
A map or section of a map that defines its own key space and serves as the resolution context for its key references.

Key definitions

A key definition binds one or more keys to zero or more resources. Resources can be:

- Any URI-addressed resource that is referenced directly by the `@href` attribute or indirectly by the `@keyref` attribute on the key definition. References to the key are considered references to the URI-addressed resource.
- (If the key definition contains a child `<topicmeta>` element) The child elements of the `<topicmeta>` element. The content of those elements can be used to populate the content of elements that reference the key.

If a key definition does not contain a <topicmeta> element and does not refer to a resource by @href or @keyref, it is nonetheless a valid key definition. References to the key definition are considered resolvable, but no linking or content transclusion occurs.

Key scopes

All key definitions and key references exist within a key scope. If the @keyscope attribute is never specified within the map hierarchy, all keys exist within a single, default key scope.

Additional key scopes are created when the @keyscope attribute is used. The @keyscope attribute specifies a name or names for the scope. Within a map hierarchy, key scopes are bounded by the following:

- The root map.
- The root element of submaps when the root elements of the submaps specify the @keyscope attribute
- Any <topicref> elements that specify the @keyscope attribute

Key spaces

The key space associated with a key scope is used to resolve all key references that occur immediately within that scope. Key references in child scopes are resolved using the key spaces that are associated with those child scopes.

A key scope is associated with exactly one key space. That key space contains all key definitions that are located directly within the scope; it might also contain definitions that exist in other scopes. Specifically, the key space associated with a key scope is comprised of the following key definitions, in order of precedence:

1. All key definitions from the key space associated with the parent key scope, if any.
2. Key definitions within the scope-defining element, including those defined in directly-addressed, locally-scoped submaps, but excluding those defined in child scopes. (Keys defined in child scopes cannot be addressed without qualifiers.)
3. The key definitions from child scopes, with each key prepended by the child scope name followed by a period. If a child scope has multiple names, the keys in that scope are addressable from the parent scope using any of the scope names as a prefix.

Note Because of rules 1 and 3, the key space that is associated with a child scope includes the scope-qualified copies of its own keys that are inherited from the key space of the parent scope, as well as those from other "sibling" scopes.

Effective key definitions

A key space can contain many definitions for a given key, but only one definition is effective for the purpose of resolving key references.

When a key has a definition in the key space that is inherited from a parent scope, that definition is effective. Otherwise, a key definition is effective if it is first in a breadth-first traversal of the locally-scoped submaps beneath the scope-defining element. Put another way, a key definition is effective if it is the first definition for that key name in the shallowest map that contains that key definition. This allows higher-level map authors to override keys defined in referenced submaps.

Note A key definition that specifies more than one key name in its @keys attribute might be the effective definition for some of its keys but not for others.

Within a key scope, keys do not have to be defined before they are referenced. The key space is effective for the entire scope, so the order of key definitions and key references relative to one another is not significant. This has the following implications for processors:

- All key spaces for a root map must be determined before any key reference processing can be performed.
- Maps referenced solely by key reference have no bearing on key space contents.

For purposes of key definition precedence, the scope-qualified key definitions from a child scope are considered to occur at the location of the scope-defining element within the parent scope. See 6.4.11.5 Example: How key scopes affect key precedence (99) for more information.

6.4.2 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:

Figure 35: Root map

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

```
  <!-- ... -->
</map>
```

Figure 36: installation.ditamap

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

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

6.4.3 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"`.

6.4.4 Addressing keys across scopes

When referencing key definitions that are defined in a different key scope, key names might need to be qualified with key scope names.

A root map might contain any number of key scopes; relationships between key scopes are discussed using the following terms:

child scope
: A key scope that occurs directly within another key scope. For example, in the figure below, key scopes "A-1" and "A-2" are child scopes of key scope "A".

parent scope
A key scope that occurs one level above another key scope. For example, in the figure below, key scope "A" is a parent scope of key scopes "A-1" and "A-2".

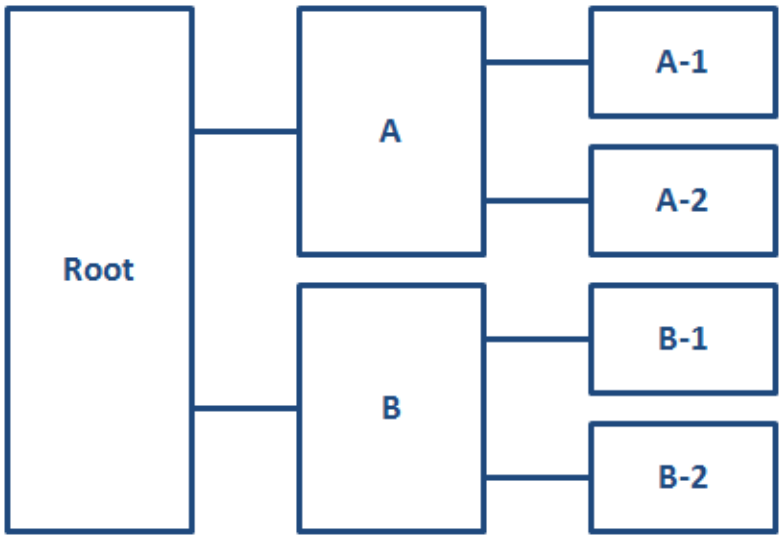
ancestor scope
A key scope that occurs any level above another key scope. For example, in the figure below, key scopes "A" and "Root" are both ancestor scopes of key scopes "A-1" and "A-2"

descendant scope
A key scope that occurs any level below another key scope. For example, in the figure below, key scopes "A", "A-1", and "A-2" are all descendant scopes of the implicit, root key scope

sibling scope
A key scope that shares a common parent with another key scope. For example, in the figure below, key scopes "A" and "B" are sibling scopes; they both are children of the implicit, root key scope.

key scope hierarchy
A key scope and all of its descendant scopes.

Figure 37: A key scope hierarchy



Keys that are defined in parent key scopes

The key space that is associated with a key scope also includes all key definitions from its parent key scope. If a key name is defined in both a key scope and its parent scope, the key definition in the parent scope takes precedence. This means that a key definition in a parent scope overrides all definitions for the same key name in all descendant scopes. This enables map authors to override the keys that are defined in submaps, regardless of whether the submaps define key scopes.

In certain complex cases, a scope-qualified key name (such as "scope.key") can override an unqualified key name from the parent scope. See 6.4.11.5 Example: How key scopes affect key precedence (99).

Keys that are defined in child key scopes

The key space associated with a key scope does not include the *unqualified* key definitions from the child scopes. However, it does include scope-qualified keys from the child scopes. This enables sibling key scopes to have different key definitions for the same key name.

A *scope-qualified key name* is a key name, prepended by one or more key scope names and separated by periods. For example, to reference a key "keyName" defined in a child scope named "keyScope", specify `keyref="keyScope.keyName"`.

If a key scope has multiple names, its keys can be addressed from its parent scope using any of the scope names. For example, if a key scope is defined with `keyscope="a b c"`, and it contains a key name of "product", that key can be referenced from the parent scope by `keyref="a.product"`, `keyref="b.product"`, or `keyref="c.product"`

Because a child scope contributes its scope-qualified keys to its parent scope, and that parent scope contributes *its* scope-qualified keys to *its* parent scope, it is possible to address the keys in any descendant scope by using the scope-qualified key name. For example, consider a key scope named "ancestorScope" that has a child scope named "parentScope" which in turn has a child scope named "childScope". The scope "childScope" defines a key named "keyName". To reference the key "keyName" from scope "ancestorScope", specify the scope-qualified key name: `keyref="parentScope.childScope.keyName"`.

Keys that are defined in sibling key scopes

Because a parent key scope contains scope-qualified keys from all of its child scopes, and a child scope inherits all of the key definitions (including scope-qualified keys) from its parent scope, it is possible for a child scope to reference its own scope-qualified keys, as well as those defined by its sibling scopes.

For example, consider two sibling scopes, "scope1" and "scope2". Each scope defines the key "productName". References to "productName" in each scope resolve to the local definition. However, since each scope inherits the scope-qualified keys that are available in their parent scope, either scope can reference "scope1.productName" and "scope2.productName" to refer to the scope-specific definitions for that key.

6.4.5 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.

Comment by Kristen J Eberlein on 04 July 2019

Should the following statement about what processors do "when a reference to a peer map cannot be resolved" contain RFC-2119 language?

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.

6.4.6 Processing key references

Key references can resolve as links, as text, or as both. Within a map, they also can be used to create or supplement information on a topic reference. This topic covers information that is common to all key processing, regardless of how the key is used.

Processing of undefined keys

If both `@keyref` and `@href` attributes are specified on an element, the `@href` value *MUST* be used as a fallback address when the key name is undefined. If both `@conkeyref` and `@conref` attributes are specified on an element, the `@conref` value *MUST* be used as a fallback address when the key name is undefined.

Determining effective attributes on the key-referencing element

The attributes that are common to the key-defining element and the key-referencing element, other than the `@keys`, `@processing-role`, and `@id` attributes, are combined as for content references, including the special processing for the `@xml:lang`, `@dir`, and `@translate` attributes.

Keys and conditional processing

The effective key definitions for a key space might be affected by conditional processing (filtering). Processors *SHOULD* perform conditional processing before determining the effective key definitions. However, processors might determine effective key definitions before filtering. Consequently, different processors might produce different effective bindings for the same map when there are key definitions that might be filtered out based on their filtering attributes.

Note In order to retain backwards compatibility with DITA 1.0 and 1.1, the specification does not mandate a processing order for different DITA features. This makes it technically possible to determine an effective key definition, resolve references to that key definition, and then filter out the definition. However, the preferred approach is to take conditional processing into account when resolving keys, so that key definitions which are excluded by processing are not used in resolving key references.

Reusing a topic in multiple key scopes

If a topic that contains key references is reused in multiple key scopes within a given root map such that its references resolve differently in each use context, processors *MUST* produce multiple copies of the source topic in resolved output for each distinct set of effective key definitions that are referenced by the topic.

In such cases, authors can use the `@copy-to` attribute to specify different source URIs for each reference to a topic.

Error conditions

If a referencing element contains a key reference with an undefined key, it is processed as if there were no key reference, and the value of the `@href` attribute is used as the reference. If the `@href` attribute is not specified, the element is not treated as a navigation link. If it is an error for the element to be empty, an implementation *MAY* give an error message; it also *MAY* recover from this error condition by leaving the key reference element empty.

6.4.7 Processing key references for navigation links and images

Keys can be used to create or redirect links and cross references. Keys also can be used to address resources such as images or videos. This topic explains how to evaluate key references on links and cross references to determine a link target.

When a key definition is bound to a resource that is addressed by the `@href` or `@keyref` attributes, and does not specify "none" for the `@linking` attribute, all references to that key definition become links to the bound resource. When a key definition is not bound to a resource or specifies "none" for the `@linking` attribute, references to that key definition do not become links.

When a key definition has no `@href` value and no `@keyref` value, references to that key will not result in a link, even if they do contain an `@href` attribute of their own. If the key definition also does not contain a `<topicmeta>` subelement, empty elements that refer to the key (such as `<link keyref="a"/>` or `<xref keyref="a" href="fallback.dita"/>`) are ignored.

The `<object>` element has additional key-referencing attributes (`@archivekeyrefs`, `@classidkeyref`, `@codebasekeyref`, and `@datakeyref`). Key names in these attributes are resolved using the same processing that is described for the normal `@keyref` attribute.

6.4.8 Processing key references on <topicref> elements

While `<topicref>` elements are used to define keys, they also can reference keys that are defined elsewhere. This topic explains how to evaluate key references on `<topicref>` elements and its specializations.

Determining the effective resource

For topic references that use the `@keyref` attribute, the effective resource bound to the `<topicref>` element is determined by resolving all intermediate key references. Each key reference is resolved either to a resource addressed directly by URI reference in an `@href` attribute, or to no resource. Processors *MAY* impose reasonable limits on the number of intermediate key references that they will resolve. Processors *SHOULD* support at least three levels of key references.

Note This rule applies to all topic references, including those that define keys. The effective bound resource for a key definition that uses the `@keyref` attribute cannot be determined until the key space has been constructed.

Combining metadata

Content from a key-defining element cascades to the key-referencing element following the rules for combining metadata between maps and other maps and between maps and topics.

The combined attributes and content cascade from one map to another or from a map to a topic, but this is controlled by existing rules for cascading, which are not affected by the use of key references.

If, in addition to the `@keys` attribute, a key definition specifies a `@keyref` attribute that can be resolved after the key resolution context for the key definition has been determined, the resources bound to the referenced key definition take precedence.

6.4.9 Processing key references to generate text or link text

Variable text can be specified by key definitions. Processors determine the effective text by retrieving the content of elements in a specific sequence.

Empty elements

Empty elements that specify a key reference might get their effective content from the referenced key definitions. For the purpose of determining variable text, *empty elements* are defined as elements that meet the following criteria:

- Have no text content, including white space
- Have no sub-elements
- Have no attributes that would be used as text content

Key definitions with child <topicmeta> elements

When an empty element references a key definition that has a child `<topicmeta>` element, content from that `<topicmeta>` element is used to determine the effective content of the referencing element. Effective content from the key definition becomes the element content, with the following exceptions:

- For empty `<image>` elements, the effective content is used as alternate text. This is equivalent to creating an `<alt>` sub-element to hold that content.
- For empty `<link>` elements, the effective content is used as link text. This is equivalent to creating a `<linktext>` sub-element to hold that content.
- For empty `<link>` and `<xref>` elements, a key definition can provide a short description in addition to the normal effective content. If the key definition includes `<shortdesc>` inside of `<topicmeta>`, the content of the `<shortdesc>` element also provides effective content for a `<desc>` sub-element.
- The `<longdescref>` and `<longquoteref>` elements are empty elements with no effective content. Key definitions do not set effective text for these elements.
- The `<param>` element does not have any effective content, so key definitions do not result in effective content for `<param>` elements.

Processing rules

Processors *MUST* resolve variable text that is defined using keys by using the following sequence:

1. Effective text content is taken from the `<keytext>` element.
2. Effective text content is taken from the `<titlealt>` element with `@title-role` set to "linking".
3. Effective text content is taken from the `<titlealt>` element with `@title-role` set to "navigation".
4. Effective text content is taken from the `<titlealt>` element with `@title-role` set to a processor-recognized value.
5. Effective text content is taken from the title of the referenced document, if available.
6. Effective text content is determined by the processor.

Generalization of effective content

When the effective content for a key reference element results in invalid elements, those elements *SHOULD* be generalized to produce a valid result.

For example, `<keytext>` in the key definition might use a domain specialization of `<keyword>` that is not valid in the key reference context, in which case the specialized element is generalized to `<keyword>`. If the generalized content is also not valid, a text equivalent is used instead. For example, `<keytext>` might include `<ph>` or a specialized `<ph>` in the key definition, but neither of those are valid as the effective content for a `<keyword>`. In that case, the text content of the `<ph>` is used.

6.4.10 Examples of keys

This section of the specification contains examples and scenarios. They illustrate a wide variety of ways that keys can be used.

6.4.10.1 Examples: Key definition

The <topicref> element, and any specialization of <topicref> that allows the @keys attribute, can be used to define keys.

In the following example, a <topicref> element is used to define a key; the <topicref> element also contributes to the navigation structure.

```
<map>
  <!--... -->
  <topicref keys="apple-definition" href="apple-gloss-en-US.dita" />
  <!--... -->
</map>
```

The presence of the @keys attribute does not affect how the <topicref> element is processed.

In the following example, a <keydef> element is used to define a key.

```
<map>
  <!--... -->
  <keydef keys="apple-definition" href="apple-gloss-en-US.dita"/>
  <!--... -->
</map>
```

Because the <keydef> element sets the default value of the @processing-role attribute to "resource-only", the key definition does not contribute to the map navigation structure; it only serves as a key definition for the key name "apple-definition".

6.4.10.2 Examples: Key definitions for variable text

Key definitions can be used to store variable text, such as product names and user-interface labels. Depending on the key definition, the rendered output might have a link to a related resource.

In the following example, a "product-name" key is defined. The key definition contains a child <keyword> element nested within a <keydef>element.

```
<map>
  <keydef keys="product-name">
    <topicmeta>
     <keywords>
       <keyword>Thing-O-Matic</keyword>
     </keywords>
    </topicmeta>
  </keydef>
</map>
```

A topic can reference the "product-name" key by using the following markup:

```
<topic id="topicid">
  <p><keyword keyref="product-name"/> is a product designed to ...</p>
</topic>
```

When processed, the output contains the text "Thing-O-Matic is a product designed to …".

In the following example, the key definition contains both a reference to a resource and variable text.

```
<map>
  <keydef keys="product-name" href="thing-o-matic.dita">
```

```
    <topicmeta>
     <keywords>
       <keyword>Thing-O-Matic</keyword>
     </keywords>
    </topicmeta>
  </keydef>
</map>
```

When processed using the key reference from the first example, the output contains the "Thing-O-Matic is a product designed to …" text. The phrase "Thing-O-Matic" also is a link to the `thing-o-matic.dita` topic.

6.4.10.3 Example: Duplicate key definitions within a single map

In this scenario, a DITA map contains duplicate key definitions. How a processor finds the effective key definition depends on document order and the effect of filtering applied to the key definitions.

In the following example, a map contains two definitions for the key "load-toner":

```
<map>
  <!--... -->
  <keydef keys="load-toner" href="model-1235-load-toner-proc.dita"/>
  <keydef keys="load-toner" href="model-4545-load-toner-proc.dita"
  />
  <!--... -->
</map>
```

In this example, only the first key definition (in document order) of the "load-toner" key is effective. All references to the key within the scope of the map resolve to the topic `model-1235-load-toner-proc.dita`.

In the following example, a map contains two definitions for the "file-chooser-dialog" key; each key definition specifies a different value for the `@platform` attribute.

```
<map>
  <!--... -->
  <keydef keys="file-chooser-dialog" href="file-chooser-osx.dita" platform="osx"/>
  <keydef keys="file-chooser-dialog" href="file-chooser-win7.dita" platform="windows7"/>
  <!--... -->
</map>
```

In this case, the effective key definition is determined not only by the order in which the definitions occur, but also by whether the active value of the platform condition is "osx" or "windows7". Both key definitions are *potentially* effective because they have distinct values for the conditional attribute. Note that if no active value is specified for the `@platform` attribute at processing time, then both of the key definitions are present and so the first one in document order is the effective definition.

If the DITAVAL settings are defined so that both "osx" and "windows7" values for the `@platform` attribute are excluded, then neither definition is effective and the key is undefined. That case can be avoided by specifying an unconditional key definition after any conditional key definitions, for example:

```
<map>
  <!--... -->
  <keydef keys="file-chooser-dialog" href="file-chooser-osx.dita" platform="osx"/>
  <keydef keys="file-chooser-dialog" href="file-chooser-win7.dita" platform="windows7"/>
  <keydef keys="file-chooser-dialog" href="file-chooser-generic.dita"/>
  <!--... -->
</map>
```

If the above map is processed with both "osx" and "windows7" values for the `@platform` attribute excluded, then the effective key definition for "file-chooser-dialog" is the `file-chooser-generic.dita` resource.

6.4.10.4 Example: Duplicate key definitions across multiple maps

In this scenario, the root map contains references to two submaps, each of which defines the same key. The effective key definition depends upon the document order of the direct URI references to the maps.

In the following example, a root map contains a key definition for the key "toner-specs" and references to two submaps.

```
<map>
  <keydef keys="toner-specs" href="toner-type-a-specs.dita"/>
  <mapref href="submap-01.ditamap"/>
  <mapref href="submap-02.ditamap"/>
</map>
```

The first submap, `submap-01.ditamap`, contains definitions for the keys "toner-specs" and "toner-handling":

```
<map>
  <keydef keys="toner-specs" href="toner-type-b-specs.dita"/>
  <keydef keys="toner-handling" href="toner-type-b-handling.dita"/>
</map>
```

The second submap, `submap-02.ditamap`, contains definitions for the keys "toner-specs", "toner-handling", and "toner-disposal":

```
<map>
  <keydef keys="toner-specs" href="toner-type-c-specs.dita"/>
  <keydef keys="toner-handling" href="toner-type-c-handling.dita"/>
  <keydef keys="toner-disposal" href="toner-type-c-disposal.dita"/>
</map>
```

For this example, the effective key definitions are listed in the following table.

Key	Bound resource
toner-specs	`toner-type-a-specs.dita`
toner-handling	`toner-type-b-handling.dita`
toner-disposal	`toner-type-c-disposal.dita`

The key definition for "toner-specs" in the root map is effective, because it is the first encountered in a breadth-first traversal of the root map. The key definition for "toner-handling" in `submap-01.ditamap` is effective, because submap-01 is included before submap-02 and so comes first in a breadth-first traversal of the submaps. The key definition for "toner-disposal" is effective because it is the only definition of the key.

6.4.10.5 Example: Key definition with key reference

When a key definition also specifies a key reference, the key reference also must be resolved in order to determine the effective resources that are bound to that key definition.

In the following example, a `<topicref>` element references the key "widget". The definition for "widget" in turn references the key "mainProduct".

```
<map>
  <topicref keyref="widget" id="example"/>
  <keydef keys="widget" href="widgetInfo.dita" scope="local" format="dita" rev="v1r2"
          keyref="mainProduct">
    <topicmeta><navtitle>Information about Widget</navtitle></topicmeta>
  </keydef>
  <keydef keys="mainProduct" href="http://example.com/productPage" scope="external"
format="html"
```

```
          product="prodCode" audience="sysadmin">
    <topicmeta><navtitle>Generic product page</navtitle></topicmeta>
  </keydef>
</map>
```

For this example, the key reference to "widget" pulls resources from that key definition, which in turn pulls resources from "mainProduct".

The resources from the key definitions are combined as follows:

- The metadata resources from "mainProduct" are combined with the resources already specified on the "widget" key definition, resulting in the addition of `@product` and `@audience` values.
- The navigation title on the "widget" key definition overrides those on the "mainProduct" key definition.
- The `@href`, `@scope`, and `@format` attributes on the "mainProduct" key definition override those on "widget".

Thus after key references are resolved, the original `<topicref>` element is equivalent to the following:

```
<topicref id="example"
          href="http://example.com/productPage" scope="external" format="html"
          rev="v1r2"
          product="prodCode" audience="sysadmin">
   <topicmeta><navtitle>Information about Widget</navtitle></topicmeta>
</topicref>
```

6.4.10.6 Example: Link redirection

This scenario outlines how different authors can redirect links to a common topic by using key definitions. This could apply to `<xref>`, `<link>`, or any elements (such as `<keyword>` or `<term>`) that become navigation links.

A company wants to use a common DITA topic for information about recycling: `recycling.dita`. However, the topic contains a cross-reference to a topic that needs to be unique for each product line; each such topic contains product-specific URLs.

1. The editing team creates a `recycling.dita` topic that includes a cross-reference to the product-specific topic. The cross reference is implemented using a key reference:

   ```
   <xref keyref="product-recycling-info" href="generic-recycling-info.dita"/>
   ```

   The value of the `@href` attribute provides a fallback in the event that a product team forgets to include a key definition for "product-recycling-info".
2. Each product documentation group creates a unique key definition for "product-recycling-info". Each group authors the key definition in a DITA map, for example:

   ```
   <map>
     <!-- ... -->
     <keydef keys="product-recycling-info" href="acme-server-recycling.dita"/>
     <!-- ... -->
   </map>
   ```

   Each team can use the `recycling.dita` topic, and the cross reference in the topic resolves differently for each team.

3. A year later, there is an acquisition. The newly-acquired team wants to reuse Acme's common material, but it needs to direct its users to an external Web site that lists the URLs, rather than a topic in the product documentation. Their key definition looks like the following:

```
<topicref  keys="product-recycling-info"
                href="http://acme.example.com/server/recycling"
                scope="external" format="html"/>
```

   When newly-acquired team uses the `recycling.dita` topic, it resolves to the external Web site; however for all other teams, the cross reference in the topic continues to resolves to their product-specific topic.
4. A new product team is formed, and the team forgets to include a key definition for "product-recycling-info" in one of their root maps. Because the cross reference in the `recycling.dita` topic contains a value for the `@href` attribute, the link falls back to `generic-recycling-info.dita`, thus avoiding a broken cross reference in the output.

6.4.10.7 Example: Link modification or removal

This scenario outlines how different authors can effectively remove or modify a `<link>` element in a shared topic.

A company wants to use a shared topic for information about customer support. For most products, the shared topic includes a link to a topic about extended warranties. But a small number of products do not offer extended warranties.

1. Team one creates the shared topic: `customer-support.dita`. The topic contains the following mark-up:

```
 <related-links>
  <link keyref="extended-warranties" href="common/extended-warranties.dita"/>
 </related-links>
```

2. The teams that need the link to the topic about extended warranties can reference the `customer-support.dita` topic in their DITA maps. When processed, the related link in the topic resolves to the `common/extended-warranties.dita` topic.
3. The teams that do not want the related link to the topic about extended warranties can include a key definition in their DITA map that does not include an `@href` attribute, for example:

```
<map>
  <!-- ... -->
  <keydef keys="extended-warranties"/>
  <!-- ... -->
</map>
```

   When processed, the related link in the topic is not rendered.
4. Yet another team wants to simply have a paragraph about extended warranties printed. They define the key definition for "extended-warranties" as follows:

```
<map>
  <!-- ... -->
  <keydef keys="extended-warranties"/>
    <topicmeta>
      <keytext>This product does not offer extended warranties.</keytext>
    </topicmeta>
  <!-- ... -->
</map>
```

When this team renders their content, there is no hyperlink in the output, just the text "This product does not offer extended warranties" statement.

6.4.10.8 Example: Links from <term> or <keyword> elements

The `@keyref` attribute enables authors to specify that references to keywords or terms in a DITA topic can be rendered as a link to an associated resource.

In this scenario, a company with well-developed glossary wants to ensure that instances of a term that is defined in the glossary always include a link to the glossary topic.

1. An information architect adds values for the `@keys` attribute to all the of the `<topicref>` elements that are in the DITA map for the glossary, for example:

```
<map>
  <title>Company-wide glossary</title>
  <topicref keys="term-1" href="term-1.dita"/>
  <topicref keys="term-2" href="term-2.dita"/>
  <topicref keys="term-3" href="term-3.dita"/>
  <topicref keys="term-4" href="term-4.dita"/>
</map>
```

2. When authors refer to a term in a topic, they use the following mark-up:

```
<term keyref="term-1"/>
```

   When the `<term>` element is rendered, the content is provided by the `<title>` element of the glossary topic. The `<term>` element also is rendered as a link to the glossary topic.

6.4.10.9 Example: conref redirection

The `@conkeyref` attribute enables authors to share DITA topics that reuse content. It also enables map authors to specify different key definitions for common keys.

In this scenario, Acme produces content for a product that is also resold through a business partner. When the DITA content is published for the partner, several items must be different, including the following:

- Product names
- Standard notes that contain admonitions

Simply using the `@conref` attribute would not be possible for teams that use a component content management system where every DITA topic is addressed by a globally-unique identifier (GUID).

1. Authors reference the reusable content in their topics by using the `@conkeyref` attribute, for example:

```
<task id="reusable-product-content">
  <title><keyword conkeyref="reuse/product-name"/> prerequisites</title>
  <taskbody>
    <prereq><note conkeyref="reuse/warning-1"/></prereq>
    <!-- ... -->
  </taskbody>
</task>
```

2. Authors create two different topics; one topic contains elements appropriate for Acme, and the other topic contains elements appropriate for the partner. Note that each reuse topic must use the same element types (or compatible specializations) and values for the `@id` attribute. For example, the following reuse file is appropriate for use by Acme:

```
<topic id="acme-reuse">
  <title>Reuse topic for Acme</title>
```

```
  <body>
    <note id="warning-1">Admonitions for Acme</note>
    <p><keyword id="product-name">Acme product name</keyword></p>
    <!-- ... -->
  </body>
</topic>
```

The following reuse file is appropriate for use by the OEM partner:

```
<topic id="oem-reuse">
  <title>Reuse topic for OEM partner</title>
  <body>
    <note id="warning-1">Admonitions for partner</note>
    <p><keyword id="product-name">OEM product name</keyword></p>
    <!-- ... -->
  </body>
</topic>
```

3. The two versions of the DITA maps each contain different key definitions for the key name "reuse". (This associates a key with the topic that contains the appropriate reusable elements.) For example:

Figure 38: DITA map for Acme

```
<map>
  <!-- ... -->
  <keydef keys="reuse" href="acme-reuse.dita"/>
  <!-- ... -->
</map>
```

Figure 39: DITA map for OEM partner

```
<map>
  <!-- ... -->
  <keydef keys="reuse" href="oem-reuse.dita"/>
  <!-- ... -->
</map>
```

When each of the DITA maps is published, the elements that are referenced by `@conkeyref` will use the reuse topic that is referenced by the `<keydef>` element in the map. The product names and warnings will be different in the output.

6.4.10.10 Example: Keys and collaboration

Keys enable authors to collaborate and work with evolving content with a minimum of time spent reworking topic references.

In this scenario, authors collaborate on a publication that includes content for a product that is in the early stages of development. The company documentation is highly-structured and uses the same organization for all publications: "Introduction," "Example," and "Reference."

1. Author one creates a submap for the new product information. She knows the structure that the final content will have, but she does not want to create empty topics for information that is not yet available. She decides to initially author what content is available in a single topic. When more content is available, she'll create additional topics. Her DITA map looks like the following:

```
<map>
  <title>New product content</title>
  <topicref keys="1-overview 1-intro 1-example 1-reference" href="1-overview.dita"/>
</map>
```

2. Author two knows that he needs to add a `<topicref>` to the "Example" topic that will eventually be authored by author one. He references the not-yet-authored topic by key reference:

```
<topicref keyref="1-example"/>
```

   His topic reference initially resolves to the `1-overview.dita` topic.
3. Author one finally gets the information that she was waiting on. She creates additional topics and modifies her DITA map as follows:

```
<map>
  <title>New product content</title>
  <topicref keys="1-overview" href="1-overview.dita">
    <topicref keys="1-intro" href="1-intro.dita"/>
    <topicref keys="1-example" href="1-example.dita"/>
    <topicref keys="1-reference" href="1-reference.dita"/>
  </topicref>
</map>
```

   Without needing to make any changes to the content, author two's topic reference now resolves to the `1-example.dita` topic.

6.4.11 Examples of scoped keys

This section of the specification contains examples and scenarios. They illustrate how scoped keys can be used.

6.4.11.1 Example: Scoped key definitions for variable text

Scoped key definitions can be used for variable text. This enables you to use the same DITA topic multiple times in a DITA map, and in each instance the variable text can resolve differently.

The Acme Tractor Company produces two models of tractor: X and Y. Their product manual contains sets of instructions for each model; until now, the maintenance procedures have been different for each model. Now, the product manual needs to add instructions for changing the oil, and the procedure is identical for both model X and model Y. While most maintenance procedures are different for each model, the instructions for changing the oil are identical for both model X and model Y. The company policies call for including the specific model number in each topic, so a generic topic that could be used for both models is not permitted. Scoped keys can solve this problem.

1. The authoring team creates the new `changing-the-oil.dita`. The new topic uses the following markup to reference the product model:

```
<keyword keyref="model"/>
```

2. The information architect examines the root map for the manual, and decides how to define key scopes. Originally, the map looked like the following:

```
<map>
  <!-- Model X: Maintenance procedures -->
  <topicref href="model-x-procedures.dita">
    <topicref href="model-x/replacing-a-tire.dita"/>
    <topicref href="model-x/adding-fluid.dita"/>
  </topicref>

<!-- Model Y: Maintenance procedures -->
  <topicref href="model-y-procedures.dita">
    <topicref href="model-y/replacing-a-tire.dita"/>
    <topicref href="model-y/adding-fluid.dita"/>
  </topicref>
</map>
```

3. The information architect wraps each set of procedures in a `<topicgroup>` element and sets the `@keyscope` attribute.

```
<map>
  <!-- Model X: Maintenance procedures -->
  <topicgroup keyscope="model-x">
    <topicref href="model-x-procedures.dita">
      <topicref href="model-x/replacing-a-tire.dita"/>
      <topicref href="model-x/adding-fluid.dita"/>
    </topicref>
  </topicgroup>

<!-- Model Y: Maintenance procedures -->
  <topicgroup keyscope="model-y">
    <topicref href="model-y-procedures.dita">
      <topicref href="model-y/replacing-a-tire.dita"/>
      <topicref href="model-y/adding-fluid.dita"/>
    </topicref>
  </topicgroup>
</map>
```

This defines the key scopes for each set of procedures.

4. The information architect then adds key definitions to each set of procedures, as well as a reference to the `changing-the-oil.dita` topic.

```
<map>
  <!-- Model X: Maintenance procedures -->
  <topicgroup keyscope="model-x">
    <keydef keys="model">
      <topicmeta>
        <keytext>X</keytext>
      </topicmeta>
    </keydef>
    <topicref href="model-x-procedures.dita">
      <topicref href="model-x/replacing-a-tire.dita"/>
      <topicref href="model-x/adding-fluid.dita"/>
      <topicref href="common/changing-the-oil.dita"/>
    </topicref>
  </topicgroup>

<!-- Model Y: Maintenance procedures -->
  <topicgroup keyscope="model-y">
    <keydef keys="model">
      <topicmeta>
        <keytext>Y</keytext>
      </topicmeta>
    </keydef>
    <topicref href="model-y-procedures.dita">
      <topicref href="model-y/replacing-a-tire.dita"/>
      <topicref href="model-y/adding-fluid.dita"/>
      <topicref href="common/changing-the-oil.dita"/>
    </topicref>
  </topicgroup>
</map>
```

When the DITA map is processed, the `changing-the-oil.dita` topic is rendered twice. The model variable is rendered differently in each instance, using the text as specified in the scoped key definition. Without key scopes, the first key definition would win, and "model "X" would be used in all topics.

6.4.11.2 Example: References to scoped keys

You can address scoped keys from outside the key scope in which the keys are defined.

```
<map xml:lang="en">
  <title>Examples of scoped key references</title>
```

```
  <!-- Key scope #1 -->
  <topicgroup keyscope="scope-1">
    <keydef keys="key-1" href="topic-1.dita"/>
    <topicref keyref="key-1"/>
    <topicref keyref="scope-1.key-1"/>
    <topicref keyref="scope-2.key-1"/>
  </topicgroup>

  <!-- Key scope #2 -->
  <topicgroup keyscope="scope-2">
    <keydef keys="key-1" href="topic-2.dita"/>
    <topicref keyref="key-1"/>
    <topicref keyref="scope-1.key-1"/>
    <topicref keyref="scope-2.key-1" />
  </topicgroup>

  <topicref keyref="key-1" />
  <topicref keyref="scope-1.key-1" />
  <topicref keyref="scope-2.key-1" />

</map>
```

For this example, the effective key definitions are listed in the following tables.

Figure 40: Effective key definitions for scope-1

Key reference	Resource
key-1	topic-1.dita
scope-1.key-1	topic-1.dita
scope-2.key-1	topic-2.dita

Figure 41: Effective key definitions for scope-2

Key reference	Resource
key-1	topic-2.dita
scope-1.key-1	topic-1.dita
scope-2.key-1	topic-2.dita

Figure 42: Effective key definitions for the key scope associated with the root map

Key reference	Resource
key-1	Undefined
scope-1.key-1	topic-1.dita
scope-2.key-1	topic-2.dita

6.4.11.3 Example: Key definitions in nested key scopes

In this scenario, the root map contains nested key scopes, each of which contain duplicate key definitions. The effective key definition depends on key-scope precedence rules.

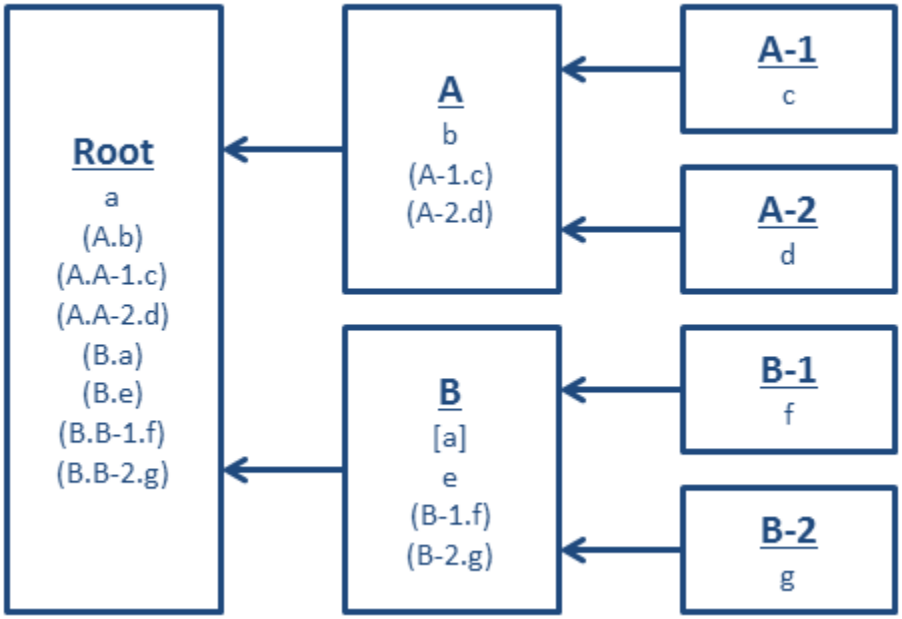
Consider the following DITA map:

```
<map>
  <title>Root map</title>
  <!-- Root scope -->
  <keydef keys="a" href="topic-1.dita"/>
```

```
  <!-- Key scope A -->
  <topicgroup keyscope="A">
    <keydef keys="b" href="topic-2.dita"/>

    <!-- Key scope A-1 -->
    <topicgroup keyscope="A-1">
      <keydef keys="c" href="topic-3.dita"/>
    </topicgroup>

    <!-- Key scope A-2 -->
    <topicgroup keyscope="A-2">
      <keydef keys="d" href="topic-4.dita"/>
    </topicgroup>
  </topicgroup>

  <!-- Key scope B -->
  <topicgroup keyscope="B">
    <keydef keys="a" href="topic-5.dita"/>
    <keydef keys="e" href="topic-6.dita"/>

    <!-- Key scope B-1 -->
    <topicgroup keyscope="B-1">
      <keydef keys="f" href="topic-7.dita"/>
    </topicgroup>

    <!-- Key scope B-2 -->
    <topicgroup keyscope="B-2">
      <keydef keys="g" href="topic-8.dita"/>
    </topicgroup>
  </topicgroup>
</map>
```

The key scopes in this map form a tree structure.

Figure 43: Graphical representation of the key scopes



Each box in the diagram represents a key scope; the name of the key scope is indicated in bold with upper-case letters. Below the name of the key scope, the key definitions that are present in the scope are listed. Different typographic conventions are used to indicate where the key definition occurs:

No styling
The key definition occurs in the immediate key scope and is not overridden by a key definition in a parent scope. For example, key "a" in the root map.

Parentheses
The key definition occurs in a child scope. For example, keys "A-1.c" and "A-2.d" in key scope A.

Brackets
The key definition occurs in the immediate key scope, but it is overridden by a key definition in an ancestor scope. For example, key "a" in key scope B.

Arrows point from child to parent scopes.

Assume that each key scope contains numerous key references. The following tables demonstrate how key references resolve in key scopes A-2 and B. The first column shows the value used in key references; the second column shows the resource to which the key resolves.

Table 2: Key scope A-2

Key reference	Resource to which the key resolves
a	"a", defined in the root map: `topic-1.dita`
d	"d", as defined in the immediate key scope: `topic-4.dita`
A-2.d	"d", as defined in the immediate key scope: `topic-4.dita`
c	Undefined
A-1.c	"A-1.c", as defined in key scope A-1. This key name is available because it exists in the parent scope, key scope A. The key name resolves to `topic-3.dita`
A.A-1.c	"A-1.c", as defined in key scope A-1. This key name is available because it exists in the root key scope. The key name resolves to `topic-3.dita`

Table 3: Key scope B

Key reference	Resource to which the key resolves
e	"e", defined in the immediate key scope: `topic-6.dita`
a	"a", as defined in the *root key scope*. (While a key definition for "a" exists in the immediate key scope, it is overridden by the key definition that occurs in the parent key scope.) The key name resolves to `topic-1.dita`
B.a	"a", as defined in the *immediate key scope*. Because the key reference uses the scope-qualified names, it resolves to the key "a" in scope B. The key name resolves to `topic-5.dita`
g	Undefined. The key "g" is defined only in key scope B-2, so no unqualified key named "g" is defined in scope B.
B-2.g	"g", as defined in key scope B-2: `topic-8.dita`.

6.4.11.4 Example: Key scopes and omnibus publications

Key scopes enable you to create omnibus publications that include multiple submaps that define the same key names for common items, such as product names or common topic clusters.

In this scenario, a training organization wants to produce a deliverable that includes all of their training course materials. Each course manual uses common keys for standard parts of the course materials, including "prerequisites," "overview", "assessment", and "summary.

An information architect creates a root map that contains the following markup:

```
<map xml:lang="en">
  <title>Training courses</title>
  <mapref href="course-1.ditamap"/>
  <mapref href="course-2.ditamap"/>
  <mapref href="course-3.ditamap"/>
  <topicref href="omnibus-summary.dita"/>
</map>
```

Each of the submaps contain `<topicref>` elements that refer to resources using the `@keyref` attribute. Each submap uses common keys for standard parts of the course materials, including "prerequisites," "overview", "assessment", and "summary", and their key definitions bind the key names to course-specific resources. For example:

```
<map xml:lang="en">
  <title>Training course #1</title>
  <mapref href="course-1/key-definitions.ditamap"/>
  <topicref keyref="prerequisites"/>
  <topicref keyref="overview"/>
  <topicref keyref="assessment"/>
  <topicref keyref="summary"/>
</map>
```

Without using key scopes, the effective key definitions for the common keys resolve to those found in `course-1.ditamap`. This is not the desired outcome. By adding key scopes to the submaps, however, the information architect can ensure that the key references in the submaps resolve to the course-specific key definitions.

```
<map xml:lang="en">
  <title>Training courses</title>
  <mapref href="course-1.ditamap" keyscope="course-1"/>
  <mapref href="course-2.ditamap" keyscope="course-2"/>
  <mapref href="course-3.ditamap" keyscope="course-3"/>
  <topicref href="omnibus-summary.dita"/>
</map>
```

The information architect does **not** set `keys="summary"` on the `<topicref>` element in the root map. Doing so would mean that all key references to "summary" in the submaps would resolve to `omnibus-summary.dita`, rather than the course-specific summary topics. This is because key definitions located in parent scopes override those located in child scopes.

6.4.11.5 Example: How key scopes affect key precedence

For purposes of key definition precedence, the scope-qualified key definitions from a child scope are considered to occur at the location of the scope-defining element within the parent scope.

Within a single key scope, key precedence is determined by which key definition comes first in the map, or by the depth of the submap that defines the key. This was true for all key definitions prior to DITA 1.3, because all key definitions were implicitly in the same key scope. Scope-qualified key names differ in that precedence is determined by the location where the key scope is defined.

This distinction is particularly important when key names or key scope names contain periods. While avoiding periods within these names will avoid this sort of issue, such names are legal so processors will need to handle them properly.

The following root map contains one submap and one key definition. The submap defines a key named "sample".

Figure 44: Root map

```
<map>
  <!-- The following mapref defines the key scope "scopeName" -->
  <mapref href="submap.ditamap" keyscope="scopeName"/>

  <!-- The following keydef defines the key "scopeName.sample" -->
  <keydef keys="scopeName.sample" href="losing-key.dita"/>

  <!-- Other content, key definitions, etc. -->
</map>
```

Figure 45: Submap

```
<map>
  <keydef keys="sample" href="winning-key.dita"/>
  <!-- Other content, key definitions, etc. -->
</map>
```

When determining precedence, all keys from the key scope "scopeName" occur at the location of the scope-defining element—in this case, the `<mapref>` element in the root map. Because the `<mapref>` comes first in the root map, the scope-qualified key name "scopeName.sample" that is pulled from `submap.ditamap` occurs before the definition of "scopeName.sample" in the root map. This means that in the context of the root map, the effective definition of "scopeName.sample" is the scope-qualified key definition that references `winning-key.dita`.

The following illustration shows a root map and several submaps. Each submap defines a new key scope, and each map defines a key. In order to aid understanding, this sample does not use valid DITA markup; instead, it shows the content of submaps inline where they are referenced.

Figure 46: Complex map with multiple submaps and scopes

```
<map>   <!-- Start of the root map -->

  <mapref href="submapA.ditamap" keyscope="scopeA">
    <!-- Contents of submapA.ditamap begin here -->
    <mapref href="submapB.ditamap" keyscope="scopeB">
      <!-- Contents of submapB.ditamap: define key MYKEY -->
      <keydef keys="MYKEY" href="example-ONE.dita"/>
    </mapref>
    <keydef keys="scopeB.MYKEY" href="example-TWO.dita"/>
    <!-- END contents of submapA.ditamap -->
  </mapref>

  <mapref href="submapC.ditamap" keyscope="scopeA.scopeB">
    <!-- Contents of submapC.ditamap begin here -->
    <keydef keys="MYKEY" href="example-THREE.dita"/>
  </mapref>

  <keydef keys="scopeA.scopeB.MYKEY" href="example-FOUR.dita"/>
</map>
```

The sample map shows four key definitions. From the context of the root scope, all have key names of "scopeA.scopeB.MYKEY".

1. `submapB.ditamap` defines the key "MYKEY". The key scope "scopeB" is defined on the `<mapref>` to `submapB.ditamap`, so from the context of `submapA.ditamap`, the scope-qualified key name is "scopeB.MYKEY". The key scope "scopeA" is defined on the `<mapref>` to `submapA.ditamap`, so from the context of the root map, the scope-qualified key name is "scopeA.scopeB.MYKEY".
2. `submapA.ditamap` defines the key "scopeB.MYKEY". The key scope "scopeA" is defined on the `<mapref>` to `submapA.ditamap`, so from the context of the root map, the scope-qualified key name is "scopeA.scopeB.MYKEY".
3. `submapC.ditamap` defines the key "MYKEY". The key scope "scopeA.scopeB" is defined on the `<mapref>` to `submapC.ditamap`, so from the context of the root map, the scope-qualified key name is "scopeA.scopeB.MYKEY".
4. Finally, the root map defines the key "scopeA.scopeB.MYKEY".

Because scope-qualified key definitions are considered to occur at the location of the scope-defining element, the effective key definition is the one from `submapB.ditamap` (the definition that references `example-ONE.dita`).

7 DITA processing

DITA processing is affected by a number of factors, including attributes that indicate the set of vocabulary and constraint modules on which a DITA document depends; navigation; linking; content reuse (using direct or indirect addressing); conditional processing; branch filtering; chunking; and more. In addition, translation of DITA content is expedited through the use of the `@dir`, `@translate`, and `@xml:lang` attributes.

7.1 Navigation

DITA includes markup that processors can use to generate reader navigation to or across DITA topics. Such navigation behaviors include table of contents (TOCs) and indexes.

7.1.1 Table of contents

Processors can generate a table of contents (TOC) based on the hierarchy of the elements in a DITA map. By default, each `<topicref>` element in a map represents a node in the TOC. These topic references define a navigation tree.

When a map contains a topic reference to a map (often called a map reference), processors integrate the navigation tree of the referenced map with the navigation tree of the referencing map at the point of reference. In this way, a deliverable can be compiled from multiple DITA maps.

> **Note** If a `<topicref>` element that references a map contains child `<topicref>` elements, the processing behavior regarding the child `<topicref>` elements is undefined.

The effective navigation title is used for the value of the TOC node. A TOC node is generated for every `<topicref>` element that references a topic or specifies a navigation title, except in the following cases:

- The `@processing-role` attribute that is specified on the `<topicref>` element or an ancestor element is set to "resource-only".
- Conditional processing is used to filter out the node or an ancestor node.
- There is no information from which a TOC entry can be constructed; there is no referenced resource or navigation title.
- The node is a `<topicgroup>` element, even if it specifies a navigation title.

To suppress a `<topicref>` element from appearing in the TOC, set the `@toc` attribute to "no". The value of the `@toc` attribute cascades to child `<topicref>` elements, so if `@toc` is set to "no" on a particular `<topicref>`, all children of the `<topicref>` element are also excluded from the TOC. If a child `<topicref>` overrides the cascading operation by specifying `toc="yes"`, then the node that specifies `toc="yes"` appears in the TOC (minus the intermediate nodes that set `@toc` to "no").

7.2 Indexes

Processors can generate an index from the content of indexing elements.

Comment by Kristen J Eberlein on 09 August 2019

Comment by Eliot Kimber:

I think we need a general discussion of index processing and rendering that covers:

- Generation of page numbers or other locators (addresses "typical" question)
- General rules or expectations for merging entries to produce the "effective index markup"

- General guidance for reporting conditions such as see-also references to non-existent entries and missing @end for @start specified in maps.

The current writeup reflects a lot of unstated assumptions about how index processing does or should work, obviously reflecting how IBM's tools worked (and work today). These assumptions need to be surfaced.

Comment by Kristen J Eberlein on 12 August 2019

We need to clearly state our assumptions, but I think we need to be careful about specifying expected processing too precisely:

- We have no idea what output format people are transforming their DITA to; we cannot assume that it is print-based. And we have no idea what sort of output formats might emerge after DITA 2.0 is released!
- Not all DITA processors support indexing.

It would be good to be upfront about the fact that many details about index generation will be implementation specific.

7.2.1 Index elements

The content of `<indexterm>` elements provides the text for the entries in a generated index. `<indexterm>` elements can be nested to create secondary and tertiary index entries.

The following elements contain information that processors use to generate indexes.

<indexterm>
: Instructs a processor to generate an index entry. The `@start` and `@end` attributes on the `<indexterm>` element can specify index ranges.

<index-see>
: Instructs a processor to generate a *see reference*. See references direct a reader to the preferred term.

<index-see-also>
: Instructs a processor to generate a *see also reference*. See also references direct a reader to an alternate index entry for additional information.

How the index elements are combined, the location of `<indexterm>` elements, and the hierarchy of the DITA maps all effect how the index elements are processed and the resulting generated index entries.

7.2.2 Location of <indexterm> elements

`<indexterm>` elements can occur in topic prologs, anywhere else in DITA topics, and in DITA maps.

The location of an `<indexterm>` element determines where the `<indexterm>` element points to, and where an `<indexterm>` element points to determines the locators that are rendered in a generated index.

Topic prologs
: An `<indexterm>` element that is located in a topic prolog is a point reference to the title of the topic. If an `<indexterm>` element has an `@end` attribute, it is a point reference to the end of the topic.

Anywhere else in a DITA topic
: An `<indexterm>` element that is located in a topic (and not the topic prolog) is a point reference to the location where the `<indexterm>` element occurs.

DITA maps

An <indexterm> element that is contained by a <topicref> element is a point reference to the title of the topic. If an <indexterm> element has an @end attribute, it is a point reference to the end of the topic. If the topic reference is not bound to a resource, the <indexterm> element has no stated purpose.

7.2.3 Index locators

Typically, an <indexterm> element instructs a processor to generate an index entry with a locator.

The nesting of <indexterm> elements and the presence of <index-see> elements determines whether locators are rendered in the generated index entries:

- An <indexterm> element that does not contain child <indexterm> elements (or an <index-see> element) contributes a locator to the generated index entry.
- An <indexterm> element that contains child <indexterm> elements contributes to the hierarchy of the multilevel index entry that is generated. Only the final nested <indexterm> element contributes a locator to the generated index entry.
- If an <indexterm> element also contains one or more <index-see> elements, no locator is included in the generated index entry.
- If an <indexterm> element also contains one or more <index-see-also> elements, the <indexterm> element contributes a locator to the generated index entry, and <index-see-also> element provides only a redirection.

7.2.4 Index redirection

The <index-see> and <index-see-also> elements enable redirection to other index entries within the generated index.

The <index-see> element contains text for an index entry that the reader should use *instead of* the current one, whereas the <index-see-also> element contains text for an index entry that the reader should use *in addition to* the current one.

Generated index entries should not contain both locators and redirections. Therefore, it is an error if the following conditions occur:

An <indexterm> contains <index-see>, and the publication contains other <indexterm> with matching content

An <indexterm> element contains an <index-see> element, and the publication contains one or more <indexterm> elements with matching textual content.

For example, topics referenced by the master map include the following markup:

```
<!-- Topic A -->
<indexterm>memory stick
  <index-see>USB drive</index-see>
</indexterm>

<!-- Topic B -->
<indexterm>memory stick</indexterm>
```

An <indexterm> contains <index-see> and <index-see-also>

An <indexterm> element contains both an <index-see> element and an <index-see-also> element.

For example, a topic contains the following <indexterm> element:

```
<indexterm>
  memory stick
    <index-see>USB drive</index-see>
    <index-see-also>flash stick</index-see-also>
</indexterm>
```

A processor *MAY* give an error message when it encounters the following error conditions:

- An <indexterm> element contains an <index-see> element, and the publication contains one or more <indexterm> elements with matching textual content.
- Both <index-see> and <index-see-also> elements within the same <indexterm> element.

Processors *MAY* recover from these error conditions by treating the <index-see> element as an <index-see-also> element.

Comment by Joyce Lam on 08 August 2019

Context = Normative statement about "An <indexterm> contains <index-see> and <index-see-also>"

Because we emphasize "MAY", it also means that it "may not". I would like to ask "Who are we leaving the decision up to?"

Is this instead a question of "SHOULD"?

Comment by Eliot Kimber on 09 August 2019

I think SHOULD is better too

Comment by Kristen J Eberlein on 12 August 2019

I think we cannot make this a stronger statement here than *MAY*:

- There are no interoperability issues involved.
- Processors might choose to handle this error condition different. Basically, we (the TC) are suggesting one possible approach that seems well considered.

Reminder for people to review the following material when considering any spec statement that contains RFC-2119 terminology:

- 1.2 Terminology (7)
- Guidelines to Writing Conformance Clauses for OASIS Specifications

7.2.5 Index ranges

Authors can use the @start and @end attributes on <indexterm> elements to index extended discussions. Processors generate index entries that range over several locators.

The start of an index range is indicated by an <indexterm> with a @start attribute. This is called a *start element*.

The end of a range is indicated by whichever of the following occurs first:

- An <indexterm> element with an @end attribute with a value that matches the @start attribute on the <indexterm> element that begins the range. This is called an *end element*.

- The applicable scope boundary.

The applicable scope boundary depends on the location of the start element:

Topic body
End of the topic body.

Comment by Eliot Kimber on 09 August 2019

I'm not sure I agree with this rule. If I start a range in one topic body and end it in another topic body and the two topics are presented in a sequence such that the second topic follows the first I would expect the range to span from the first topic to the second.

So the rule is at least dependent on the presentation context--in a continuous presentation there's no reason to impose boundaries, but in a chunked presentation there might be.

I think there's an unstated assumption that indexes are rendered for paged media where page numbers make sense but that's not a necessary condition--for example, I could have chunked HTML output that includes knowledge of the page numbers the content is rendered on in some other paged rendering (i.e., the printed version of municipal code, where the page numbers are captured in an element-to-page-number mapping and the HTML rendering of the same content, where the page numbers are included as literal or meta data in the HTML pages).

Comment by Kristen J Eberlein on 12 August 2019

Comment updated on 19 August 2019:

FYI, these rules have been in the DITA spec since DITA 1.1, when the `@start` and `@end` attributes were added to `<indexterm>`.

Hopefully all references to "page numbers" have been changed to the more neutral term "locator".

Eliot, I think that for your use case, the appropriate markup would be to index in the map. Per the processing outlined in this draft, the range should begin at the start of one topic and end at the end of the other topic.

For example, given the following map, the generated index entry should be "test, x- y", where x = the title of `index-page-references.dita` and y = the end of `merging-index-elements.dita`.

```
<map>
    <title>Indexes</title>
    <topicref href="indexes.dita">
        <topicref href="index-elements.dita"/>
        <topicref href="location-of-indexterm-elements.dita"/>
        <topicref href="index-page-references.dita">
            <topicmeta>
                <keywords>
                    <indexterm start="test">test</indexterm>
                </keywords>
            </topicmeta>
        </topicref>
        <topicref href="index-redirection.dita"/>
        <topicref href="index-ranges.dita"/>
        <topicref href="merging-index-elements.dita">
            <topicmeta>
                <keywords>
                    <indexterm end="test">test</indexterm>
                </keywords>
            </topicmeta>
        </topicref>
        <topicref href="examples-indexing.dita">
            <topicref href="example-of-nested-indexterm-elements.dita"/>
```

```
            <topicref href="example-of-an-index-range.dita"/>
        </topicref>
    </topicref>
</map>
```

FYI, this is the markup for the map for the indexing content, with the start and end elements added.

Topic prolog
End of the topic that contains the start element, including any child topics

DITA map

Whichever of the following occurs first:

- End of the topic that the start element references,, including any child topics
- End of the DITA map

Processors that support index ranges *SHOULD* do the following:

- Match `@start` and `@end` attributes by a character-by-character comparison with all characters significant and no case folding. occurring
- Ignore `@start` and `@end` attributes if they occur on an `<indexterm>` element that has child `<indexterm>` elements.
- When index ranges with the same identifier overlap, the effective range is determined by matching the earliest start element from the set of overlapping ranges with the latest end element from the set of overlapping ranges.
- Handle an end-of-range `<indexterm>` element that is nested within one or more `<indexterm>` elements. The end-of-range `<indexterm>` element should have no content of its own; if it contains content, that content is ignored.
- Ignore unmatched end-of-range `<indexterm>` elements.

The `@start` and `@end` attributes are defined as CDATA. However, we recommend that authors do not include whitespace characters (spaces or tabs) or control characters in values for these attributes.

7.2.6 Index sorting

The combination of an `<indexterm>` and `<sort-as>` element specifies a sort phrase under which an index entry is sorted.

This gives an author the flexibility to sort an index entry in an index differently from how its text normally would be sorted. The common use for this is to disregard insignificant leading text, such as punctuation or words like "the" or "a". For example, the author might want `<data>` to be sorted under the letter D rather than the left angle bracket (<). An author might want to include such an entry under both the punctuation heading and the letter D, in which case there can be two index entry directives differentiated only by the sort order.

Comment by Kristen J Eberlein on 18 July 2019

Comment from Dawn Stevens during the stage three review of proposal #253, "Indexing changes":

“So .. an author might want to sort <data> under both D and <. If that was the case, they would need two separate indexterm elements, each with its own <sort-as>?”

I think we need to clarify the above paragraph, but I think it will be better done as part of a targeted review of all spec content about indexing, rather than as part of the review for this proposal.

Certain languages have special sort order needs. For example, Japanese index entries might be written partially or wholly in kanji, but need to be sorted in phonetic order according to its hiragana/katakana rendition. There is no reliable automated way to map written to phonetic text: for kanji text, there can be multiple phonetic possibilities depending on the context. The only way to correctly sort Japanese index entries is to keep the phonetic counterparts with the written forms. The phonetic text would be presented as the sort order text for indexing purposes.

7.2.7 Merging index elements

Processors merge indexing elements in order to construct the effective index for a publication.

Processors *SHOULD* use the following criteria to match the content of `<indexterm>` elements:

- Ignore leading and trailing whitespace, as well as newline characters
- Be case sensitive
-

It is implementation-dependent as to whether processors consider the presence of phrase-level markup within `<indexterm>` elements when performing a matching operation.

Comment by Kristen J Eberlein on 11 August 2019

This is a new topic. It needs to lay out the rules and expectations for how a processor merges indexing elements and constructs the effective index, including:

- Matching the content of indexing elements
- Constructing multilevel index entries (as suggested in 7.2.8.1 Example: Merging indexterm elements (109))

What are other critical aspects?

Comment by Kristen J Eberlein on 08 July 2019

Do we make the following RFC-2119 statement for DITA 2.0. Thoughts?

FYI, the following text appeared in the DITA 1.1 - 1.3 specs: “All indexterms with the same content are ″merged″ to form a single index entry in the resulting index, and all contributed page numbers are included in that index entry.”

(For processors that support indexing) If multiple `<indexterm>` elements exist that would result in matching index entries, a processor *SHOULD* generate only a single index entry, although all locations associated with the `<indexterm>` element contribute locators.

Comment by Eliot Kimber on 09 August 2019

This is the correct behavior. By "identical content" is a little too strict. Probably need to say something like "content that the processor considers to be the same. For example, by normalizing white space."

Comment by Kristen J Eberlein on 13 August 2019

Eliot raised the following scenario in his review of the indexing content:

Consider a publication that contains the following index markup in different topics:

```
<!-- Topic A -->
<indexterm>apple</indexterm>
```

```
<!-- Topic B -->
<indexterm>apple
  <indexterm>computer</indexterm>
</indexterm>
```

What would the author of topic A expect to see in the index? What would the author of topic B expect to see? How might this combine with indexing markup added at the map level by the map owner?

Eliot suggested that a processor might produce any of the following (arrow character used to indicate indentation):

Option #1
apple 12
computer 14

Option #2
apple 12
apple
computer 14

Option #3
apple
computer 14

My personal expectation would be to expect to see option #1, which matches the statement made in the DITA 1.1-1.3 specs: “All indexterms with the same content are ″merged″ to form a single index entry in the resulting index, and all contributed page numbers are included in that index entry.”

If a company's styleguide does not want locators on parents of leaf nodes, I'd expect authors to spend time "massaging" index markup in order to generate an index that followed the styleguide requirements.

7.2.8 Examples of indexing

This section contains examples and scenarios that illustrate the use and processing of indexing elements.

7.2.8.1 Example: Merging <indexterm> elements

This example contains a multilevel `<indexterm>` element.

Given the following `<indexterm>` elements:

```
<indexterm>cheese
  <indexterm>sheeps milk
    <indexterm>pecorino</indexterm>
  </indexterm>
</indexterm>
<indexterm>cheese
  <indexterm>goats milk
    <indexterm>chevre</indexterm>
  </indexterm>
</indexterm>
```

A processor treats the `<indexterm>` elements as equivalent to the following multilevel `<indexterm>` element:

```
<indexterm>cheese
  <indexterm>sheeps milk
```

```
    <indexterm>pecorino</indexterm>
  </indexterm>
  <indexterm>goats milk
    <indexterm>chevre</indexterm>
  </indexterm>
</indexterm>
```

A processor generates the following index entries:

- A primary entry for "cheese"
- Secondary entries for "goats milk" and "sheeps milk"
- Tertiary entries for "chevre" and "pecorino" that include page numbers

The rendered index entry might look like the following:

cheese
    goats milk
        chevre 9
    sheeps milk
        pecorino 9

7.2.8.2 Example: Index range defined in a single topic

In this scenario, an index range is defined directly in the body of a topic. This strategy is useful for lengthy topics.

In the following code sample, the index range begins at the start of the second paragraph and continues to the beginning of the last paragraph. If the end element was not present, the index range would end at the end of the topic.

```
<topic id="accounting">
  <title>Accounting regulations</title>
  <body>
    <p>Be ethical in your accounting.</p>
    <p><indexterm start="acctrules">rules</indexterm>Remember to do all of the
following: ...</p>
    <!-- ...pages worth of rules... -->
    <p><indexterm end="acctrules"/>Failure to comply will get you audited.</p>
  </body>
  <!-- Potential sub-topics -->
</topic>
```

7.2.8.3 Example: Index range defined in a topic prolog

In this scenario, an index range is defined in the topic prolog. Ranges defined in a prolog cover subtopics, including those nested based on a map.

Specifying an index range in a topic prolog is useful for defining an index range that contains a topic and its children.

Consider the following DITA map which contains topics about a small company's operating procedures. The map contains a topic about accounting (`acct.dita`), which has child topics: `procedures.dita` and `forms.dita`.

```
<map>
  <title>Company procedures</title>
  <topicref href="acct.dita">
    <topicref href="procedures.dita"/>
    <topicref href="forms.dita"/>
  </topicref>
  ...
</map>
```

The information developer wants an index entry that will span `acct.dita` and its children. He uses the following markup in `acct.dita`:

```
<topic id="accounting-at-acme">
  <title>Accounting at Acme</title>
  <prolog>
    <metadata>
      <keywords>
        <indexterm start="acct">accounting</indexterm>
      </keywords>
    </metadata>
  </prolog>
  <!-- ... -->
</topic>
```

This markup specifies that the index range begins with the start of the topic title, and the end of the range is the end of the `forms.dita` topic. The index range includes the "Accounting at Acme" topic and its two child topics.

The information developer could have included an end element (for example, `<indexterm end="acct"/>`) in the topic prolog, but it is not necessary. If the topic prolog had included an end element, the effective index range would be identical.

Comment by Eliot Kimber on 09 August 2019

I think this rule has the same problem as for ending the range at topic body boundaries: it means you couldn't start a range in one topic's prolog and end it in the prolog of another sibling topic.

That would be a hard thing to manage so probably not a realistic use case, so I can see an argument that prolog-defined ranges span the effective topic tree rooted at the topic with the prolog. But there's no particular reason to impose the constraint.

That is, a processor that can handle index ranges can also detect when a range is started in a topic prolog but not ended in one and report the case.

7.2.8.4 Example: Index range defined in a map

In this scenario, an index range is defined in the DITA map. Ranges defined in a DITA map can span peer topics.

Consider the following DITA map:

```
<map>
  <title>Food available in the Acme cafeteria</title>
  <!-- ... -->
  <topicref href="apples.dita">
    <topicmeta>
      <keywords>
        <indexterm start="acme-fruit">fruit</indexterm>
      </keywords>
    </topicmeta>
  </topicref>
  <topicref href="oranges.dita"/>
  <topicref href="pineapples.dita">
    <topicmeta>
      <keywords>
        <indexterm start="acme-fruit"/>
      </keywords>
    </topicmeta>
  </topicref>
  <!-- ... -->
</map>
```

The index range begins with the start of the first topic title in `apples.dita`, and it continues until the end of the last element in `pineapples.dita`. If an end element was not specified, the range would continue to the end of the map.

Comment by Eliot Kimber on 09 August 2019
I think this should be a reportable error or warning since it's almost certainly not the author's intent for a range to span to the end of the publication.

Comment by Kristen J Eberlein on 13 August 2019

Eliot, I disagree. If the map is a submap that is consumed by a larger publication (for example, a map of examples of using keys), I think the default range boundary of "end of the map" might be entirely appropriate.

7.3 Content reference (conref)

The DITA conref attributes provide mechanisms for reusing content. DITA content references support reuse scenarios that are difficult or impossible to implement using other XML-based inclusion mechanisms like XInclude and entities. Additionally, DITA content references have rules that help ensure that the results of content inclusion remain valid after resolution

7.3.1 Conref overview

The DITA `@conref`, `@conkeyref`, `@conrefend`, and `@conaction` attributes provide mechanisms for reusing content within DITA topics or maps. These mechanisms can be used both to pull and push content.

This topic uses the definitions of referenced element (14) and referencing element (14) as defined in 2.6 Linking and addressing terminology (14).

Pulling content to the referencing element

When the `@conref` or `@conkeyref` attribute is used alone, the referencing element acts as a placeholder for the referenced element, and the content of the referenced element is rendered in place of the referencing element.

The combination of the `@conrefend` attribute with either `@conref` or `@conkeyref` specifies a range of elements that is rendered in place of the referencing element. Although the start and end elements must be of the same type as the referencing element (or specialized from that element type), the elements inside the range can be any type.

Pushing content from the referencing element

The `@conaction` attribute reverses the direction of reuse from pull to push. With a push, the referencing element is rendered before, after, or in place of the referenced element. The location (before, after, or in place of) is determined by the value of the `@conaction` attribute. Because the `@conaction` and `@conrefend` attributes cannot both be used within the same referencing element, it is not possible to push a range of elements.

A fragment of DITA content, such as an XML document that contains only a single paragraph without a topic ancestor, does not contain enough information for the conref processor to be able to determine the validity of a reference to it. Consequently, the value of a conref must specify one of the following items:

- A referenced element within a DITA map
- A referenced element within a DITA topic
- An entire DITA map

- An entire DITA topic

Related reference

The conaction attribute (377)

The `@conaction` attribute allows users to push content from one topic into another. It causes the `@conref` attribute to work in reverse, so that the content is pushed from the current topic into another, rather than pulled from another topic into the current one. Allowable values for `@conaction` are: "pushafter", "pushbefore", "pushreplace", mark"", and "-dita-use-conref-target".

The conkeyref attribute (384)

The `@conkeyref` attribute provides an indirect content reference to topic elements, map elements, or elements within maps or topics. When the DITA content is processed, the key references are resolved using key definitions from DITA maps.

The conref attribute (384)

The `@conref` attribute is used to reference content that can be reused. It allows reuse of DITA elements, including topic or map level elements.

The conrefend attribute (380)

The `@conrefend` attribute is used when referencing a range of elements with the conref mechanism. The `@conref` or `@conkeyref` attribute references the first element in the range, while `@conrefend` references the last element in the range.

7.3.2 Processing conrefs

When processing content references, DITA processors compare the restrictions of each context to ensure that the conrefed content is valid in its new context.

Note The DITA `@conref` attribute is a transclusion mechanism similar to XInclude and to HyTime value references. DITA differs from these mechanisms, however, in that conref validity does not apply simply to the current content at the time of replacement, but to the possible content given the restrictions of both the referencing document type and the referenced document type.

When content is reused between two documents with different domains or constraints, it is possible for the reused content to include domain extensions that are not defined for the new context, or to include elements that would be constrained out of the new context. When pulling or pushing content with the conref mechanism, processors resolving conrefs *SHOULD* tolerate specializations of valid elements. Processors *MAY* generalize elements in the pushed or pulled content fragment as needed for the resolving context.

Comment by robander
Without @domains, without constraints, and without a built-in knowledge of every document type, it is not be possible to determine every case that could be rendered invalid. As such, should we change the following MUST NOT to SHOULD NOT?

A conref processor *MUST NOT* permit resolution of a reuse relationship that could be rendered invalid under the rules of either the reused or reusing content.

7.3.3 Processing attributes when resolving conrefs

When resolving conrefs, processors need to combine the attributes that are specified on the referencing and referenced element.

The attribute specifications on the resolved element are drawn from both the referencing element and the referenced element, according to the following priority:

1. All attributes as specified on the referencing element, except for attributes set to "-dita-use-conref-target".
2. All attributes as specified on the referenced element except the `@id` attribute.
3. The `@xml:lang` attribute has special treatment as described in 7.6.1 The xml:lang attribute (136).

The token "-dita-use-conref-target" is defined by the specification to enable easier use of `@conref` on elements with required attributes. The only time the resolved element would include an attribute whose specified value is "-dita-use-conref-target" is when the referenced element had that attribute specified with the "-dita-use-conref-target" value and the referencing element either had no specification for that attribute or had it also specified with the "-dita-use-conref-target" value.

If the final resolved element (after the complete resolution of any conref chain) has an attribute with the "-dita-use-conref-target" value, that element *MUST* be treated as equivalent to having that attribute unspecified.

A given attribute value on the resolved element comes in its entirety from either the referencing element or the referenced element; the attribute values of the referencing and referenced elements for a given attribute are never additive, even if the property (such as `@audience`) takes a list of values.

Comment by Kristen J Eberlein on 04 July 2019

Do we want the following paragraph to contain normative RFC-2119 wording?

If the referenced element has a `@conref` attribute specified, the above rules should be applied recursively with the resolved element from one referencing/referenced combination becoming one of the two elements participating in the next referencing/referenced combination. The result should preserve without generalization all elements that are valid in the originating context, even if they are not valid in an intermediate context.

For example, if topic A and topic C allow highlighting, and topic B does not, then a content reference chain of topic A-to-topic B-to-topic C should preserve any highlighting elements in the referenced content. The result, however it is achieved, must be equivalent to the result of resolving the conref pairs recursively starting from the original referencing element in topic A.

Related reference

Using the -dita-use-conref-target value (385)

The value "-dita-use-conref-target" is available on enumerated attributes and can also be specified on other attributes. When an element uses `@conref` to pull in content, for any of its attributes assigned a value of "-dita-use-conref-target", the resulting value for those attributes is also pulled in from the referenced element.

7.3.4 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-

Paragraph	Value of `@id` attribute on conrefed paragraph	`<xref>` within conrefed paragraph	Resolution
			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`.

7.4 Conditional processing (profiling)

Conditional processing, also known as profiling, is the filtering or flagging of information based on processing-time criteria.

DITA defines attributes that can be used to enable filtering and flagging individual elements. `@props` and any attribute specialized from `@props` (including those integrated by default in OASIS shells: `@audience`, `@deliveryTarget`, `@platform`, `@product`, `@otherprops`) allow conditions to be assigned to elements so that the elements can be included, excluded, or flagged during processing. The `@rev` flagging attribute allows values to be assigned to elements so that special formatting can be applied to those elements during processing. A conditional-processing profile specifies which elements to include, exclude, or flag. DITA defines a document type called DITAVAL for creating conditional-processing profiles.

Processors *SHOULD* be able to perform filtering and flagging using the following attributes: `@props`, `@audience`, `@deliveryTarget`, `@platform`, `@product`, and `@otherprops`.

The `@props` attribute can be specialized to create new attributes, and processors *SHOULD* be able to perform conditional processing on specializations of `@props`.

Although metadata elements exist with similar names, such as the `<audience>` element, processors are not required to perform conditional processing using metadata elements.

Related concepts

Filtering and flagging attributes (32)

Conditional-processing attributes are available on most elements.

Related reference

DITAVAL elements (356)

A conditional processing profile (DITAVAL file) is used to identify which values are to be used for conditional processing during a particular output, build, or some other purpose. The profile should have an extension of `.ditaval`.

7.4.1 Conditional processing values and groups

Conditional processing attributes classify elements with metadata. The metadata is specified using space-delimited string values or grouped values.

For example, the string values within `@product` in `<p product="basic deluxe">` indicate that the paragraph applies to the “basic” product and to the “deluxe” product.

Groups can be used to organize classification metadata into subcategories. This is intended to support situations where a predefined metadata attribute applies to multiple specialized subcategories. The grouping syntax exactly matches the syntax used for generalized attributes, making it valid inside `@props` and any attribute specialized from `@props` (including those integrated by default in OASIS shells: `@audience`, `@deliveryTarget`, `@platform`, `@product`, `@otherprops`).

For example, the `@product` attribute can be used to classify an element based on both related databases and related application servers. Using groups for these subcategories allows each category to be processed independently; when filter conditions exclude all applicable databases within a group, the element can be safely excluded, regardless of any other `@product` conditions.

Groups can be used within `@props` and any attribute specialized from `@props` (including those integrated by default in OASIS shells: `@audience`, `@deliveryTarget`, `@platform`, `@product`, `@otherprops`). The following rules apply:

- Groups consist of a name immediately followed by a parenthetical group of zero or more space-delimited string values. For example, `"groupName(valueOne valueTwo)"`.
- Groups cannot be nested.
- If two groups with the same name are found in a single attribute, they are treated as if all values are specified in the same group. The following values for the `@otherprops` attribute are equivalent:

```
otherprops="groupA(a b) groupA(c) groupZ(APPNAME)"
otherprops="groupA(a b c) groupZ(APPNAME)"
```

- If both grouped values and ungrouped values are found in a single attribute, the ungrouped values belong to an implicit group; the name of that group matches the name of the attribute. Therefore, the following values for the `@product` attribute are equivalent:

```
product="a database(dbA dbB) b appserver(mySERVER) c"
product="product(a b c) database(dbA dbB) appserver(mySERVER)"
```

Setting a conditional processing attribute to an empty value, such as `product=""`, is equivalent to omitting that attribute from the element. An empty group within an attribute is equivalent to omitting that group from the attribute. For example, `<ph product="database() A">` is equivalent to `<ph product="A">`. Combining both rules into one example, `<ph product="operatingSystem()">` is equivalent to `<ph>`.

If two groups with the same name exist on different attributes, a rule specified for that group will evaluate the same way on each attribute. For example, if the group "sample" is specified within both `@platform` and `@otherprops`, a DITAVAL rule for `sample="value"` will evaluate the same in each attribute. If there is a need to distinguish between similarly named groups on different attributes, the best practice is to use more specific group names such as *platformGroupname* and *productGroupname*. Alternatively, DITAVAL rules can be specified based on the attribute name rather than the group name.

If the same group name is used in different attributes, a complex scenario could be created where different defaults are specified for different attributes, with no rule set for a group or individual value. In this case a value might end up evaluating differently in the different attributes. For example, a DITAVAL can set a default of "exclude" for `@platform` and a default of "flag" for `@product`. If no rules are specified for group `oddgroup()`, or for the value `oddgroup="edgecase"`, the attribute `platform="oddgroup(edgecase)"` will evaluate to "exclude" while `product="oddgroup(edgecase)"` will resolve to "flag". See 10.7.2 DITAVAL elements (356) for information on how to change default behaviors in DITAVAL profile.

Related reference

DITAVAL elements (356)

A conditional processing profile (DITAVAL file) is used to identify which values are to be used for conditional processing during a particular output, build, or some other purpose. The profile should have an extension of `.ditaval`.

7.4.2 Filtering

At processing time, a conditional processing profile can be used to specify profiling attribute values that determine whether an element with those values is included or excluded.

By default, values in conditional processing attributes that are not defined in a DITAVAL profile evaluate to "include". For example, if the value audience="novice" is used on a paragraph, but this value is not defined in a DITAVAL profile, the attribute evaluates to "include".

However, the DITAVAL profile can change this default to "exclude", so that any value not explicitly defined in the DITAVAL profile will evaluate to "exclude". The DITAVAL profile also can be used to change the default for a single attribute; for example, it can declare that values in the `@platform` attribute default to "exclude", while those in the `@product` attribute default to include. See 10.7.2 DITAVAL elements (356) for information on how to set up a DITAVAL profile and how to change default behaviors.

When deciding whether to include or exclude a particular element, a processor evaluates each attribute independently:

1. For each attribute:
    - If the attribute is empty, or contains only empty groups, it is equivalent to omitting the attribute from the element. If evaluated for the purposes of filtering, the attribute is treated as "include", because an omitted attribute cannot evaluate to "exclude".
    - If the attribute value does not contain any groups, then if any token in the attribute value evaluates to "include", the element evaluates to "include"; otherwise it evaluates to "exclude". In other words, the attribute evaluates to "exclude" only when **all** the values in that attribute evaluate to "exclude".
    - If the attribute value does include groups, evaluate as follows, treating ungrouped tokens together as a group:
        a. For each group (including the group of ungrouped tokens), if any token inside the group evaluates to "include", the group evaluates to "include"; otherwise it evaluates to "exclude". In other words, a group evaluates to "exclude" only when every token in that group evaluates to "exclude".
        b. If any group within an attribute evaluates to "exclude", that attribute evaluates to "exclude"; otherwise it evaluates to "include".
2. If **any single attribute** evaluates to exclude, the element is excluded.

For example, if a paragraph applies to three products and the publisher has chosen to exclude all of them, the processor will exclude the paragraph. This is true even if the paragraph applies to an audience or platform that is not excluded. But if the paragraph applies to an additional product that has not been excluded, then its content is still relevant for the intended output and is preserved.

Similarly, with groups, a step might apply to one application server and two database applications:

```
<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>
```

If a publisher decides to exclude the application server mySERVER, then the appServer() group evaluates to exclude. This can be done by setting `product="mySERVER"` to exclude *or* by setting `appServer="mySERVER"` to exclude. This means the step is excluded, regardless of how the values "ABC" or "dbOtherName" evaluate. If a rule is specified for both `product="mySERVER"` and `appServer="mySERVER"`, the rule for the more specific group name "appServer" takes precedence.

Similarly, if both "ABC" and "dbOtherName" evaluate to exclude, then the database() group evaluates to exclude and the element is excluded regardless of how the "mySERVER" value is set.

In more advanced usage, a DITAVAL can be used to specify a rule for a group name. For example, an author could create a DITAVAL rule that sets `product="database"` to "exclude". This is equivalent to setting a default of "exclude" for any individual value in a database() group; it also excludes the more common usage of "database" as a single value within the `@product` attribute. Thus when "myDB"

appears in a database() group within the `@product` attribute, the full precedence for determining whether to include or exclude the value is as follows:

1. Check for a DITAVAL rule for `database="myDB"`
2. Check for a DITAVAL rule for `product="myDB"`
3. Check for a DITAVAL rule for `product="database"` (default for the database group)
4. Check for a DITAVAL rule for "product" (default for the `@product` attribute)
5. Check for a default rule for all conditions (default of include or exclude for all attributes)
6. Otherwise, evaluate to "include"

Related reference

DITAVAL markup with extended filtering example (363)

The `<val>` element is the root element of a DITAVAL document.

7.4.3 Flagging

Flagging is the application of text, images, or styling during rendering. This can highlight the fact that content applies to a specific audience or operating system, for example; it also can draw a reader's attention to content that has been marked as being revised.

At processing time, a conditional processing profile can be used to specify profiling attribute values that determine whether an element with those values is flagged.

When deciding whether to flag a particular element, a processor evaluates each value. Wherever an attribute value that has been set as flagged appears (for example, `audience="administrator"`), the processor adds the flag. When multiple flags apply to a single element, multiple flags are rendered, typically in the order that they are encountered.

When the same element evaluates as both flagged and included, the element is both flagged and included. When the same element evaluates as both flagged and filtered (for example, flagged because of a value for the `@audience` attribute and filtered because of a value for the `@product` attribute value), the element is filtered.

Related reference

DITAVAL markup with extended flagging example (363)

The `<val>` element is the root element of a DITAVAL document.

DITAVAL markup for flagging revisions (360)

The `<revprop>` element in a DITAVAL document identifies a value in the `@rev` attribute that should be flagged in some manner. Unlike the conditional processing attributes, which can be used for both filtering and flagging, the `@rev` attribute only can be used for flagging.

7.4.4 Conditional processing to generate multiple deliverable types

By default, the content of most elements is included in all output media. Within maps and topics, elements can specify the delivery targets to which they apply.

Within maps, topic references can use the `@deliveryTarget` attribute to indicate the delivery targets to which they apply.

Within topics, most elements can use the `@deliveryTarget` attribute to indicate the delivery targets to which they apply.

If you want a referenced topic to be excluded from all output formats, set the `@processing-role` attribute to "resource-only" instead of using the `@deliveryTarget`. Content within that topic can still be referenced for display in other locations.

@deliveryTarget attribute

@deliveryTarget

The intended delivery target of the content, for example, "html", "pdf", or "epub".

The `@deliveryTarget` attribute is specialized from the `@props` attribute. It is defined in the deliveryTargetAttDomain, which is integrated into all OASIS-provided document-type shells. If this domain is not integrated into a given document-type shell, the `@deliveryTarget` attribute will not be available.

The `@deliveryTarget` attribute is processed the same way as any other conditional processing attribute. For example, the element `<topicref deliveryTarget="html5 epub" href="example.dita"/>` uses two values for `@deliveryTarget`. A conditional processing profile can then set rules for `@deliveryTarget` that determine whether the topic `example.dita` is included or excluded when the map is rendered as HTML5 or EPUB.

7.4.5 Examples of conditional processing

This section provides examples that illustrate the ways that conditional processing attributes can be set and used.

Related reference

DITAVAL markup with additional filtering and flagging examples (363)

The `<val>` element is the root element of a DITAVAL document.

7.4.5.1 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>
```

7.4.5.2 Example: Filtering and flagging topic content

A publisher might want to flag information that applies to administrators and exclude information that applies to the extended product.

Consider the following DITA source fragment and conditional processing profile:

Figure 47: DITA source fragment

```
<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>
```

Figure 48: DITAVAL profile

```
<val>
  <prop att="audience" val="administrator" action="flag">
    <startflag><alt-text>ADMIN</alt-text></startflag>
  </prop>
  <prop att="product" val="extendedProd" action="exclude"/>
</val>
```

When the content is rendered, the paragraph is flagged, and the first list item is excluded (since it applies to extendedProd). The second list item is still included; even though it does apply to extendedProd, it also applies to basicProd, which was not excluded.

The result will look something like the following:

ADMIN Set the configuration options:

- Set your blink rate
- Do some other stuff
- Do a special thing for Linux

7.5 Branch filtering

The branch filtering mechanism enables map authors to set filtering conditions for specific branches of a map. This makes it possible for multiple conditional-processing profiles to be applied within a single publication.

Without the branch filtering mechanism, the conditions specified in a DITAVAL document are applied globally. With branch filtering, the `<ditavalref>` element specifies a DITAVAL document that can be applied to a subset of content; the location of the `<ditavalref>` element determines the content to which filtering conditions are applied. The filtering conditions then are used to filter the map branch itself (map elements used to create the branch), as well as the local maps or topics that are referenced by that branch.

The `<ditavalref>` element also provides the ability to process a single branch of content multiple times, applying unique conditions to each instance of the branch.

7.5.1 Overview of branch filtering

Maps or map branches can be filtered by adding a <ditavalref> element that specifies the DITAVAL document to use for that map or map branch.

The <ditavalref> element is designed to manage conditional processing for the following use cases.

1. A map branch needs to be filtered using conditions that do not apply to the rest of the publication. For example, a root map might contain content that is written for both novice and expert users. However, the authors want to add a section that targets only novice users. Using branch filtering, a map branch can be filtered so that it only includes content germane to a novice audience, while the content of the rest of the map remains appropriate for multiple audiences.
2. A map branch needs to be presented differently for different audiences. For example, a set of software documentation might contain installation instructions that differ between operating systems. In that case, the map branch with the installation instructions needs to be filtered multiple times with distinct sets of conditions, while the rest of the map remains common to all operating systems.

Filtering rules often are specified globally, outside of the content. When global conditions set a property value to "exclude", that setting overrides any other settings for the same property that are specified at a branch level. Global conditions that set a conditional property to "include" or "flag" do not override branch-level conditions that set the same property to "exclude".

Using <ditavalref> elements, it is possible to specify one set of conditions for a branch and another set of conditions for a subset of the branch. As with global conditions, properties set to "exclude" for a map branch override any other settings for the same property specified for a subset of the branch. Branch conditions that set a conditional property to "include" or "flag" do not override conditions on a subset of the branch that explicitly set the same property to "exclude".

In addition to filtering, applications *MAY* support flagging at the branch level based on conditions that are specified in referenced DITAVAL documents.

7.5.2 Branch filtering: Single condition set for a branch

Using a single <ditavalref> element as a child of a map or map branch indicates that the map or map branch must be conditionally processed using the rules specified in the referenced DITAVAL document.

The following rules outline how the filtering conditions that are specified in DITAVAL document are applied:

<ditavalref> element as a direct child of a map
: The filtering conditions are applied to the entire map.

<ditavalref> element within a map branch
: The filtering conditions are used to process the entire branch, including the parent element that contains the <ditavalref> element.

<ditavalref> element within a <topicref> reference to a local map
: The filtering conditions are applied to the submap.

<ditavalref> element within a <topicref> reference to peer map
: The reference conditions are **not** applied to the peer map.

7.5.3 Branch filtering: Multiple condition sets for a branch

Using multiple <ditavalref> elements as the children of a map or map branch indicates that the map or map branch will be conditionally processed using the rules that are specified in the referenced DITAVAL documents.

When multiple <ditavalref> elements occur as children of the same element, the rules in the referenced DITAVAL documents are processed independently. This effectively requires a processor to

maintain one copy of the branch for each <ditavalref>, so that each copy can be filtered using different conditions.

Note In most cases, it is possible to create a valid, fully-resolved view of a map with branches copied to reflect the different <ditavalref> conditions. However, this might not be the case when multiple <ditavalref> elements occur as direct children of a root map. In this case, it is possible that the map could be filtered in a manner that results in two or more distinct versions of the <title> or metadata. How this is handled is processor dependent. For example, when a root map has three <ditavalref> elements as children of <map>, a conversion to EPUB could produce an EPUB with three versions of the content, or it could produce three distinct EPUB documents.

When a processor maintains multiple copies of a branch for different condition sets, it has to manage situations where a single resource with a single key name results in two distinct resources. Key names must be modified in order to allow references to a specific filtered copy of the resource; without renaming, key references could only be used to refer to a single filtered copy of the resource, chosen by the processor. See 7.5.4 Branch filtering: Impact on resource and key names (124) for details on how to manage resource names and key names.

7.5.4 Branch filtering: Impact on resource and key names

When map branches are cloned by a processor in order to support multiple condition sets, processors must manage conflicting resource and key names. The DITAVALref domain includes metadata elements that authors can use to specify how resource and key names are renamed.

Note While the processing controls that are described here are intended primarily for use with map branches that specify multiple condition sets, they also can be used with map branches that include only a single <ditavalref> element.

When a map branch uses multiple condition sets, processors create multiple effective copies of the branch to support the different conditions. This results in potential conflicts for resource names, key names, and key scopes. Metadata elements inside of the <ditavalref> element are available to provide control over these values, so that keys, key scopes, and URIs can be individually referenced within a branch.

For example, the following map branch specifies two DITAVAL documents:

```
<topicref href="productFeatures.dita" keys="features" keyscope="prodFeatures">
  <ditavalref href="novice.ditaval"/>
  <ditavalref href="admin.ditaval"/>
  <topicref href="newFeature.dita" keys="newThing"/>
</topicref>
```

In this case, the processor has two effective copies of `productFeatures.dita` and `newFeature.dita`. One copy of each topic is filtered using the conditions specified in `novice.ditaval`, and the other copy is filtered using the conditions specified in `admin.ditaval`. If an author has referenced a topic using `keyref="features"` or `keyref="prodFeatures.features"`, there is no way for a processor to distinguish which filtered copy is the intended target.

Metadata elements in the DITAVAL reference domain

Metadata within the <ditavalref> element makes it possible to control changes to resource names and key scope names, so that each distinct filtered copy can be referenced in a predictable manner.

<dvrResourcePrefix>
Enables a map author to specify a prefix that is added to the start of resource names for each resource in the branch.

<dvrResourceSuffix>
Enables a map author to specify a suffix that is added to the end of resource names (before any extension) for each resource in the branch.

<dvrKeyscopePrefix>
Enables a map author to specify a prefix that is added to the start of key scope names for each key scope in the branch. If no key scope is specified for the branch, this can be used to establish a new key scope, optionally combined with a value specified in `<dvrKeyscopeSuffix>`.

<dvrKeyscopeSuffix>
Enables a map author to specify a suffix that is added to the end of key scope names for each key scope in the branch.

For example, the previous code sample can be modified as follows to create predictable resource names and key scopes for the copy of the branch that is filtered using the conditions that are specified in `admin.ditaval`.

```
<topicref href="productFeatures.dita" keys="features" keyscope="prodFeatures">
  <ditavalref href="novice.ditaval"/>
  <ditavalref href="admin.ditaval">
    <ditavalmeta>
      <dvrResourcePrefix>admin-</dvrResourcePrefix>
      <dvrKeyscopePrefix>adminscope-</dvrKeyscopePrefix>
    </ditavalmeta>
  </ditavalref>
  <topicref href="newFeature.dita" keys="newThing"/>
</topicref>
```

The novice branch does not use any renaming, which allows it to be treated as the default copy of the branch. As a result, when the topics are filtered using the conditions that are specified in `novice.ditaval`, the resource names and key names are unmodified, so that references to the original resource name and key name will resolve to topics in the novice copy of the branch. This has the following effect on topics that are filtered using the conditions specified in `admin.ditaval`:

- The prefix `admin-` is added to the beginning of each resource name in the admin branch.
  - The resource `productFeatures.dita` becomes `admin-productFeatures.dita`
  - The resource `newFeature.dita` becomes `admin-newFeature.dita`
- The prefix `adminscope-` is added to the existing key scope "prodFeatures".
  - The attribute value `keyref="adminscope-prodFeatures.features"` refers explicitly to the admin copy of `productFeatures.dita`
  - The attribute `keyref="adminscope-prodFeatures.newThing"` refers explicitly to the admin copy of `newFeature.dita`

Note In general, the best way to reference a topic that will be modified based on branch filtering is to use a key rather than a URI. Key scopes and key names (including those modified based on the elements above) must be calculated by processors before other processing. This means that in the example above, a key reference to `adminscope-prodFeatures.features` will always refer explicitly to the instance of `productFeatures.dita` filtered against the conditions in `admin.ditaval`, regardless of whether a processor has performed the filtering yet. References that use the URI `productFeatures.dita` or `admin-productFeatures.dita` could resolve differently (or fail to resolve), as discussed in 7.5.5 Branch filtering: Implications of processing order (126).

Renaming based on multiple <ditavalref> elements

It is possible for a branch with `<ditavalref>` already in effect to specify an additional `<ditavalref>`, where each `<ditavalref>` includes renaming metadata. When renaming, metadata on the `<ditavalref>` nested more deeply within the branch appears closer to the original resource or key name. For example:

```
<topicref href="branchParent.dita">
  <ditavalref href="parent.ditaval">
    <ditavalmeta>
      <dvrResourcePrefix>parentPrefix-</dvrResourcePrefix>
    </ditavalmeta>
  </ditavalref>
  <!-- additional topics or layers of nesting -->
  <topicref href="branchChild.dita">
    <ditavalref href="child.ditaval">
      <ditavalmeta>
        <dvrResourcePrefix>childPrefix-</dvrResourcePrefix>
      </ditavalmeta>
    </ditavalref>
  </topicref>
</topicref>
```

In this situation, the resource `branchChild.dita` is given a prefix based on both the reference to `parent.ditaval` and the reference to `child.ditaval`. The value "childPrefix-" is specified in the `<ditavalref>` that is nested more deeply within the branch, so it appears closer to the original resource name. The resource `branchChild.dita` would result in `parentPrefix-childPrefix-branchChild.dita`. Suffixes (if specified) would be added in a similar manner, resulting in a name like `branchChild-childSuffix-parentSuffix.dita`. Note that the hyphens are part of the specified prefix; they are not added automatically.

Handling resource name conflicts

It is an error if `<ditavalref>`-driven branch cloning results in multiple copies of a topic that have the same resolved name. Processors *SHOULD* report an error in such cases. Processors *MAY* recover by using an alternate naming scheme for the conflicting topics.

In rare cases, a single topic might appear in different branches that set different conditions, yet still produce the same result. For example, a topic might appear in both the admin and novice copies of a branch but not contain content that is tailored to either audience; in that case, the filtered copies would match. A processor *MAY* consider this form of equivalence when determining if two references to the same resource should be reported as an error.

7.5.5 Branch filtering: Implications of processing order

Because the branch filtering process can result in new or renamed keys, key scopes, or URIs, the full effects of the branch filtering process *MUST* be calculated by processors before they construct the effective map and key scope structure.

> **Note** The `@keyref` attribute and related attributes are explicitly disallowed on `<ditavalref>`. This prevents any confusion resulting from a `@keyref` that resolves to additional key- or resource-renaming metadata.

In general, the DITA specification refrains from mandating a processing order; thus publication results can vary slightly depending on the order in which processes are run. With branch filtering, processors are not required to apply filter conditions specified outside of the map and filter conditions specified with `<ditavalref>` at the same time in a publishing process.

For example, a processor might use the following processing order:

1. Apply externally-specified filter conditions to maps
2. Apply filtering based on <ditavalref> elements

Because externally-specified "exclude" conditions always take precedence over branch-specific conditions, content excluded based on external conditions will always be removed, regardless of the order in which processors evaluate conditions.

Processors should consider the following points when determining a processing order:

- If links are generated based on the map hierarchy, those links should be created using the renamed keys and URIs that result from evaluating the <ditavalref> filter conditions, to ensure that the links are consistent within the modified branches. For example, sequential links based on a map hierarchy should remain within the appropriate modified branch.
- If conrefs are resolved in topics before the <ditavalref> filtering conditions are evaluated, content that applies to multiple audiences can be brought in and (later in the process) selectively filtered by branch. For example, if a set of installation steps is pulled in with conref (from outside the branch), it might contain information that is later filtered by platform based on <ditavalref>. This results in copies of the steps that are specific to each operating system. If conref is processed after the <ditavalref>, content might be pulled in that has not been appropriately filtered for the new context.
- The same scenario applies to conref values that push content into the branch.
  - Pushing content into a branch before resolving the <ditavalref> conditions allows content for multiple conditions to be pushed and then filtered by branch based on the <ditavalref> conditions.
  - If the branch using <ditavalref> pushes content elsewhere, resolving <ditavalref> first could result in multiple copies of the content to be pushed (one for each branch), resulting in multiple potentially conflicting copies pushed to the new destination.

7.5.6 Examples of branch filtering

The branch filtering examples illustrate the processing expectations for various scenarios that involve <ditavalref> elements. Processing examples use either before and after sample markup or expanded syntax that shows the equivalent markup without the <ditavalref> elements.

7.5.6.1 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:

Figure 49: input.ditamap:

```
<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 50: Contents of novice.ditaval

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

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.

7.5.6.2 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.

Figure 51: input.ditamap

```
<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 52: Contents of win.ditaval

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

Figure 53: Contents of mac.ditaval

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

Figure 54: Contents of linux.ditaval

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

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.

Figure 55: input.ditamap

```
<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>
```

7.5.6.3 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>
```

7.5.6.4 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 is filtered using the conditions specified in the some.ditaval document.

Figure 56: Map fragment

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

Figure 57: Contents of other.ditamap

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

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.

7.5.6.5 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.

Figure 58: input.ditamap

```
<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 59: Contents of mac.ditaval

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

Figure 60: Contents of linux.ditaval

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

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.

7.5.6.6 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.

Figure 61: Map fragment

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

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:

Figure 62: Map fragment

```
<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>
```

7.5.6.7 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`.

7.5.6.8 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 processors need to report.
- 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`.

7.6 Translation and localization

DITA has features that facilitate preparing content for translation and working with multilingual content, including the `@xml:lang` attribute, the `@dir` attribute, and the `@translate` attribute. In addition, the `<sort-as>` element provides support for sorting in languages in which the correct sorting of an element requires text that is different from the base content of the element.

7.6.1 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 `@xml:lang` attribute *SHOULD* be explicitly set on the root element of each map and topic.

Setting the `@xml:lang` attribute in the DITA source ensures that processors handle content in a language- and locale-appropriate way. If the `@xml:lang` attribute is not set, processors assume a default value which might not be appropriate for the DITA content.

When the `@xml:lang` attribute is specified for a document, DITA processors *MUST* use the specified value to determine the language of the document.

Setting the `@xml:lang` attribute in the source language document facilitates the translation process; it enables translation tools (or translators) to simply change the value of the existing `@xml:lang` attribute to the value of the target language. Some translation tools support changing the value of an existing `@xml:lang` attribute, but they do not support adding new markup to the document that is being translated. Therefore, if source language content does not set the `@xml:lang` attribute, it might be difficult or impossible for the translator to add the `@xml:lang` attribute to the translated document.

If the root element of a map or a top-level topic has no value for the`@xml:lang` attribute, a processor *SHOULD* assume a default value. The default value of the processor can be either fixed, configurable, or derived from the content itself, such as the `@xml:lang` attribute on the root map.

The `@xml:lang` attribute is described in the XML Recommendation. Note that the recommended style for the `@xml:lang` attribute is lowercase language and (optional) uppercase, separated by a hyphen, for example, "en-US" or "sp-SP" or "fr". According to RFC 5646, *Tags for Identifying Languages,* language codes are case insensitive.

Recommended use in topics

For a DITA topic that contains a single language, set the `@xml:lang` attribute on the highest-level element that contains content.

Comment by Kristen J Eberlein on 04 July 2019

Should the following paragraph contain a *SHOULD* statement?

When a DITA topic contains more than one language, set the `@xml:lang` attribute on the highest-level element to specify the *primary language and locale* that applies to the topic. If part of a topic is written in a different language, authors should ensure that the part is enclosed in an element with the `@xml:lang` attribute set appropriately. This method of overriding the default document language applies to both block and inline elements that use the alternate language.

Processors *SHOULD* style each element in a way that is appropriate for its language as identified by the `@xml:lang` attribute.

Recommended use in maps

The `@xml:lang` attribute can be specified on the `<map>` element. The `@xml:lang` attribute cascades within the map in the same way that it cascades within a topic. However, since the `@xml:lang` attribute is an inherent property of the XML document, the value of the `@xml:lang` attribute does not cascade from one map to another or from a map to a topic; the value of the `@xml:lang` attribute that is specified in a map does not override `@xml:lang` values that are specified in other maps or in topics.

The primary language for the map *SHOULD* be set on the `<map>` element. The specified language remains in effect for all child `<topicref>` elements, unless a child specifies a different value for the `@xml:lang` attribute.

When no `@xml:lang` value is supplied locally or on an ancestor, a processor-determined default value is assumed.

Recommended use with the @conref or @conkeyref attribute

When a `@conref` or `@conkeyref` attribute is used to include content from one element into another, the processor *MUST* use the effective value of the `@xml:lang` attribute from the referenced element, that is, the element that contains the content. If the referenced element does not have an explicit value for the `@xml:lang` attribute, the processor *SHOULD* default to using the same value that is used for topics that do not set the `@xml:lang` attribute.

This behavior is shown in the following example, where the value of the `@xml:lang` attribute of the included note is obtained from its parent `<section>` element that sets the `@xml:lang` attribute to "fr". When the `installingAcme.dita` topic is processed, the `<note>` element with the `@id` attribute set to "mynote" has an effective value for the `@xml:lang` attribute of "fr".

Figure 63: installingAcme.dita

```
<?xml version="1.0"?>
<!DOCTYPE task PUBLIC "-//OASIS//DTD DITA Task//EN" "task.dtd">
<task xml:lang="en" id="install_acme">
 <title>Installing Acme</title>
 <shortdesc>Step-by-step details about how to install Acme.</shortdesc>
 <taskbody>
   <prereq>
   <p>Special notes when installing Acme in France:</p>
   <note id="mynote" conref="warningsAcme.dita#topic_warnings/frenchwarnings"/>
  </prereq>
```

```
 </taskbody>
</task>
```

Figure 64: warningsAcme.dita

```
<?xml version="1.0"?>
<!DOCTYPE topic PUBLIC "-//OASIS//DTD DITA Topic//EN" "topic.dtd">
<topic id="topic_warnings">
 <title>Warnings</title>
 <body>
  <section id="qqwwee" xml:lang="fr">
   <title>French warnings</title>
   <p>These are our French warnings.</p>
   <note id="frenchwarnings">Note in French!</note>
  </section>
  <section xml:lang="en">
   <title>English warnings</title>
   <p>These are our English warnings.</p>
   <note id="englishwarnings">Note in English!</note>
  </section>
 </body>
</topic>
```

7.6.2 The @dir attribute

The `@dir` attribute provides instructions to processors about how *bi-directional text* is rendered.

Bi-directional text is text that contains text in both text directionalities, right-to-left (RTL) and left-to-right (LTR). For example, languages such as Arabic, Hebrew, Farsi, Urdu, and Yiddish have text written from right-to-left; however, numerics and embedded sections of Western language text are written from left to right. Some multilingual documents also contain a mixture of text segments in two directions.

DITA contains the following attributes that have an effect on bi-directional text processing:

@xml:lang
: Identifies the language and locale, and so can be used to identify text that requires bi-directional rendering.

@dir
: Identifies or overrides the text directionality. It can be set to "ltr", "rtl"," lro", or "rlo".

In general, properly-written mixed text does not need any special markers; the Unicode bidirectional algorithm positions the punctuation correctly for a given language. The processor is responsible for displaying the text properly. However, some rendering systems might need directions for displaying bidirectional text, such as Arabic, properly. For example, Apache FOP might not render Arabic properly unless the left-to-right and right-to-left indicators are used.

The use of the `@dir` attribute and the Unicode algorithm is explained in the article Specifying the direction of text and tables: the `dir` attribute (http://www.w3.org/TR/html4/struct/dirlang.html#h-8.2) . This article contains several examples of how to use the `@dir` attribute set to either "ltr" or "rtl". There is no example of setting the `@dir` attribute to either "lro" or "rlo", although it can be inferred from the example that uses the `<bdo>` element, a now-deprecated W3C mechanism for overriding the entire Unicode bidirectional algorithm.

Recommended usage

The `@dir` attribute, together with the `@xml:lang` attribute, is essential for rendering table columns and definition lists in the proper order.

In general text, the Unicode Bidirectional algorithm, as specified by the `@xml:lang` attribute together with the `@dir` attribute, provides for various levels of bidirectionality:

- Directionality is either explicitly specified via the `@xml:lang` attribute in combination with the `@dir` attribute on the highest level element (topic or derived peer for topics, map for DITA maps) or assumed by the processing application.
- When embedding a right-to-left text run inside a left-to-right text run (or vice-versa), the default direction might provide incorrect results based on the rendering mechanism, especially if the embedded text run includes punctuation that is located at one end of the embedded text run. Unicode defines spaces and punctuation as having neutral directionality and defines directionality for these neutral characters when they appear between characters having a strong directionality (most characters that are not spaces or punctuation). While the default direction is often sufficient to determine the correct directionality of the language, sometimes it renders the characters incorrectly (for example, a question mark at the end of a Hebrew question might appear at the beginning of the question instead of at the end or a parenthesis might render incorrectly). To control this behavior, the `@dir` attribute is set to "ltr" or "rtl" as needed, to ensure that the desired direction is applied to the characters that have neutral bidirectionality. The "ltr" and "rtl" values override only the neutral characters (for example, spaces and punctuation), not all Unicode characters.

  Note Problems with Unicode rendering can be caused by the rendering mechanism. The problems are not due to the XML markup itself.
- Sometimes you might want to override the default directionality for strongly bidirectional characters. Overrides are done using the "lro" and "rlo" values, which overrides the Unicode Bidirectional algorithm. This override forces a direction on the contents of the element. These override attributes give the author a brute force way of setting the directionality independent of the Unicode Bidirectional algorithm. The gentler "ltr" and "rtl" values have a less radical effect, only affecting punctuation and other so-called neutral characters.

For most authoring needs, the "ltr" and "rtl" values are sufficient. Use the override values only when you cannot achieve the desired effect using the "ltr" and "rtl" values.

Processing expectations

Applications that process DITA documents, whether at the authoring, translation, publishing, or any other stage, *SHOULD* fully support the Unicode bidirectional algorithm to correctly implement the script and directionality for each language that is used in the document.

Applications *SHOULD* ensure that the root element in every topic document and the root element in the root map has values for the `@dir` and `@xml:lang` attributes.

Related information

What you need to know about the BIDI algorithm and inline markup

XHTML Bi-directional Text Attribute Module

Specifying the direction of text and tables: the `@dir` attribute

HTML 4.0 Common Attributes

7.7 Sorting

Processors can be configured to sort elements. Typical processing includes sorting glossary entries, index entries, lists of parameters or reference entries in custom navigation structures, and tables based on the contents of cells in specific columns or rows.

Each element to be sorted must have some inherent text on which it will be sorted. This text is the *base sort phrase* for the element. For elements that have titles, the base sort phrase usually is the content of the `<title>` element. For elements that do not have titles, the base sort phrase might be literal content

in the DITA source, or it might be generated or constructed based on the semantics of the element involved; for example, it could be constructed from various attribute or metadata values.

Processors that perform sorting *SHOULD* explicitly document how the base sort phrase is determined for a given element.

The `<sort-as>` element can be used to specify an effective sort phrase when the base sort phrase is not appropriate for sorting. For index terms, the `<sort-as>` element specifies the effective sort phrase for an index entry.

The details of sorting and grouping are implementation specific. Processors might provide different mechanisms for defining or configuring collation and grouping details. Even where the `<sort-as>` element is specified, two processors might produce different sorted and grouped results because they might use different collation and grouping rules. For example, one processor might be configured to sort English terms before non-English terms, while another might be configured to sort them after. The grouping and sorting of content is subject to local editorial rules.

When a `<sort-as>` element is specified, processors that sort the containing element *MUST* construct the effective sort phrase by prepending the content of the `<sort-as>` element to the base sort phrase. This ensures that two items with the same `<sort-as>` element but different base sort phrases will sort in the appropriate order.

For example, if a processor uses the content of the `<title>` element as the base sort phrase, and the title of a topic is "24 Hour Support Hotline" and the value of the `<sort-as>` element is "twenty-four hour", then the effective sort phrase would be "twenty-four hour24 Hour Support Hotline".

Related reference

sort-as (353)

For elements that are sorted, the `<sort-as>` element provides text that is combined with the base sort phrase to construct the effective sort phrase.

8 Configuration, specialization, generalization, constraints, and expansion

The extension facilities of DITA allow document-type shells, vocabulary modules, and element-configuration modules (constraints and expansion) to be combined to create specific DITA document types. Vocabulary modules also can be specialized to meet requirements that are not satisfied by existing markup.

8.1 Overview of DITA extension facilities

DITA provides three extension facilities: Document-type configuration, specialization, and element-type configuration. In addition, generalization augments specialization.

Document-type configuration

Document-type configuration enables the definition of DITA document types that include only the vocabulary modules that are required for a given set of documents. There is no need to modify the vocabulary modules. Configurations are implemented as document-type shells.

Specialization

Specialization enables the creation of new element types in a way that preserves the ability to interchange those new element types with conforming DITA applications. Specializations are implemented as vocabulary modules, which are integrated into document-type shells.

Specializations are declare the elements and entities that are unique to a specialization. The separation of the vocabulary and its declarations into modules makes it easy to extend existing modules, because new modules can be added without affecting existing document types. It also makes it easy to assemble elements from different sources into a single document-type shell and to reuse specific parts of the specialization hierarchy in more than one document-type shell.

Element-type configuration

Element-type configuration enables DITA architects to modify the content models and attribute lists for individual elements, without modifying the vocabulary modules in which the elements are defined.

There are two types of element configuration: Constraints and expansion. Both constraints and expansions are implemented as modules that are integrated into document-type shells:

Constraints

Constraint modules enables the restriction of content models and attribute lists for individual elements.

Expansions

Expansion modules enable the expansion of content models and attribute lists for individual elements.

Generalization

Generalization is the process of reversing a specialization. It converts specialized elements or attributes into the original types from which they were derived.

8.2 Document-type configuration

Document-type configuration enables the definition of DITA document types that include only the vocabulary modules that are required for a given set of documents. There is no need to modify the vocabulary modules. Configurations are implemented as document-type shells.

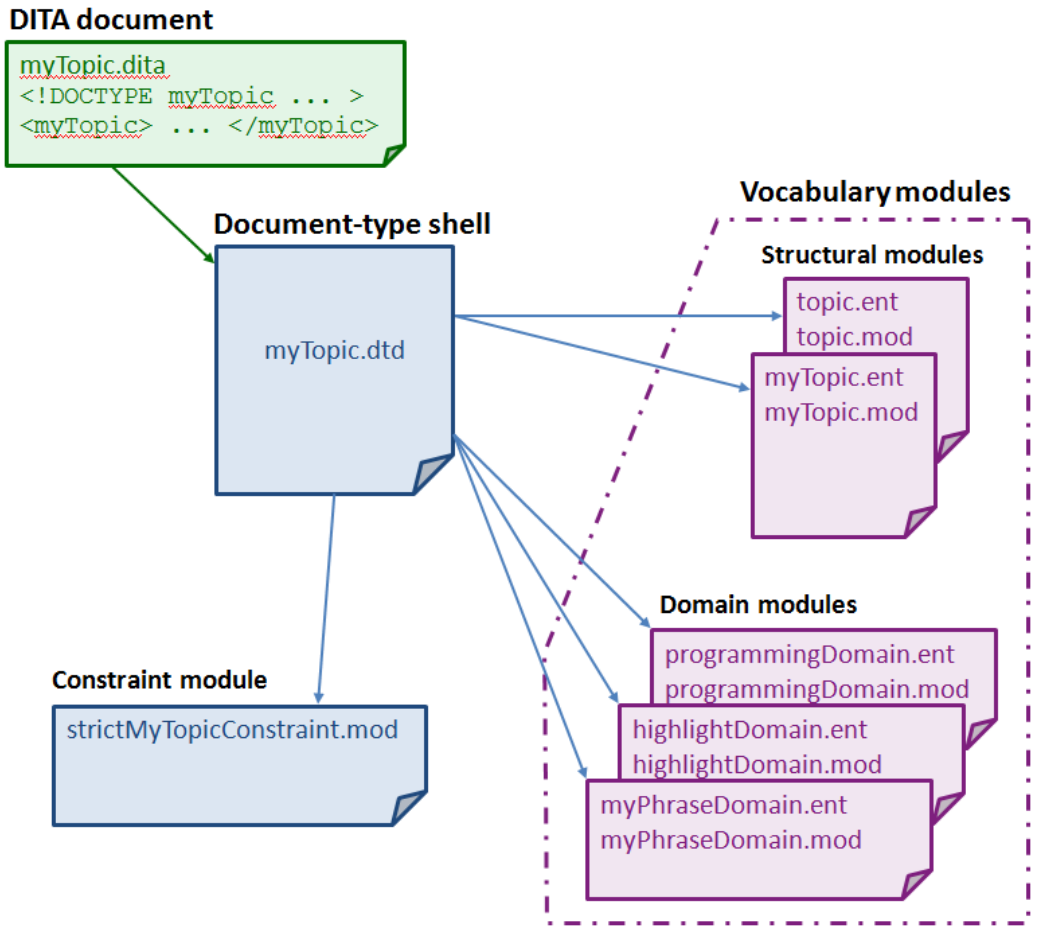
8.2.1 Overview of document-type shells

A document-type shell is an XML grammar file that specifies the elements and attributes that are allowed in a DITA document. The document-type shell integrates structural modules, domain modules, and element-configuration modules. In addition, a document-type shell specifies whether and how topics can nest.

A DITA document must either have an associated document-type definition or all required attributes must be made explicit in the document instances. Most DITA documents have an associated document-type shell. DITA documents that reference a document-type shell can be validated using standard XML processors. Such validation enables processors to read the XML grammar files and determine default values for the `@specializations` and `@class` attributes.

The following figure illustrates the relationship between a DTD-based DITA document, its document-type shell, the vocabulary modules that it uses, and the element-configuration modules (constraints and expansions) that it integrates. Similar structure applies to DITA documents that use other XML grammars.

Figure 65: Document type shell



Comment by Kristen J Eberlein on 28 March 2021

The illustration needs to be updated to include expansion modules.

The DITA specification contains a starter set of document-type shells. These document-type shells are commented and can be used as templates for creating custom document-type shells. While the OASIS-provided document-type shells can be used without any modification, creating custom document-type shells is a best practice. If the document-type shells need to be modified in the future, for example, to include a specialization or integrate a constraint, the existing DITA documents will not need to be modified to reference a new document-type shell.

8.2.2 Rules for document-type shells

This topic collects the rules that concern DITA document-type shells.

XML grammars

While the DITA specification only defines coding requirements for DTD, RELAX NG, and XML Schema documents, conforming DITA documents *MAY* use other document-type constraint languages, such as Schematron.

Defining element or attribute types

With two exceptions, a document-type shell *MUST NOT* directly define element or attribute types; it only includes vocabulary and element-configuration modules (constraints and expansions). The exceptions to this rule are the following:

- The ditabase document-type shell directly defines the `<dita>` element.
- RNG-based shells directly specify values for the `@specializations` attribute; these values reflect the details of the attribute domains that are integrated by the document-type shell.

Document-type shells not provided by OASIS

Document-type shells that are not provided by OASIS *MUST* have a unique public identifier, if public identifiers are used.

Document-type shells that are not provided by OASIS *MUST NOT* indicate OASIS as the owner; the public identifier or URN for such document-type shells *SHOULD* reflect the owner or creator of the document-type shell.

For example, if `example.com` creates a copy of the document type shell for topic, an appropriate public identifier would be "-//EXAMPLE//DTD DITA Topic//EN", where "EXAMPLE" is the owner identifier component of the public identifier. An appropriate URN would be "urn:example.com:names:dita:rng:topic.rng".

8.2.3 Equivalence of document-type shells

Two distinct DITA document types that are taken from different tools or environments might be functionally equivalent.

A DITA document type is defined by the following:

- The set of vocabulary and element-configuration modules (constraints and expansion) that are integrated by the document type shell
- The values of the `@class` attributes of all the elements in the document
- Rules for topic nesting

Two document-type shells define the same DITA document type if they integrate identical vocabulary modules, element-configuration modules (constraint and expansion), and rules for topic nesting. For

example, a document type shell that is an unmodified copy of the OASIS-provided document-type shell for topic defines the same DITA document type as the original document-type shell. However, the new document-type shell has the following differences:

- It is a distinct file that is stored in a different location.
- It has a distinct system identifier.
- If it has a public identifier, the public identifier is unique.

Note The public or system identifier that is associated with a given document-type shell is not, by itself, necessarily distinguishing. This is because two different people or groups might use the same modules and constraints to assemble equivalent document type shells, while giving them different names or public identifiers.

8.2.4 Conformance of document-type shells

DITA documents typically are governed by a conforming DITA document-type shell. However, the conformance of a DITA document is a function of the document instance, not its governing grammar. Conforming DITA documents are not required to use a conforming document-type shell.

Conforming DITA documents are not required to have any governing document type declaration or schema. There might be compelling or practical reasons to use non-conforming document-type shells. For example, a document might use a document-type shell that does not conform to the DITA requirements for shells in order to meet the needs of a specific application or tool. Such a non-conforming document-type shell still might enable the creation of conforming DITA content.

8.3 Specialization

The specialization feature of DITA allows for the creation of new element types and attributes that are explicitly and formally derived from existing types. This facilitates interchange of conforming DITA content and ensures a minimum level of common processing for all DITA content. It also allows specialization-aware processors to add specialization-specific processing to existing base processing.

8.3.1 Overview of specialization

Specialization allows information architects to define new kinds of information (new structural types or new domains of information), while reusing as much of existing design and code as possible, and minimizing or eliminating the costs of interchange, migration, and maintenance.

Specialization modules enable information architects to create new element types and attributes. These new element types and attributes are derived from existing element types and attributes.

In traditional XML applications, all semantics for a given element instance are bound to the element type, such as `<para>` for a paragraph or `<title>` for a title. The XML specification provides no built-in mechanism for relating two element types to say "element type B is a subtype of element type A".

In contrast, the DITA specialization mechanism provides a standard mechanism for defining that an element type or attribute is derived from an ancestor type. This means that a specialized type inherits the semantics and default processing behavior from its ancestor type. Additional processing behavior optionally can be associated with the specialized descendant type.

For example, the `<section>` element type is part of the DITA base or core. It represents an organizational division in a topic. Within the task information type (itself a specialization of `<topic>`), the `<section>` element type is further specialized to other element types (such as `<prereq>` and `<context>`) that provide more precise semantics about the type of organizational division that they represent. The specialized element types inherit both semantic meaning and default processing from the ancestor elements.

There are two types of DITA specializations:

Structural specialization

Structural specializations are developed from either topic or map types. Structural specializations enable information architect to add new document types to DITA. The structures defined in the new document types either directly use or inherit from elements found in other document types. For example; concept, task, and reference are specialized from topic, whereas bookmap is specialized from map.

Domain specialization

Domain specializations are developed from elements defined with topic or map, or from the `@props` or `@base` attributes. They define markup for a specific information domain or subject area. Domain specializations can be added to document-type shells.

Each type of specialization module represents an “is a” hierarchy, in object-oriented terms, with each structural type or domain being a subclass of its parent. For example, a specialization of task is still a task, and a specialization of the user interface domain is still part of the user interface domain. A given domain can be used with any map or topic type. In addition, specific structural types might require the use of specific domains.

Use specialization when you need a new structural type or domain. Specialization is appropriate in the following circumstances:

- You need to create markup to represent new semantics (meaningful categories of information). This might enable you to have increased consistency or descriptiveness in your content model.
- You have specific needs for output processing and formatting that cannot be addressed using the current content model.

Do not use specialization to simply eliminate element types from specific content models. Use constraint modules to restrict content models and attribute lists without changing semantics.

8.3.2 Modularization

Modularization is at the core of DITA design and implementation. It enables reuse and extension of the DITA specialization hierarchy.

The DITA XML grammar files are a set of module files that declare the markup and entities that are required for each specialization. The document-type shell then integrates the modules that are needed for a particular authoring and publishing context.

Because all the pieces are modular, the task of developing a new information type or domain is easy. An information architect can start with existing base types (topic or map)—or with an existing specialization if it comes close to matching their business requirements—and only develop an extension that adds the extra semantics or functionality that is required. A specialization reuses elements from ancestor modules, but it only needs to declare the elements and attributes that are unique to the specialization. This saves considerable time and effort; it also reduces error, enforces consistency, and makes interoperability possible.

Because all the pieces are modular, it is easy to reuse different modules in different contexts.

For example, a company that produces machines can use the task requirements and hazard statements domains, while a company that produces software can use the software, user interface, and programming domains. A company that produces health information for consumers can avoid using any of the standard domains, and instead develop a new domain that contains the elements necessary for capturing and tracking the comments made by medical professionals who review their information for accuracy and completeness.

Because all the pieces are modular, new modules can be created and put into use without affecting existing document-type shells. For example, a marketing division of a company can develop a new

specialization for message campaigns and have their content authors begin using that specialization, without affecting any of the other information types that they have in place.

8.3.3 Vocabulary modules

A DITA element type or attribute is declared in exactly one vocabulary module.

The following terminology is used to refer to DITA vocabulary modules:

structural module
: A vocabulary module that defines a top-level map or topic type. Structural modules also can define specializations of, or reuse elements from, domain or other structural modules. When this happens, the structural module becomes dependent.

element domain module
: A vocabulary module that defines one or more specialized element types that can be integrated with maps or topics.

attribute domain module
: A vocabulary module that defines exactly one specialization of either the `@base` or `@props` attribute.

For structural types, the module name is typically the same as the root element. For example, "task" is the name of the structural vocabulary module whose root element is `<task>`.

For element domain modules, the module name is typically a name that reflects the subject domain to which the domain applies, such as "highlight" or "software". Domain modules often have an associated short name, such as "hi-d" for the highlighting domain or "sw-d" for the software domain.

The name (or short name) of an element domain module is used to identify the module in `@class` attribute values. While module names need not be globally unique, module names must be unique within the scope of a given specialization hierarchy. The short name must be a valid XML name token.

Structural modules based on topic *MAY* define additional topic types that are then allowed to occur as subordinate topics within the top-level topic. However, such subordinate topic types *MAY NOT* be used as the root elements of conforming DITA documents.

For example, a top-level topic type might require the use of subordinate topic types that would only ever be meaningful in the context of their containing type and thus would never be candidates for standalone authoring or aggregation using maps. In that case, the subordinate topic type can be declared in the module for the top-level topic type that uses it. However, in most cases, potential subordinate topics are best defined in their own vocabulary modules.

Domain elements intended for use in topics *MUST* ultimately be specialized from elements that are defined in the topic module. Domain elements intended for use in maps *MUST* ultimately be specialized from elements defined by or used in the map module. Maps share some element types with topics but no map-specific elements can be used within topics.

8.3.4 Specialization rules for element types

There are certain rules that apply to element type specializations.

A specialized element type has the following characteristics:

- A properly-formed `@class` attribute that specifies the specialization hierarchy of the element
- A content model that is the same or less inclusive than that of the element from which it was specialized
- A set of attributes that are the same or a subset of those of the element from which it was specialized
- Values or value ranges of attributes that are the same or a subset of those of the element from which it was specialized

DITA elements are never in a namespace. Only the `@DITAArchVersion` attribute is in a DITA-defined namespace. All other attributes, except for those defined by the XML standard, are in no namespace.

This limitation is imposed by the details of the `@class` attribute syntax, which makes it impractical to have namespace-qualified names for either vocabulary modules or individual element types or attributes. Elements included as descendants of the DITA `<foreign>` element type can be in any namespace.

Note Domain modules that are intended for wide use should define element type names that are unlikely to conflict with names used in other domains, for example, by using a domain-specific prefix on all names.

8.3.5 Specialization rules for attributes

There are certain rules that apply to attribute specializations.

A specialized attribute has the following characteristics:

- It is specialized from `@props` or `@base`.
- It can be integrated into a document-type shell either globally, which makes it available on all elements, or it can be assigned to specific elements by using an expansion module.
- It does not have values or value ranges that are more extensive than those of the attribute from which it was specialized.
- Its values must be alphanumeric space-delimited values. In generalized form, the values must conform to the rules for attribute generalization.

8.3.6 @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

Comment by robander
Question about the following statement. If a topic has no integrated attribute domains in 2.0, the value of @specializations will be empty. Is this really a MUST? Should we clarify to indicate that this must be declared in the grammar, but does not necessarily need to have a value - I know that has caused confusion in the past for a tool that reported errors for empty @domains

The root element of every topic and map *MUST* declare the following attributes:

- `@class`
- `@specializations`

Every DITA element (except the `<dita>` element that is used as the root of a ditabase document) *MUST* declare a `@class` attribute.

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.

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 does not need to be generalized.

8.3.7 @specializations attribute rules and syntax

The `@specializations` attribute enables processors to determine what attribute extensions are available in a document. The attribute is declared on the root element for each topic or map type. Each attribute domain defines a token to declare the extension; the effective value of the `@specializations` attribute is composed of these tokens.

Syntax and rules

The `@props` and `@base` attributes are the only two core attributes available for specialization. Each specialization of one of these attributes *MUST* provide a token for use by the `@specializations` attribute.

attribute domains

The `@specializations` token for an attribute specialization begins with either `@props` or `@base` followed by a slash, followed by the name of the new attribute:

```
'@', props-or-base, ('/', attname)+
```

For example:

- If `@props` is specialized to create `@myNewProp`, this results in the following token: `@props/myNewProp`
- If `@base` is specialized to create `@myFirstBase`, this results in the following token: `@base/myFirstBase`
- If that specialized attribute `@myFirstBase` is further specialized to create `@mySecondBase`, this results in the following token: `@base/myFirstBase/mySecondBase`

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
`@deliveryTarget` attribute	deliveryTarget
`@platform` attribute	platform
`@product` attribute	product

The value of the `@specializations` attribute includes one value from each attribute module; the effective value is the following:

```
specializations="@props/deliveryTarget @props/platform @props/product"
```

If the document-type shell also used a specialization of the `@platform` attribute that describes the hardware platform, the new `@hardwarePlatform` attribute domain would add an additional value to the `@specializations` attribute:

```
specializations="@props/deliveryTarget @props/platform @props/platform/hardwarePlatform
@props/product"
```

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

8.3.8 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.

Do not use `<foreign>` or `<unknown>` elements 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
   "
>
```

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.

Comment by robander
The following statement needs to be updated to refer to the eventual final name of the DITA-techcomm package.

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.

8.3.9 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.

8.4 Generalization

Generalization is the process of reversing a specialization. It converts specialized elements or attributes into the original types from which they were derived.

8.4.1 Overview of generalization

Specialized content can be generalized to any ancestor type. The generalization process can preserve information about the former level of specialization to allow round-tripping between specialized and unspecialized forms of the same content.

All DITA documents contain a mix of markup from at least one structural type and zero or more domains. When generalizing the document, any individual structural type or domain can be left as-is, or it can be generalized to any of its ancestors. If the document will be edited or processed in generalized form, it

might be necessary to have a document-type shell that includes all non-generalized modules from the original document-type shell.

Generalization serves several purposes:

- It can be used to migrate content. For example, if a specialization is unsuccessful or is no longer needed, the content can be generalized back to a less specialized form.
- It can be used for temporary round-tripping. For example, if content is shared with a process that is not specialization aware, it can be temporarily generalized for that process and then returned to specialized form.
- It can allow reuse of specialized content in an environment that does not support the specialization. Similar to round-tripping, content can be generalized for sharing, without the need to re-specialize.

When generalizing for migration, the `@class` attribute and `@specializations` attribute need to be absent from the generalized instance document, so that the default values in the document-type shell are used.

When generalizing for round-tripping, the `@class` attribute and `@specializations` attribute *SHOULD* retain the original specialized values in the generalized instance document.

Note that when using constraints, a document instance can always be converted from a constrained document type to an unconstrained document type merely by switching the binding of the document instance to the less restricted document type shell. No renaming of elements is needed to remove constraints.

However, a document whose document-type shell uses expansion modules might not be interchangeable without first generalizing the element and attribute types that were introduced by the expansion modules.

8.4.2 Element generalization

Elements are generalized by examining the `@class` attribute. When a generalization process detects that an element belongs to one of the modules that is being generalized, the element is renamed to a more general form.

For example, the `<step>` element has a `@class` attribute value of `"- topic/li task/step "`. If the task module is being generalized, the `<step>` element is renamed to its more general form from the topic module: `<li>`.

For specific concerns when generalizing structural types with dependencies on non-ancestor modules, see 8.4.5 Generalization with cross-specialization dependencies (154).

While the tag name of a given element is normally the same as the type name of the last token in the `@class` value, this is not required. For example, if a generalization process has already run on the element, the `@class` attribute could contain tokens from two or more modules based on the original specialization. In that case, the element name could already match the first token or an intermediate token in the `@class` attribute. A second generalization process could end up renaming the element again or could leave it alone, depending on the target module or document type.

8.4.3 Processor expectations when generalizing elements

Generalization processors convert elements from one or more modules into their less specialized form. The list of modules can be supplied to a generalization processor, or it can be inferred based on knowledge of a target document-type shell.

The person or application initiating a generalization process can supply the source and target modules for each generalization, for example, "generalize from reference to topic". Multiple target modules can be specified, for example, "generalize from reference to topic and from user-interface domain to topic". When

the source and target modules are not supplied, the generalization process is assumed to be from all structural types to the base (topic or map), and no generalization is performed for domains.

The person or application initiating a generalization process also can supply the target document-type shell. When the target document-type shell is not supplied, the generalized document will not contain a reference to a document-type shell.

A generalization processor *SHOULD* be able to handle cases where it is given:

- Only source modules for generalization (in which case the designated source types are generalized to topic or map)
- Only target modules for generalization (in which case all descendants of each target are generalized to that target)
- Both (in which case only the specified descendants of each target are generalized to that target)

For each structural type instance, the generalization processor checks whether the structural type instance is a candidate for generalization, or whether it has domains that are candidates for generalization. It is important to be selective about which structural type instances to process; if the process simply generalizes every element based on its `@class` attribute values, an instruction to generalize "reference" to "topic" could leave a specialization of reference with an invalid content model, since any elements it reuses from "reference" would have been renamed to topic-level equivalents.

The `@class` attribute for the root element of the structural type is checked before generalizing structural types:

	Source module unspecified	**Source module specified**
Target module unspecified	Generalize this structural type to its base ancestor	Check whether the root element of the topic type matches a specified source module; generalize to its base ancestor if it does, otherwise ignore the structural type instance unless it has domains to generalize.
Target module specified	Check whether the `@class` attribute contains the target module. If it does contain the target, rename the element to the value associated with the target module. Otherwise, ignore the element.	It is an error if the root element matches a specified source but its `@class` attribute does not contain the target. If the root element matches a specified source module and its `@class` attribute does contain the target module, generalize to the target module. Otherwise, ignore the structural type instance unless it has domains to generalize.

For each element in a candidate structural type instance:

	Source module unspecified	**Source module specified**
Target module unspecified	If the `@class` attribute starts with "-" (part of a structural type), rename the element to its base ancestor equivalent. Otherwise ignore it.	Check whether the last value of the `@class` attribute matches a specified source; generalize to its base ancestor if it does, otherwise ignore the element.
Target module specified	Check whether the `@class` attribute contains the target module; rename the element to the value associated with the target module if it does contain the target, otherwise ignore the element.	It is an error if the last value in the `@class` attribute matches a specified source but the previous values do not include the target. If the last value in the `@class` attribute matches a specified source module and the previous values do include the target module, rename the element to the value associated with the target module. Otherwise, ignore the element.

When renaming elements during round-trip generalization, the generalization processor *SHOULD* preserve the values of all attributes. When renaming elements during one-way or migration generalization, the process *SHOULD* preserve the values of all attributes except the `@class` attribute, which is supplied by the target document type.

8.4.4 Attribute generalization

DITA provides a syntax to generalize attributes that have been specialized from the `@props` or `@base` attribute.

Specialization-aware processors *MUST* process both the specialized and generalized forms of an attribute as equivalent in their values.

When a specialized attribute is generalized to an ancestor attribute, the value of the ancestor attribute consists of the name of the specialized attribute followed by its specialized value in parentheses.

For example, if `@jobrole` is an attribute specialized from `@person`, which in turn is specialized from `@props`:

- `jobrole="programmer"` can be generalized to `person="jobrole(programmer)"` or to `props="jobrole(programmer)"`
- `props="jobrole(programmer)"` can be respecialized to `person="jobrole(programmer)"` or to `jobrole="programmer"`

In this example, processors performing generalization and respecialization can use the `@specializations` attribute to determine the ancestry of the specialized `@jobrole` attribute, and therefore the validity of the specialized `@person` attribute as an intermediate target for generalization.

If more than one attribute is generalized, the value of each is separately represented in this way in the value of the ancestor attribute.

Generalized attributes are typically not expected to be authored or edited directly. They are used by processors to preserve the values of the specialized attributes during the time or in the circumstances in which the document is in a generalized form.

A single element *MUST NOT* contain both generalized and specialized values for the same attribute.

For example, the following `<p>` element provides two values for the `@jobrole` attribute, one in a generalized syntax and the other in a specialized syntax:

```
<p person="jobrole(programmer)" jobrole="admin">
    <!-- ... -->
</p>
```

This is an error condition, since it means the document has been only partially generalized, or that the document has been generalized and then edited using a specialized document type.

8.4.5 Generalization with cross-specialization dependencies

Dependencies across specializations limit generalization targets to those that either preserve the dependency or eliminate them. Some generalization targets will not be valid and need to be detected before generalization occurs.

When a structural specialization has a dependency on a domain specialization, then the domain cannot be generalized without also generalizing the reusing structural specialization.

For example, a structural specialization codeConcept might incorporate and require the `<codeblock>` element from the programming domain. A generalization process that turns programming domain elements back into topic elements would convert `<codeblock>` to `<pre>`, making a document that uses

codeConcept invalid. However, codeConcept could be generalized to concept or topic, without generalizing programming domain elements, as long as the target document type includes the programming domain.

When a structural specialization has a dependency on another structural specialization, then both must be generalized together to a common ancestor.

For example, if the task elements in checklist were generalized without also generalizing checklist elements, then the checklist content models that referenced task elements would be broken. And if the checklist elements were generalized to topic without also generalizing the task elements, then the task elements would be out of place, since they cannot be validly present in topic. However, checklist and task can be generalized together to any ancestor they have in common: in this case topic.

When possible, generalizing processes *SHOULD* detect invalid generalization target combinations and report them as errors.

8.5 Constraints

Constraint modules define additional constraints for vocabulary modules in order to restrict content models or attribute lists for specific element types, remove certain extension elements from an integrated domain module, or replace base element types with domain-provided, extension element types.

8.5.1 Overview of constraints

Constraint modules enable information architects to restrict the content models or attributes of OASIS-defined DITA grammars. A constraint is a simplification of an XML grammar such that any instance that conforms to the constrained grammar also will conform to the original grammar.

A constraint module can perform the following functions:

Restrict the content model for an element

Constraint modules can modify content models by removing optional elements, making optional elements required, or requiring unordered elements to occur in a specific sequence. Constraint modules cannot make required elements optional or change the order of element occurrence for ordered elements.

For example, a constraint for `<topic>` can require `<shortdesc>`, can remove `<abstract>`, and can require that the first child of `<body>` be `<p>`. A constraint cannot allow `<shortdesc>` to follow `<prolog>`, because the content model for `<topic>` requires that `<shortdesc>` precedes `<prolog>`.

Restrict the attributes that are available on an element

Constraint modules can restrict the attributes that are available on an element. They also can limit the set of permissible values for an attribute.

For example, a constraint for `<note>` can limit the set of allowed values for the `@type` attribute to "note" and "tip". It also can omit the `@othertype` attribute, since it is needed only when the value of the `@type` attribute is "other".

Restrict the elements that are available in a domain

Constraint modules can restrict the set of extension elements that are provided in a domain. They also can restrict the content models for the extension elements.

For example, a constraint on the programming domain can reduce the list of included extension elements to `<codeph>` and `<codeblock>`.

Replace base elements with domain extensions

Constraint modules can replace base element types with the domain-provided extension elements.

For example, a constraint module can replace the `<ph>` element with the domain-provided elements, making `<ph>` unavailable.

8.5.2 Constraint rules

There are certain rules that apply to the design and implementation of constraints.

Content model

The content model for a constrained element must be at least as restrictive as the unconstrained content model for the element.

The content model and attributes of an element can be constrained by only one constraint module. If two constraint modules exist that constrain the content model or attributes for a specific element, those two modules must be replaced with a new constraint module that reflects the aggregation of the two original constraint modules.

Domain constraints

When a domain module is integrated into a document-type shell, the base domain element can be omitted from the domain extension group or parameter entity. In such a case, there is no separate constraint declaration, because the content model is configured directly in the document-type shell.

A domain module can be constrained by only one constraint module. This means that all restrictions for the extension elements that are defined in the domain must be contained within that one constraint module.

Structural constraints

Each constraint module can constrain elements from only one vocabulary module. For example, a single constraint module that constrains `<refsyn>` from `reference.mod` and constrains `<context>` from `task.mod` is not allowed. This rule maintains granularity of reuse at the module level.

Constraint modules that restrict different elements from within the same vocabulary module can be combined with one another. Such combinations of constraints on a single vocabulary module have no meaningful order or precedence.

8.5.3 Constraints, processing, and interoperability

Because constraints can make optional elements required, documents that use the same vocabulary modules might have incompatible constraints. Thus the use of constraints can affect the ability for content from one topic or map to be used in another topic or map.

A constraint does not change basic or inherited element semantics. The constrained instances remain valid instances of the unconstrained element type, and the element type retains the same semantics and `@class` attribute declaration. Thus, a constraint never creates a new case to which content processing might need to react.

For example, a document type constrained to require the `<shortdesc>` element allows a subset of the possible instances of the unconstrained document type with an optional `<shortdesc>` element. Thus, the content processing for topic still works when `<topic>` is constrained to require a short description.

A constrained document type allows only a subset of the possible instances of the unconstrained document type. Thus, for a processor to determine whether a document instance is compatible with another document type, the document instance *MUST* declare any constraints on the document type.

For example, an unconstrained task is compatible with an unconstrained topic, because the `<task>` element can be generalized to `<topic>`. However, if the topic is constrained to require the `<shortdesc>` element, a document type with an unconstrained task is not compatible with the constrained document type, because some instances of the task might not have a `<shortdesc>` element. However, if the task document type also has been constrained to require the `<shortdesc>` element, it is compatible with the constrained topic document type.

8.5.4 Examples: Constraints implemented using DTDs

8.5.4.1 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 constraint module is named `acme-TopicConstraint.mod`.
2. She adds the following content to `acme-TopicConstraint.mod`:

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

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

8.5.4.2 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                 -->
<!-- ============================================================= -->

<!-- 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.

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

8.5.4.4 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. 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;
                        ">
```

Note Because no other entities are modified or declared outside of the usual "DOMAIN EXTENSIONS" section, this completes the information architect's task. Because no new grammar file or entity is created that would highlight this change, adding a comment to highlight the constraint becomes particularly important (as shown in the example above).

8.5.4.5 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
example-TopicConstraint.mod	<topic>	• Removes <abstract> • Makes <shortdesc> required • Removes <related-links> • Disallows topic nesting
example-SectionConstraint.mod	<section>	• Makes @id required • Removes @spectitle attribute
example-HighlightingDomainConstraint.mod	Highlighting domain	Reduces the highlighting domain elements to <b> and <i>
N/A	<ph>	Remove the <ph> element, allowing only domain extensions (does not require a .mod file)

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 constraints are listed does not matter.

8.5.4.6 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 <tasktroubleshooting> element available in the body of the machinery task.

Comment by robander
Is this example still relevant in DITA 2.0? It can still be valid as an example, but I think earlier examples cover the same type of update, and we no longer need to "fix" the machinery task in 2.0. If it is still necessary, it probably needs to be moved out of the base.

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 % 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.

8.5.5 Examples: Constraints implemented using RNG

8.5.5.1 Example: Constrain a domain using RNG

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>`. Her implementation uses RNG for its grammar files.

When using RNG, domains can be constrained directly in the document-type shells.

1. She opens the document-type shell for topic in an XML editor, and then modifies the "Modules inclusions" section to exclude the elements that she does not want the implementation to use:

```
<div>
  <a:documentation>MODULE INCLUSIONS</a:documentation>
  <include href="topicMod.rng">
    <define name="topic-info-types">
      <ref name="topic.element"/>
    </define>
  </include>
  <include href="audienceAttDomain.rng" dita:since="2.0"/>
  <include href="deliveryTargetAttDomain.rng"/>
  <include href="alternativeTitlesDomain.rng" dita:since="2.0"/>
  <include href="highlightDomain.rng">
    <define name="line-through">
      <notAllowed/>
    </define>
    <define name="overline">
      <notAllowed/>
    </define>
    <define name="sub">
      <notAllowed/>
    </define>
    <define name="sup">
      <notAllowed/>
    </define>
    <define name="tt">
      <notAllowed/>
    </define>
    <define name="u">
      <notAllowed/>
    </define>
  </include>
</div>
```

Note The information architect made an arbitrary choice as to where in the document-type shell she would implement the constraint. It can be placed either in the "Element-type configuration integration" or the "Module inclusions" section.

Comment by Kristen J Eberlein on 03 April 2021

I assume that if someone does this redefintion in the "Element-type configuration integration" section, then the domain module would not be included in the "Module inclusions" section?

2. She makes similar changes to all the other document-type shells in which she wants to constrain the highlighting domain.

8.6 Expansion modules

Expansion modules enable the extension of content models and attribute lists for individual elements. Expansion modules are the opposite of constraints; they add elements and attributes to specific content models and attribute lists, rather than removing them.

8.6.1 Overview of expansion modules

Expansion modules enable information architects to include specialized attributes or elements in specific element types, without making them globally available.

An expansion module can perform the following functions:

Expand content models

Expansion modules extend the content models of specific elements, without making the specialized elements available wherever the specialization base is permitted.

For example, an expansion for `<section>` can make a new element (`<section-shortdesc>`) available as an optional, child element. The `<section-shortdesc>` is specialized from `<p>`, but it is available only within `<section>`.

The elements must be defined in a separate element domain that declares the content model and attributes list for the new elements.

Expand attribute lists

Expansion modules extend the attribute lists of specific elements by adding attributes specialized from either `@base` or `@props`.

For example, an expansion for `<entry>`, `<row>`, and `<colspec>` can make `@cell-purpose` available only on those elements. The `@cell-purpose` attribute is specialized from `@base`. (It could be specialized from `@props`, but the ability to perform filtering is not wanted – and would be problematic if applied on `<entry>` or `<colspec>`.

The additional attributes either can be defined directly within the expansion module, or they can be defined in separate attribute-specialization modules. In either case, the token used as value for the `@specialization` attribute must be defined.

8.6.2 Expansion module rules

There are certain rules that apply to the design and implementation of expansion modules.

Specialization base of expanded elements

Elements added to content models by expansion models must be specializations of existing elements that are permitted in the content model. The elements that are added must only be allowed where their specialization base is allowed.

For example, when creating an expansion model that adds a specialization of `<data>` to `<ol>`, the element must only be allowed before any `<li>` elements, as that is the only place `<data>` is allowed in the content model for ordered lists.

Content model of expanded elements

Expansion modules must not affect the ordinality of the original content model. If the original content model only permits an element to occur once, then the expanded content model cannot break this requirement.

For example, a DITA architect wants to add a new specialization of `<title>` to the `<topic>` element. Since the `<title>` element is only permitted once within a topic, the expansion module must perform one of the following actions:

- Replace `<title>` with the new specialization of title: `<nomarkup-title>`
- Modify the content model of topic to require a choice between `<title>` and the new specialized element

Aggregation of expansion modules

The content model of an element can be modified by either of the following element-configuration modules:

- Constraint module
- Expansion module

The content model of an element only can be modified by a single element-type configuration module. If multiple constraints or extensions need to be applied to a single element, the element configurations must be combined into a single module that reflects all the constraints and expansions that were defined in the original separate modules.

8.6.3 Example: Expansion modules

This section of the specification contains examples and scenarios. They illustrate a variety of ways that expansion modules can be used; they also provide examples of the coding requirements for expansion modules and how expansion modules are integrated into document-type shells.

8.6.3.1 Examples: Expansion implemented using DTDs

8.6.3.1.1 DTD: Adding an attribute to certain table elements

In this scenario, a company makes extensive use of complex tables to present product listings. They occasionally highlight individual cells, rows, or columns for various purposes. The DITA architect wants to implement a semantically meaningful way to identify the purpose of various table elements.

The DITA architect decides to create a new attribute (`@cell-purpose`) and add it to the content model of the following elements:

- `<entry>`
- `<row>`
- `<colspec>`
- `<stentry>`
- `<strow>`

The new attribute will be specialized from `@base`; it will have a small set of tokens that can be values for the new attribute.

The DITA architect decides to integrate the attribute declaration and its assignment to elements into a single expansion module. A more flexible approach would be to separate each `<!ATTLIST` declaration in its own separate expansion module, thus allowing DITA architects who construct document-type shells to decide the elements to which to apply the attribute.

1. First, the DITA architect creates the attribute domain module for the `@cell-purpose` attribute: `acme-cellPurposeAttExpansion.ent`.

```
<!-- Define the attribute -->
<!ENTITY % cellPurposeAtt-d-attribute-expansion
  "cell-purpose  (sale | out-of-stock | new | last-chance | inherit | none)  #IMPLIED"
>

<!-- Declare the entity to be used in the @specializations attribute -->
<!ENTITY cellPurposeAtt-d-att "@base/cell-purpose" >

<!-- Add the attribute to the elements. -->
<!ATTLIST entry %cellPurposeAtt-d-attribute-expansion;>
<!ATTLIST row %cellPurposeAtt-d-attribute-expansion;>
<!ATTLIST colspec %cellPurposeAtt-d-attribute-expansion;>
<!ATTLIST strow %cellPurposeAtt-d-attribute-expansion;>
<!ATTLIST stentry %cellPurposeAtt-d-attribute-expansion;>
```

   Note The attribute definition entity ends in `-expansion`; this indicates that this is an expansion attribute and should not be included in the `%base-attribute-extensions;` entity in the document-type shell.

2. The DITA architect integrates this module into the document-type shell.

```
<!-- ============================================================= -->
<!--              DOMAIN ATTRIBUTES DECLARATIONS                   -->
<!-- ============================================================= -->

<!ENTITY % cellPurposeAttExpansion-d-dec
  PUBLIC "-//ACME//ENTITIES DITA Cell Purpose Attribute Expansion//EN"
         "cellPurposeAttExpansion.ent"
>%cellPurposeAttExpansion-d-dec;
```

3. After updating the `catalog.xml` file to include the new expansion module, the work is done.

8.6.3.1.2 DTD: Adding an element to the <section> element

In this scenario, a DITA architect wants to modify the content model for the `<section>` element. He wants to add an optional `<section-shortdesc>` element that is specialized from `<p>`.

To accomplish this, the DITA architect needs to create the following modules and integrate them into the document-type shell:

- An element specialization module that defines the `<section-shortdesc>` element
- An expansion module that adds the `<section-shortdesc>` element to the content model for `<section>`

1. First, the DITA architect creates the entity declaration file: `sectionShortdescDomain.ent`

```
<!ENTITY sectionShortdesc-d-p-expansion "section-shortdesc">

<!ENTITY % section-shortdesc "section-shortdesc">
```

   Note This entity file declares a `%sectionShortdesc-d-p-expansion;` entity, not `%sectionShortdesc-d-p;`, indicating that the expansion nature of this domain. The content author wants to control exactly where the element is allowed; he does not want to allow it everywhere that `<p>` is allowed.

2. Then the DITA architect creates the `.mod` file that defines the content model and attributes for `<section-shortdesc>`: `sectionShortdescDomain.mod`

```
<!ENTITY % section-shortdesc.content "(%para.cnt;)*">
<!ENTITY % section-shortdesc.attributes "%univ-atts;">

<!ELEMENT section-shortdesc %section-shortdesc.content;>
<!ATTLIST section-shortdesc %section-shortdesc.attributes;>
```

```
<!ATTLIST section-shortdesc    class CDATA "+ topic/p sectionShortdesc-d/section-
shortdesc">
```

3. Next, the content architect modifies the document-type shell to integrate the domain modules:

```
<!-- ============================================================= -->
<!--              DOMAIN ENTITY DECLARATIONS                       -->
<!-- ============================================================= -->

<!-- ... other domains ... -->

<!ENTITY % sectionShortdesc-d-dec
  PUBLIC "-//ACME//ENTITIES DITA Section Short Description Domain//EN"
         "sectionShortdescDomain.ent"
>%sectionShortdesc-d-dec;

<!-- ... other DTD content ... -->

<!-- ============================================================= -->
<!--                     DOMAIN ELEMENT INTEGRATION                -->
<!-- ============================================================= -->

<!-- ... other domains ... -->

<!ENTITY % sectionShortdesc-d-def
  PUBLIC "-//ACME//ELEMENTS DITA Section Short Description Domain//EN"
         "sectionShortdescDomain.mod"
>%sectionShortdesc-d-def;
```

At this point, the new domain is integrated into the topic document-type shell. However, the new element is not added to the content model for `<section>`.

4. Next, the DITA architect creates an expansion module that adds the `<section-shortdesc>` element to the content model of `<section>`: `acme-SectionExpansion.mod`

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

<!ENTITY % dl "dl">
<!ENTITY % div "div">
<!ENTITY % fig "fig">
<!ENTITY % image "image">
<!ENTITY % lines "lines">
<!ENTITY % lq "lq">
<!ENTITY % note "note">
<!ENTITY % object "object">
<!ENTITY % ol "ol">
<!ENTITY % p "p">
<!ENTITY % pre "pre">
<!ENTITY % simpletable "simpletable">
<!ENTITY % sl "sl">
<!ENTITY % table "table">
<!ENTITY % ul "ul">
<!ENTITY % cite "cite">
<!ENTITY % include "include">
<!ENTITY % keyword "keyword">
<!ENTITY % ph "ph">
<!ENTITY % q "q">
<!ENTITY % term "term">
<!ENTITY % text "text">
<!ENTITY % tm "tm">
<!ENTITY % xref "xref">
<!ENTITY % state "state">
<!ENTITY % data "data">
<!ENTITY % data-about "data-about">
<!ENTITY % foreign "foreign">
<!ENTITY % unknown "unknown">
<!ENTITY % sectiondiv "sectiondiv">
<!ENTITY % title "title">
<!ENTITY % draft-comment "draft-comment">
<!ENTITY % fn "fn">
```

```
<!ENTITY % indexterm "indexterm">
<!ENTITY % required-cleanup "required-cleanup">

<!-- Defines the modified content model for <section>.       -->

<!ENTITY % section.content
              "(#PCDATA |
               %dl; |
               %div; |
               %fig; |
               %image; |
               %lines; |
               %lq; |
               %note; |
               %object; |
               %ol; |
               %p; |
               %pre; |
               %simpletable; |
               %sl; |
               %table; |
               %ul; |
               %cite; |
               %include; |
               %keyword; |
               %ph; |
               %q; |
               %term; |
               %text; |
               %tm; |
               %xref; |
               %state; |
               %data; |
               %data-about; |
               %foreign; |
               %unknown; |
               %sectiondiv; |
               %title; |
               %draft-comment; |
               %fn; |
               %indexterm; |
               %required-cleanup; |
               %sectionShortdesc-d-p-expansion;)*"
>
```

Note The DITA architect needed to explicitly declare all the elements, rather than using the parameter entity used in the definition of `<section>`: `%section.cnt;`. Because the element-configuration modules are integrated into the document-type shell before the base grammar modules, none of the parameter entities that used in the base DITA vocabulary modules are available.

5. Finally, the DITA architect integrates the expansion module into the document-type shell:

```
<!-- ============================================================= -->
<!--                    ELEMENT-TYPE CONFIGURATION INTEGRATION              -->
<!-- ============================================================= -->

<!ENTITY % acmeSection-def
  PUBLIC "-//ACME//ELEMENTS DITA Section Mix-in//EN"
         "acme-SectionMixin.mod"
>%acmeSection-def;
```

6. After updating the `catalog.xml` file to include the new domain modules and the expansion module, the work is done.

8.6.3.1.3 DTD: Aggregating constraint and expansion modules

With the new specialization rules introduced in DITA 2.0, DITA architect wants to add certain extension modules to a document-type shell.

The following table lists the constraints that are currently integrated into the document-type shell.

File name	What it constrains	Details
example-TopicConstraint.mod	<topic>	• Removes <abstract> • Makes <shortdesc> required • Removes <related-links> • Disallows topic nesting
example-SectionConstraint.mod	<section>	• Makes @id required • Removes @spectitle attribute
example-HighlightingDomainConstraint.mod	Highlighting domain	Reduces the highlighting domain elements to <b> and <i>
N/A	<ph>	Remove the <ph> element, allowing only domain extensions (does not require a .mod file)

The following table lists the expansion modules that the DITA architect wants to add to the document-type shell.

File name	What it modifies	Details
example-sectionSectionShortdescExpansion.mod	<section>	Adds an optional <section-shortdesc> element to <section>.
example-dlentryModeAttExpansion.ent	<dlentry>	Adds @dlentry-mode to the attributes of <dlentry>.

The constraint and expansion modules that target the <section> element must be combined into a single element-configuration module.

Comment by Kristen J Eberlein on 28 March 2021

Need to add full example here.

8.6.3.2 Examples: Expansion implemented using RNG

8.6.3.2.1 RNG: Adding an element to the <section> element

In this scenario, a DITA architect wants to modify the content model for the <section> element. He wants to add an optional <section-shortdesc> element that is specialized from <p>.

To accomplish this, the DITA architect needs to create the following modules and integrate them into the document-type shell:

- An element specialization module that defines the <section-shortdesc> element
- An expansion module that adds the <section-shortdesc> element to the content model for <section>

Comment by Kristen J Eberlein on 20 April 2021

Is this true? Or will this simply be done in the document-type shell by redefining the content model for <section>?

1. First, the DITA architect creates the element-specialization module: X

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="urn:oasis:names:tc:dita:rng:vocabularyModuleDesc.rng"
                          schematypens="http://relaxng.org/ns/structure/1.0"?>
<grammar xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  xmlns:dita="http://dita.oasis-open.org/architecture/2005/"
  xmlns="http://relaxng.org/ns/structure/1.0">
  <!--<div>
    <a:documentation>DOMAIN EXTENSION PATTERNS</a:documentation>

  </div>-->
  <div>
    <a:documentation>ELEMENT TYPE NAME PATTERNS</a:documentation>
    <define name="section-shortdesc">
      <ref name="section-shortdesc.element"/>
    </define>
  </div>

  <div>
    <a:documentation>ELEMENT TYPE DECLARATIONS</a:documentation>
    <div>
      <a:documentation>LONG NAME: Section short description</a:documentation>
      <define name="section-shortdesc.content">
        <zeroOrMore>
            <ref name="para.cnt"/>
          </zeroOrMore>
      </define>
      <define name="section-shortdesc.attributes">
        <ref name="univ-atts"/>
      </define>
    </div>
  </div>
</grammar>
```

2. How does the new element get added to the content model for <section>? Where does the redefinition happen?
3. Integrate the element-specialization module into the doctype shell.
4. Update catalog.xml file.

8.6.3.2.2 RNG: Adding an attribute to certain table elements

In this scenario, a company makes extensive use of complex tables to present product listings. They occasionally highlight individual cells, rows, or columns for various purposes. The DITA architect wants to implement a semantically meaningful way to identify the purpose of various table elements.

The DITA architect decides to create a new attribute (@cell-purpose) and add it to the content model of the following elements:

- <entry>
- <row>
- <colspec>
- <stentry>
- <strow>

The new attribute will be specialized from @base; it will have a small set of tokens that can be values for the new attribute.

1. The DITA architect creates an attribute-domain specialization module: `acme-cellPurposeAttExpansion.rng`. It contains the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="urn:oasis:names:tc:dita:rng:vocabularyModuleDesc.rng"
                          schematypens="http://relaxng.org/ns/structure/1.0"?>
<grammar
  xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  xmlns:dita="http://dita.oasis-open.org/architecture/2005/"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

  <define name="cellPurposeAtt-d-attribute-expansion">
    <optional>
      <attribute name="cellPurpose">
        <a:documentation>Specifies the purpose of the table cell. This is a specialized
                          attribute for Acme Corporation.
        </a:documentation>
        <choice>
          <value>sale</value>
          <value>out-of-stock</value>
          <value>new</value>
          <value>last-chance</value>
          <value>inherit</value>
          <value>none</value>
        </choice>
      </attribute>
    </optional>
  </define>

  <define name="base-attribute-extensions" combine="interleave">
    <ref name="cellPurposeAtt-d-attribute-expansion"/>
  </define>

</grammar>
```

Comment by Kristen J Eberlein on 20 April 2021

Where does the application of this new attribute to only certain elements take place?

2. Integrate expansion module into the doc-type shell.
3. Update `catalog.xml` file.

8.6.3.2.3 RNG: Aggregating constraint and expansion modules

9 Coding practices for DITA grammar files

This section collects all of the rules for creating modular DTD, RELAX NG, or XML Schema grammar files to represent DITA document types, specializations, and constraints.

9.1 Recognized XML-document grammar mechanisms

The DITA standard recognizes the following XML-document grammar mechanisms by which conforming DITA vocabulary modules and document types can be constructed: document type declarations (DTDs), XML Schema declarations (XSDs), and RELAX NG grammars. Conforming DITA document types and vocabulary modules can be constructed using several XML document-grammar mechanisms. The DITA specification provides coding requirements for DTDs and RNG; it also includes grammar files constructed using those mechanisms.

This specification defines implementation requirements for all of these document grammar mechanisms. The OASIS DITA Technical Committee recognizes that other XML grammar languages might provide similar modularity and extensibility mechanisms. However, because the Technical Committee has not yet defined implementation requirements for those languages, their conformance cannot be determined.

Of these document-grammar mechanisms, RELAX NG grammars offer the easiest-to-use syntax and the most precise constraints. For this reason, the RELAX NG definitions of the standard DITA vocabularies are the normative versions. The DTD and XSD versions shipped by OASIS are generated from the RELAX NG version using open source tools.

Related information

Tools for generating DTD or XSD from RELAX NG

9.2 Normative versions of DITA grammar files

The OASIS DITA Technical Committee uses the RELAX NG XML syntax for the normative versions of the XML grammar files that comprise the DITA release.

The DITA Technical Committee chose the RELAX NG XML syntax for the following reasons:

Easy use of foreign markup

The DITA grammar files maintained by OASIS depend on this feature of RELAX NG in order to capture metadata about document-type shells and modules; such metadata is used to generate the DTD- and XSD-based versions of the grammar files.

The foreign vocabulary feature also can be used to include Schematron rules directly in RELAX NG grammars. Schematron rules can check for patterns that either are not expressible with RELAX NG directly or that would be difficult to express.

RELAX NG element

This general grouping element allows for arbitrary organization and grouping of patterns within grammar documents. Such grouping tends to make the grammar documents easier to work with, especially in XML-aware editors. The use or non-use of the RELAX NG `<div>` element does not affect the meaning of the patterns that are defined in a RELAX NG schema.

Capability of expressing precise restrictions

RELAX NG is capable of expressing constraints that are more precise than is possible with either DTDs or XSDs. For example, RELAX NG patterns can be context specific such that the same element type can allow different content or attributes in different contexts.

If you plan to generate DTD- or XSD-based modules from RELAX NG modules, avoid RELAX NG features that cannot be translated into DTD or XSD constructs. When RELAX NG is used directly for DITA document validation, the document-type shells for those documents can integrate constraint modules that use the full power of RELAX NG to enforce constraints that cannot be enforced by DTDs or XSDs. The grammar files provided by the OASIS DITA Technical Committee do not use any features of RELAX NG that cannot be translated into equivalent DTD or XSD constructs.

Comment by Kristen J Eberlein on 31 March 2021

Regarding the above paragraph, do we want to remove the mentions of XSD? The advice is still sound, although I think we want to encourage people who must use XSDs to generate a single file for validation from either DTD or RNG.

The DITA use of RELAX NG depends on the *RELAX NG DTD Compatibility* specification, which provides a mechanism for defining default attribute values and embedded documentation. Processors that use RELAX NG for DITA documents in which required attributes (for example, `@class` attribute) are not explicitly present must implement the DTD compatibility specification in order to get default attribute values.

Related information

Tools for generating DTD or XSD from RELAX NG

9.3 DTD coding requirements

This section explains how to implement DTD based document-type shells, specializations, and element-configuration modules (constraint and expansion).

9.3.1 DTD: Use of entities

DITA-based DTDs use entities to implement specialization and element configuration. Therefore, an understanding of entities is critical when working with DTD-based document-type shells, vocabulary modules, or element-configuration modules (constraint and expansion).

Entities can be defined multiple times within a single document type, but only the first definition is effective. How entities work shapes DTD coding practices. The following list describes a few of the more important entities that are used in DITA DTDs:

Elements defined as entities

Every element in a DITA DTD is defined as an entity. When elements are added to a content model, they are added using the entity. This enables extension with domain specializations.

For example, the entity `%ph;` usually just means the `<ph>` element, but it can be defined in a document-type shell to mean "`<ph>` plus the elements from the highlighting domain". Because the document-type shell places that entity definition before the usual definition, every element that includes `%ph;` in its content model now includes `<ph>` plus every element in the highlighting domain that is specialized from `<ph>`.

Content models defined as entities

Every element in a DITA DTD defines its content model using an entity. This enables element configuration.

For example, rather than directly setting what is allowed in `<ph>`, that element sets its content model to `%ph.content;`; that entity defines the actual content model. A constraint module can redefine the

`%ph.content;` model to remove selected elements, or an expansion module can redefine the `%ph.content;` module to add elements..

Attribute sets defined as entities

Every element in a DITA DTD defines its attributes using an entity. This enables element configuration.

For example, rather than directly defining attributes for `<ph>`, that element sets its attributes using the `%ph.attributes;` entity; that entity defines the actual attributes. A constraint module can remove attributes from the attribute list, or an expansion module can add attributes to the attribute list.

Note When constructing an element-configuration module or document-type shell, new entities are usually viewed as "redefinitions" because they redefine entities that already exist. However, these new definitions only work because they are added to a document-type shell before the existing definitions. Most topics about DITA DTDs, including others in this specification, describe these overrides as redefinitions to ease understanding.

9.3.2 DTD: Coding requirements for document-type shells

A DTD-based document-type shell is organized into sections; each section contains entity declarations that follow specific coding rules.

The DTD-based approach to configuration, specialization, and element configuration (constraint and expansion) relies heavily upon parameter entities. Several of the parameter entities that are declared in document-type shells contain references to other parameter entities. Because parameter entities must be declared before they are used, the order of the sections in a DTD-based document-type shell is significant.

A DTD-based document-type shell contains the following sections:

1. Topic [or map] entity declarations (172)
2. Domain constraint integration (173)
3. Domain entity declarations (173)
4. Domain attributes declarations (173)
5. Domain extensions (174)
6. Domain attribute extensions (174)
7. Topic nesting override (174)
8. Specializations attribute override (174)
9. Element-type configuration integration (175)
10. Topic [or map] element integration (175)
11. Domain element integration (176)

Each of the sections in a DTD-based document-type shell follows a pattern. These patterns help ensure that the shell follows XML parsing rules for DTDs; they also establish a modular design that simplifies creation of new document-type shells.

Topic [or map] entity declarations

This section declares and references an external parameter entity for each of the following items:

- The top-level topic or map type that the document-type shell configures
- Any additional structural modules that are used by the document type shell

The parameter entity is named *`type-name`*`-dec`.

In the following example, the <concept> specialization is integrated into a document-type shell:

```
<!-- ============================================================= -->
<!--              TOPIC ENTITY DECLARATIONS                        -->
<!-- ============================================================= -->

<!ENTITY % concept-dec
  PUBLIC "-//OASIS//ENTITIES DITA 2.0 Concept//EN"
         "concept.ent"
>%concept-dec;
```

Domain constraint integration

This section declares and references an external parameter entity for each domain-constraint module that is integrated into the document-type shell.

The parameter entity is named *descriptorDomainName*-c-dec.

In the following example, the entity file for a constraint module that reduces the highlighting domain to a subset is integrated in a document-type shell:

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

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

Domain entity declarations

This section declares and references an external parameter entity for each element-domain module that is integrated into the document-type shell.

The parameter entity is named *shortDomainName*-dec.

In the following example, the entity file for the highlighting domain is included in a document-type shell:

```
<!-- ============================================================= -->
<!--              DOMAIN ENTITY DECLARATIONS                       -->
<!-- ============================================================= -->

<!ENTITY % hi-d-dec PUBLIC
    "-//OASIS//ENTITIES DITA 2.0 Highlight Domain//EN"
    "highlightDomain.ent"
>%hi-d-dec;
```

Domain attributes declarations

This section declares and references an external parameter entity for each attribute domain that is integrated into the document-type shell.

The parameter entity is named *domainName*-dec.

In the following example, the entity file for the @deliveryTarget attribute domain is included in a document-type shell:

```
<!-- ============================================================= -->
<!--              DOMAIN ATTRIBUTES DECLARATIONS                   -->
<!-- ============================================================= -->

<!ENTITY % deliveryTargetAtt-d-dec
  PUBLIC "-//OASIS//ENTITIES DITA 2.0 Delivery Target Attribute Domain//EN"
```

```
         "deliveryTargetAttDomain.ent"
>%deliveryTargetAtt-d-dec;
```

Domain extensions

This section declares and references a parameter entity for each element that is extended by one or more domain modules. These entities are used by later modules to define content models; redefining the entity adds domain specializations wherever the base element is allowed.

In the following example, the entity for the `<pre>` element is redefined to add specializations from the programming, software, and user interface domains:

```
<!-- ============================================================= -->
<!--                    DOMAIN EXTENSIONS                          -->
<!-- ============================================================= -->

<!ENTITY % pre
    "pre        |
     %pr-d-pre; |
     %sw-d-pre; |
     %ui-d-pre;">
```

Domain attribute extensions

This section redefines the parameter entity for each attribute domain that is integrated globally into the document-type shell. The redefinition adds an extension to the parameter entity for the relevant attribute.

In the following example, the parameter entities for the `@base` and `@props` attributes are redefined to include the `@newfrombase`, `@othernewfrombase`, `@new`, and `@othernew` attributes:

```
<!-- ============================================================= -->
<!--                    DOMAIN ATTRIBUTE EXTENSIONS                -->
<!-- ============================================================= -->

<!ENTITY % base-attribute-extensions
        "%newfrombaseAtt-d-attribute;
         %othernewfrombaseAtt-d-attribute;">

<!ENTITY % props-attribute-extensions
        "%newAtt-d-attribute;
         %othernewAtt-d-attribute;">
```

Topic nesting override

This section redefines the entity that controls topic nesting for each topic type that is integrated into the document-type shell.

The parameter entity is named *topictype*`-info-types`.

The definition usually is an "OR" list of the topic types that can be nested in the parent topic type. Use the literal root-element name, not the corresponding parameter entity. Topic nesting can be disallowed completely by specifying the `<no-topic-nesting>` element.

In the following example, the parameter entity specifies that `<concept>` can nest any number of `<concept>` or `<myTopicType>` topics, in any order:

```
<!-- ============================================================= -->
<!--                    TOPIC NESTING OVERRIDE                     -->
<!-- ============================================================= -->

<!ENTITY % concept-info-types "concept | myTopicType">
```

Specializations attribute override

This section redefines the `included-domains` entity to include the text entity for each attribute domain that is either included or reused in the document-type shell. The redefinition sets the effective

value of the `@specializations` attribute for the top-level document type that is configured by the document-type shell.

Comment by Kristen J Eberlein on 10 April 2021

What is the difference between "including an attribute domain" and "reusing an attribute domain"? What distinction where we trying to make here?

In the following example, parameter entities are included for the DITA conditional-processing attributes:

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

<!ENTITY included-domains
                          "&audienceAtt-d-att;
                           &deliveryTargetAtt-d-att;
                           &otherpropsAtt-d-att;
                           &platformAtt-d-att;
                           &productAtt-d-att;"
>
```

Element-type configuration integration

This section declares and references the parameter entity for each element-configuration module (constraint and expansion) that is integrated into the document-type shell

The parameter entity is named *descriptionElement*`-c-def`.

In the following example, the module that constrains the content model for the `<taskbody>` element is integrated into the document-type shell for strict task:

```
<!ENTITY % strictTaskbody-c-def
  PUBLIC "-//OASIS//ELEMENTS DITA 2.0 Strict Taskbody Constraint//EN"
  "strictTaskbodyConstraint.mod"
>%strictTaskbody-c-def;
```

Topic [or map] element integration

This section declares and references a parameter entity for each structural module that is integrated into the document-type shell.

The parameter entity is named *structuralType*`-type`.

The structural modules are included in ancestry order, so that the parameter entities that are used in an ancestor module are available for use in specializations. When a structural module depends on elements from a vocabulary module that is not part of its ancestry, the module upon which the structural module has a dependency (and any ancestor modules not already included) need to be included before the module with a dependency.

In the following example, the structural modules required by the troubleshooting topic are integrated into the document-type shell:

```
<!-- ============================================================= -->
<!--                     TOPIC ELEMENT INTEGRATION                 -->
<!-- ============================================================= -->

<!ENTITY % topic-type
  PUBLIC "-//OASIS//ELEMENTS DITA 2.0 Topic//EN"
         "../../base/dtd/topic.mod"
>%topic-type;

<!ENTITY % task-type
```

```
  PUBLIC "-//OASIS//ELEMENTS DITA 2.0 Task//EN"
         "task.mod"
>%task-type;

<!ENTITY % troubleshooting-type
  PUBLIC "-//OASIS//ELEMENTS DITA 2.0 Troubleshooting//EN"
         "troubleshooting.mod"
>%troubleshooting-type;
```

Domain element integration

This section declares and references a parameter entity for each element domain that is integrated tinto the document-type shell.

The parameter entity is named *domainName*`-def`.

In the following example, the element-definition file for the highlighting domain is integrated into the document-type shell:

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

<!ENTITY % hi-d-def PUBLIC
    "-//OASIS//ELEMENTS DITA 2.0 Highlight Domain//EN"
    "highlightDomain.mod"
>%hi-d-def;
```

If a structural module depends on a domain, the domain module needs to be included before the structural module. This erases the boundary between the final two sections of the DTD-based document-type shell, but it is necessary to ensure that modules are embedded after their dependencies. Technically, the only solid requirement is that both domain and structural modules be declared after all other modules that they specialize from or depend on.

9.3.3 DTD: Coding requirements for element-type declarations

This topic covers general coding requirements for defining element types in both structural and element-domain vocabulary modules.

Module files

A vocabulary module that defines a structural or element-domain specialization is composed of two files:

- A definition module (`.mod`) file, which declares the element names, content models, and attribute lists for the element types that are defined in the vocabulary module
- An entity declaration (`.ent`) file, which declares the text and parameter entities that are used to integrate the vocabulary module into a document-type shell

Element definitions

A structural or element-domain vocabulary module contains a declaration for each element type that is named by the module. While the XML standard allows content models to refer to undeclared element types, the DITA standard does not permit this. All element types or attribute lists that are named within a vocabulary module are declared in one of the following objects:

- The vocabulary module
- A base module of which the vocabulary module is a direct or indirect specialization
- (If the vocabulary module is a structural module) A required domain module

The following components make up a single element definition in a DITA DTD-based vocabulary module.

Element name entities

For each element type, there is a parameter entity with a name that matches the element-type name. The default value is the element-type name.

For example:

```
<!ENTITY % topichead "topichead">
```

The parameter entity provides a layer of abstraction when setting up content models; it can be redefined in a document-type shell in order to create domain extensions or implement element configuration (constraint and expansion).

Element name entities for a single vocabulary module typically are grouped together at the top of the vocabulary module. They can occur in any order.

Content-model parameter entity

For each element type, there is a parameter entity that defines the content model. The name of the parameter entity is *tagname*`.content`, and the value is the content model definition.

For example:

```
<!ENTITY % topichead.content
  "((%topicmeta;)?,
    (%anchor; |
     %data.elements.incl; |
     %navref; |
     %topicref;)*)
">
```

Attribute-list parameter entity

For each element type, there is a parameter entity that declares the attributes that are available on the element. The name of the parameter entity is *tagname*`.attributes`, and the value is a list of the attributes that are used by the element type (except for `@class`).

For example:

```
<!ENTITY % topichead.attributes
 "keys CDATA #IMPLIED
  copy-to CDATA #IMPLIED
  %topicref-atts;
  %univ-atts;"
>
```

Consistent use and naming of the *tagname*`.content` parameter entity enables the use of element-configuration modules (constraint and expansion) to redefine the content model.

Element declaration

For each element type, there is an element declaration that consists of a reference to the content-model parameter entity.

For example:

```
<!ELEMENT topichead    %topichead.content;>
```

Attribute list declaration

For each element type, there is an attribute list declaration that consists of a reference to the attribute-list parameter entity.

For example:

```
<!ATTLIST topichead    %topichead.attributes;>
```

Specialization attribute declarations

A vocabulary module defines a `@class` attribute for every element that is declared in the module. The value of the attribute is constructed according to the rules in 8.3.6 class attribute rules and syntax (147).

For example, the `ATTLIST` definition for the `<topichead>` element (a specialization of the `<topicref>` element in the base map type) includes the definition of the `@class` attribute, as follows:

```
<!ATTLIST topichead  class CDATA "+ map/topicref  mapgroup-d/topichead ">
```

Definition of the <topichead> element

The following code sample shows how the `<topichead>` element is defined in `mapGroup.mod`. Ellipses indicate where the code sample has been snipped for brevity.

```
<!-- ============================================================= -->
<!--                    ELEMENT NAME ENTITIES                      -->
<!-- ============================================================= -->

<!ENTITY % topichead      "topichead"                                  >

...

<!-- ============================================================= -->
<!--                     ELEMENT DECLARATIONS                      -->
<!-- ============================================================= -->

<!--                     LONG NAME: Topichead                      -->
<!ENTITY % topichead.content
                        "((%topicmeta;)?,
                          (%anchor; |
                           %data.elements.incl; |
                           %navref; |
                           %topicref;)*)"
>
<!ENTITY % topichead.attributes
              "keys
                           CDATA
                                     #IMPLIED
               copy-to
                           CDATA
                                     #IMPLIED
               %topicref-atts;
               %univ-atts;"
>
<!ELEMENT  topichead %topichead.content;>
<!ATTLIST  topichead %topichead.attributes;>

...

<!-- ============================================================= -->
<!--             SPECIALIZATION ATTRIBUTE DECLARATIONS             -->
<!-- ============================================================= -->

...

<!ATTLIST  topichead      class CDATA "+ map/topicref mapgroup-d/topichead ">

<!-- ================== End of DITA Map Group Domain ================== -->
```

9.3.4 DTD: Coding requirements for structural modules

This topic covers general coding requirements for DTD-based structural modules.

Required topic and map element attributes

The topic or map element type sets the `@DITAArchVersion` attribute to the version of DITA in use, typically by referencing the `arch-atts` parameter entity. It also sets the `@specializations` attribute to the `included-domains` entity.

The `@DITAArchVersion` and `@specializations` attributes give processors a reliable way to check the architecture version and look up the list of attribute domains that are available in the document type.

The following example shows how the `@DITAArchVersion` and `@specializations` attributes are defined for the `<concept>` element in DITA 2.0. Ellipses indicate where the code is snipped for brevity:

```
<!-- ============================================================= -->
<!--                    ELEMENT DECLARATIONS                       -->
<!-- ============================================================= -->

...

<!--                    LONG NAME: Concept                         -->

...

<!ATTLIST concept
  %concept.attributes;
  %arch-atts;
  specializations  CDATA  "&included-domains;">
```

Controlling nesting in topic types

Specialized topics typically use a parameter entity to define what topic types are permitted to nest. While there are known exceptions described below, the following rules apply when using parameter entities to control nesting.

Comment by Kristen J Eberlein on 07 April 2021

What are the exceptions, and are they indeed described below?

Parameter entity name

The name of the parameter entity is the topic element name plus the `-info-types` suffix.

For example, the name of the parameter entity for the concept topic is `concept-info-types`.

Parameter entity value

To set up default topic nesting rules, set the entity to the desired topic elements. The default topic nesting is used when a document-type shell does not set up different rules.

For example, the following parameter entity sets up default nesting so that `<concept>` will nest only other `<concept>` topics:

```
<!-- ============================================================= -->
<!--                    ELEMENT DECLARATIONS                       -->
<!-- ============================================================= -->

<!ENTITY % concept-info-types
             "%info-types;"
>
```

As an additional example, the following parameter entity sets up a default that will not allow any nesting:

```
<!ENTITY % glossentry-info-types "no-topic-nesting">
```

Default topic nesting in a structural module often is set up to use the `%info-types;` parameter entity rather than using a specific element. When this is done consistently, a shell that includes multiple structural modules can set common nesting rules for all topic types by setting `%info-types;` entity.

The following example shows a structural module that uses `%info-types;` for default topic nesting:

```
<!ENTITY % concept-info-types "%info-types;">
```

Content model of the root element

The last position in the content model defined for the root element of a topic type *SHOULD* be the *`topictype`*`-info-types` parameter entity.

A document-type shell then can control how topics are allowed to nest for this specific topic type by redefining the *`topictype`*`-info-types` entity for each topic type. If default nesting rules reference the `info-types` parameter entity, a shell can efficiently create common nesting rules by redefining the `info-types` entity.

For example, with the following content model defined for `<concept>`, a document-type shell that uses the concept specialization can control which topics are nested in `<concept>` by redefining the `concept-info-types` parameter entity:

```
<!ENTITY % concept.content
  "((%title;),
    (%titlealts;)?,
    (%abstract; | %shortdesc;)?,
    (%prolog;)?,
    (%conbody;)?,
    (%related-links;)?,
    (%concept-info-types;)*)"
>
```

In certain cases, you do not need use `info-types` parameter entity to control topic nesting:

- If you want a specialized topic type to never allow any nested topics, regardless of context, it can be defined without any entity or any nested topics.
- If you want a specialized topic type to only allow specific nesting patterns, such as allowing only other topic types that are defined in the same module, it can nest those topics directly in the same way that other nested elements are defined.

Comment by Kristen J Eberlein on 07 April 2021

Examples? I think the wording needs to be crisped up.

9.3.5 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.

For example, the name of the parameter entity for the highlight domain that extends the `<ph>` element is `hi-d-ph`.

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 is separated by the vertical line (|) symbol.

For example, the value of the `%hi-d-ph;` parameter entity is `"b | u | i | line-through | overline | tt | sup | sub"`.

Example

The following code sample shows the parameter entity for the highlight domain, as declared in `highlightDomain.ent`:

```
<!-- ============================================================= -->
<!--                    ELEMENT EXTENSION ENTITY DECLARATIONS      -->
<!-- ============================================================= -->

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

<!-- ================ End DITA Highlight Domain ================== -->
```

9.3.6 DTD: Coding requirements for attribute domain modules

The vocabulary modules that define attribute domains have additional coding requirements. The module must include a parameter entity for the new attribute, which can be referenced in document-type shells, as well as a text entity that specifies the contribution to the `@specializations` attribute for the attribute domain.

The name of an attribute domain is the name of the attribute plus "Att". For example, for the attribute named `@deliveryTarget`, the attribute-domain name is "deliveryTargetAtt". The attribute-domain name is used to construct entity names for the domain.

Parameter entity name and value

The name of the parameter entity is the attribute domain name, followed by the literal `-d-attribute`. The value of the parameter entity is a DTD declaration for the attribute.

Text entity name and value

The text entity name is the attribute domain name, followed by the literal `-d-Att`. The value of the text entity is the `@specializations` attribute contribution for the module; see 8.3.7 specializations attribute rules and syntax (149) for details on how to construct this value.

Example

The `@deliveryTarget` attribute can be defined in a vocabulary module using the following two entities.

```
<!ENTITY % deliveryTargetAtt-d-attribute
  "deliveryTarget  CDATA  #IMPLIED"
>

<!ENTITY deliveryTargetAtt-d-att "@props/deliveryTarget" >
```

9.3.7 DTD: Coding requirements for constraint modules

An element-type constraint module defines the constraints for a map or topic element type. A domain constraint module defines the constraints for an element or attribute domain.

Element-type constraint modules

Element-type constraint modules have the following requirements:

The *tagname*.attributes parameter entity

When the attribute set for an element is constrained, there must be a declaration of the *tagname*.attributes parameter entity that defines the constrained attributes.

For example, the following parameter entity defines a constrained set of attributes for the <note> element that removes most of the values defined for @type; it also removes @spectitle and @othertype:

```
<!ENTITY % note.attributes
       "type  (attention | caution | note ) #IMPLIED
        %univ-atts;">
```

The *tagname*.content parameter entity

When the content model for an element is constrained, there must be a declaration of the *tagname*.content parameter entity that defines the constrained content model.

For example, the following parameter entity defines a more restricted content model for <topic>, in which the <shortdesc> element is required.

```
<!ENTITY % topic.content

  "((%title;),
    (%titlealts;)?,
    (%shortdesc;),
    (%prolog;)?,
    (%body;)?,
    (%topic-info-types;)*)"
>
```

Domain constraint modules

Domain constraint modules have the following requirements:

Parameter entity

When the set of extension elements are restricted, there must be a parameter entity that defines the constrained content model.

For example, the following parameter entity restricts the highlighting domain to <b> and <i>:

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

Constraining to replace a base element with domain extensions

When element domains are used to extend a base element, those extensions can be used to replace the base element. This form of constraint is done inside the document-type shell.

Within a document-type shell, domain extensions (174) are implemented by declaring an entity for a base element. The value of the entity can omit any base element types from which the other element types that are listed are specialized.

In the following example, the `<pre>` base type is removed from the entity declaration, effectively allowing only specializations of `<pre>` but not `<pre>` itself.

```
<!ENTITY % pre
    "%pr-d-pre; |
     %sw-d-pre; |
     %ui-d-pre;">
```

9.3.8 DTD: Coding requirements for expansion modules

An expansion module defines the expanded configuration for a map or topic element type.

Expansion modules

Expansion modules have the following coding requirements:

The *tagname*.attributes parameter entity

The *tagname*.attributes parameter entity declares the expanded attributes.

This parameter entity can be defined in an attribute specialization module, or it can be defined directly in the expansion module.

For example, the following parameter entity defines a new attribute intended for use with various table elements:

```
<!ENTITY % cellPurposeAtt-d-attribute-expansion
  "cell-purpose  (sale | out-of-stock | new | last-chance | inherit | none)  #IMPLIED"
>
```

Note that the name of the parameter entity ends with `-expansion`; this indicates that this is an expansion attribute and should not be included in the `%base-attribute-extensions;` entity in the document-type shell.

Comment by Kristen J Eberlein on 08 April 2021

Should we call this *tagname*.attributes parameter entity? Or would *Domain*.attributes parameter entity be more appropriate?

The *tagname*.content parameter entity

The *tagname*.content parameter entity declares the expanded content model for the element. This redefinition of the content model references the parameter entity that was defined in the element specialization module.

For example, the following code sample shows the entity declaration file for an element specialization module that defines a `<section-shortdesc>` element, which is intended to be added to the content model of `<section>`:

```
<!ENTITY sectionShortdesc-d-p-expansion "section-shortdesc">

<!ENTITY % section-shortdesc "section-shortdesc">
```

When the content model for <section> is redefined in the expansion module, it references the parameter entity defined in the entities file for the element specialization:

```
<!ENTITY % section.content
              "(#PCDATA |
               %dl; |
               %div; |
               %fig; |
               %image; |
               %note; |
               %ol; |
               %p; |
               %simpletable; |
               %ul; |
               %sectiondiv; |
               %title; |
               %draft-comment;|
               %sectionShortdesc-d-p-expansion;)*"
>
```

Note that this expansion module also constrains the content model of <section> to only include certain block elements.

9.4 RELAX NG coding requirements

This section explains how to implement RNG based document-type shells, specializations, and element-configuration modules (constraints and expansions).

9.4.1 RELAX NG: Overview of coding requirements

RELAX NG modules are self-integrating; they automatically add to the content models and attribute lists that they extend. This means that information architects do not have much work to do when integrating vocabulary modules and element-configuration modules (constraints and expansion) into document-type shells.

In addition to simplifying document-type shells, the self-integrating aspect of RELAX NG results in the following coding practices:

- Each specialization module consists of a single file, unlike the two required for DTDs.
- Domain modules directly extend elements, unlike DTDs, which rely on an extra file and extensions within the document-type shell.
- Constraint modules directly include the modules that they extend, which means that just by referencing a constraint module, the document-type shell gets everything it needs to both define and constrain a vocabulary module.

Comment by Kristen J Eberlein on 02 April 2021

What do we need to say about extension modules?

RELAX NG grammars for DITA document-type shells, vocabulary modules, and element-configuration modules (constraint and expansion) *MAY* do the following:

- Use the <a:documentation> element anywhere that foreign elements are allowed by RELAX NG. The <a:documentation> element refers to the <documentation> element type from the http://relaxng.org/ns/compatibility/annotations/1.0 as defined by the DTD compatibility specification. The prefix "a" is used by convention.
- Use to group pattern declarations.

- Include embedded Schematron rules or any other foreign vocabulary. Processors *MAY* ignore any foreign vocabularies within DITA grammars that are not in the `http://relaxng.org/ns/compatibility/annotations/1.0` or `http://dita.oasis-open.org/architecture/2005/` namespaces.

Syntax for RELAX NG grammars

The RELAX NG specification defines two syntaxes for RELAX NG grammars: the XML syntax and the compact syntax. The two syntaxes are functionally equivalent, and either syntax can be reliably converted into the other by using, for example, the open-source Trang tool.

DITA practitioners can author DITA modules using one RELAX NG syntax, and then use tools to generate modules in the other syntax. The resulting RELAX NG modules are conforming if there is a one-to-one file correspondence.

Conforming RELAX NG-based DITA modules *MAY* omit the annotations and foreign elements that are used in the OASIS grammar files to enable generation of other XML grammars, such as DTDs and XML Schema. When such annotations are used, conversion from one RELAX NG syntax to the other might lose the information, as processors are not required to process the annotations and information from foreign vocabularies.

The DITA coding requirements are defined for the RELAX NG XML syntax. Document type shells, vocabulary modules, and element-configuration modules (constraints and expansion) that use the RELAX NG compact syntax can use the same organization requirements as those defined for the RELAX NG XML syntax.

Comment by Kristen J Eberlein on 02 April 2021

I have a problem with having a single section with a title. Either other content in this topic needs to be grouped into at titled section, or the original section needs to be removed.

9.4.2 RELAX NG: Coding requirements for document-type shells

A RNG-based document-type shell is organized into sections; each section follows a pattern. These patterns help ensure that the shell follows XML parsing rules for RELAX NG; they also establish a modular design that simplifies creation of new document-type shells.

Because RELAX NG modules are self-integrating, RNG-based document-type shells only need to include vocabulary modules and element-configuration modules (constraints and expansion).

An RNG-based document-type shell contains the following sections:

1. Root element declaration (185)
2. specializations attribute (186)
3. Element-type configuration integration (186)
4. Module inclusions (186)
5. ID-defining element overrides (187)

Root element declaration

Document-type shells use the RELAX NG start declaration to specify the root element of the document type. The `<start>` element defines the root element, using a reference to a *tagname*`.element` pattern.

For example:

```
<div>
  <a:documentation>ROOT ELEMENT DECLARATION</a:documentation>
  <start combine="choice">
    <ref name="topic.element"/>
  </start>
</div>
```

@specializations attribute

This section lists the tokens that attribute-domain and element-configuration modules contribute to the `@specialization` attribute.

For example:

```
<div>
  <a:documentation>SPECIALIZATIONS ATTRIBUTE</a:documentation>
  <define name="specializations-att">
    <optional>
      <attribute name="specializations"
                 a:defaultValue="@props/audience
                                 @props/deliveryTarget
                                 @props/otherprops
                                 @props/platform
                                 @props/product"
      />
    </optional>
  </define>
</div>
```

Element-type configuration integration

This section of the document-type shell contains includes for element-type configuration modules (constraints and expansions). Because the element-configuration module imports the module that it override, any module that is configured in this section (including the base topic or map modules) is left out of the following "Module inclusion" section.

For example, the following code sample shows the section of an RNG-based document-type shell that redefines the `<taskbody>` element to create the strict task topic.

```
<div>
<a:documentation>ELEMENT-TYPE CONFIGURATION INTEGRATION</a:documentation>
  <include href="strictTaskbodyConstraintMod.rng">
    <define name="task-info-types">
      <ref name="task.element"/>
    </define>
  </include>
</div>
```

Module inclusions

This section of the RNG-based document-type shell includes all unconstrained domain or structural modules.

For example:

```
  <div>
    <a:documentation>MODULE INCLUSIONS</a:documentation>
    <include href="topicMod.rng">
      <define name="topic-info-types">
        <ref name="topic.element"/>
      </define>
    </include>
    <include href="audienceAttDomain.rng" dita:since="2.0"/>
    <include href="deliveryTargetAttDomain.rng"/>
    <include href="otherpropsAttDomain.rng" dita:since="2.0"/>
```

```
    <include href="platformAttDomain.rng" dita:since="2.0"/>
    <include href="productAttDomain.rng" dita:since="2.0"/>
    <include href="alternativeTitlesDomain.rng" dita:since="2.0"/>
    <include href="emphasisDomain.rng" dita:since="2.0"/>
    <include href="hazardstatementDomain.rng"/>
    <include href="highlightDomain.rng"/>
    <include href="utilitiesDomain.rng"/>
  </div>
```

ID-defining element overrides

This section declares any element in the document type that uses an `@id` attribute with an XML data type of "ID". This declaration is required in order to prevent RELAX NG parsers from issuing errors.

If the document-type shell includes domains for foreign vocabularies such as SVG or MathML, this section also includes exclusions for the namespaces used by those domains.

For example, the following code sample is from an RNG-based document-type shell for a task topic. It declares that both the `<topic>` and `<task>` elements have an `@id` attribute with a XML data type of ID. These elements and any elements in the SVG or MathML namespaces are excluded from the "any" pattern by being placed within the `<except>` element:

```
<div>
    <a:documentation> ID-DEFINING-ELEMENT OVERRIDES </a:documentation>
    <define name="any">
       <zeroOrMore>
          <choice>
             <ref name="idElements"/>
             <element>
                <anyName>
                   <except>
                      <name>topic</name>
                      <name>task</name>
                      <nsName ns="http://www.w3.org/2000/svg"/>
                      <nsName ns="http://www.w3.org/1998/Math/MathML"/>
                   </except>
                </anyName>
                <zeroOrMore>
                   <attribute>
                      <anyName/>
                   </attribute>
                </zeroOrMore>
                <ref name="any"/>
             </element>
             <text/>
          </choice>
       </zeroOrMore>
    </define>
</div>
```

9.4.3 RELAX NG: Coding requirements for element-type declarations

This topic covers general coding requirements for element types in structural and element-domain vocabulary modules.

Module files

Each RELAX NG vocabulary module consists of a single module file.

Element definitions

A structural or element-domain vocabulary module contains a declaration for each element type that is named in the module. While the XML standard allows content models to refer to undeclared element

types, the DITA standard does not permit it. All element types or attribute lists that are named in a vocabulary module are declared in one of the following objects:

- The vocabulary module
- A base module of which the vocabulary module is a direct or indirect specialization
- (If the vocabulary module is a structural module) A required domain or structural module

The element type patterns are organized into the following sections:

Element type name patterns

For each element type that is declared in the vocabulary module, there is a pattern whose name is the element type name and whose content is a reference to the element-type *tagname*`.element` pattern.

For example:

```
<div>
  <a:documentation>ELEMENT TYPE NAME PATTERNS</a:documentation>
  <define name="b">
    <ref name="b.element"/>
  </define>
  <!-- ... -->
</div>
```

The element-type name pattern provides a layer of abstraction that facilitates redefinition. The element-type name patterns are referenced from content model and domain extension patterns. Specialization modules can re-declare the patterns to include specializations of the type, allowing the specialized types in all contexts where the base type is allowed.

The declarations can occur in any order.

Common content-model patterns

Structural and element-domain modules can include a section that defines the patterns that contribute to the content models of the element types that are defined in the module.

Common attribute sets

Structural and element-domain modules can include a section that defines patterns for attribute sets that are common to one or more of the element types that are defined in the module.

Element type declarations

For each element type that is declared in the vocabulary module, the following set of patterns are used to define the content model and attributes for the element type. Each set of patterns typically is grouped within a `<div>` element.

***tagname*.content**

Defines the complete content model for the element *tagname*. The content model pattern can be overridden in element-configuration modules (constraints and expansion).

***tagname*.attributes**

Defines the complete attribute list for the element *tagname*, except for `@class`. The attribute list declaration can be overridden in element-configuration modules (constraints and expansion).

***tagname*.element**

Provides the actual element-type definition. It contains an `<element>` element whose `@name` value is the element type name and whose content is a reference to the *tagname*`.content` and *tagname*`.attlist` patterns. In OASIS grammar files, the `@longName` attribute in the DITA architecture namespace is also used to help enable generation of DTD and XSD grammar files.

Comment by Kristen J Eberlein on 07 April 2021

Are we removing the stuff for generating DTD and XSD from the RNG?

***tagname*.attlist**

An additional attribute-list pattern with a `@combine` attribute set to the value "interleave". This pattern contains only a reference to the *tagname*`.attributes` pattern.

The following example shows the declaration for the `<topichead>` element, including the definition for each pattern described above.

```
    <div>
      <a:documentation>Topic Head</a:documentation>
      <define name="topichead.content">
        <optional>
          <ref name="topicmeta"/>
        </optional>
        <zeroOrMore>
          <choice>
            <ref name="anchor"/>
            <ref name="data.elements.incl"/>
            <ref name="navref"/>
            <ref name="topicref"/>
          </choice>
        </zeroOrMore>
      </define>
      <define name="topichead.attributes">
        <optional>
          <attribute name="keys"/>
        </optional>
        <optional>
          <attribute name="copy-to"/>
        </optional>
        <ref name="topicref-atts"/>
        <ref name="univ-atts"/>
      </define>
      <define name="topichead.element">
        <element name="topichead">
          <a:documentation>The &lt;topichead&gt; element provides a title-only entry in a
navigation map, as an alternative to the fully-linked title provided by the &lt;topicref&gt;
element. Category:
            Mapgroup elements</a:documentation>
          <ref name="topichead.attlist"/>
          <ref name="topichead.content"/>
        </element>
      </define>
      <define name="topichead.attlist" combine="interleave">
        <ref name="topichead.attributes"/>
      </define>

    </div>
```

Comment by robander
Reminder to update this example with a 2.0 version of the declaration.

idElements pattern contribution

Element types that declare the `@id` attribute as type "ID", including all topic and map element types, provide a declaration for the `idElements` pattern. This is required to correctly configure the "any" pattern override in document-type shells and avoid errors from RELAX NG parsers. The declaration is specified with a `@combine` attribute set to the value "choice".

For example:

```
<div>
  <a:documentation>LONG NAME: Map</a:documentation>
  <!-- ... -->
  <define name="idElements" combine="choice">
    <ref name="map.element"/>
```

```
  </define>
</div>
```

Specialization attribute declarations

A vocabulary module must define a `@class` attribute for every specialized element. This is done in a section at the end of each module that includes a *tagname*`.attlist` pattern for each element type that is defined in the module. The declarations can occur in any order.

The *tagname*`.attlist` pattern for each element defines that element's `@class` attribute. `@class` is declared as an optional attribute; the default value is declared using the `@a:defaultValue` attribute, and the value of the attribute is constructed according to the rules in 8.3.6 class attribute rules and syntax (147).

For example:

```
<define name="anchorref.attlist" combine="interleave">
  <optional>
    <attribute name="class"
        a:defaultValue="+ map/topicref mapgroup-d/anchorref "
    />
  </optional>
</define>
```

9.4.4 RELAX NG: Coding requirements for structural modules

A structural vocabulary module defines a new topic or map type as a specialization of a topic or map type.

Required topic and map element attributes

The topic or map element type references the `arch-atts` pattern, which defines the `@DITAArchVersion` attribute in the DITA architecture namespace and sets the attribute to the version of DITA. In addition, the topic or map element type references the `specializations-att` pattern, which pulls in a definition for the `@specializations` attribute.

For example, the following definition references the `arch-atts` and `specializations-att` patterns as part of the definition for the `<concept>` element.

```
<div>
  <a:documentation> LONG NAME: Concept </a:documentation>
  <!-- ... -->
  <define name="concept.attlist" combine="interleave">
    <ref name="concept.attributes"/>
    <ref name="arch-atts"/>
    <ref name="specializations-att"/>
  </define>
  <!-- ... -->
</div>
```

The `@DITAArchVersion` and `@specialization` attributes give processors a reliable way to check the DITA version and the attribute domains that are used.

Controlling nesting in topic types

Specialized topics typically define an `info-types` style pattern to specify default topic nesting. Document-type shells then can control how topics are allowed to nest by redefining the pattern. While there are known exceptions described below, the following rules apply when using a pattern to control topic nesting.

Comment by Kristen J Eberlein on 07 April 2021

What are the exceptions, and are they indeed described below?

Pattern name

The pattern name is the topic element name plus the suffix `-info-types`.

For example, the info-types pattern for the concept topic type is `concept-info-types`.

Pattern value

To set up default topic-nesting rules, set the pattern to the desired topic elements. The default topic nesting is used when a document-type shell does not set up different rules.

For example:

```
<div>
  <a:documentation>INFO TYPES PATTERNS</a:documentation>
  <define name="mytopic-info-types">
    <ref name="subtopictype-01.element"/>
    <ref name="subtopictype-02.element"/>
  </define>
  <!-- ... -->
</div>
```

To disable topic nesting, specify the `<empty>` element.

For example:

```
<define name="learningAssessment-info-types">
  <empty/>
</define>
```

The `info-types` pattern also can be used to refer to common nesting rules across the document-type shell.

Comment by Kristen J Eberlein on 07 April 2021

What does the above sentence mean?

For example:

```
<div>
  <a:documentation>INFO TYPES PATTERNS</a:documentation>
  <define name="mytopic-info-types">
    <ref name="info-types"/>
  </define>
  <!-- ... -->
</div>
```

Content model of the root element

In the declaration of the root element of a topic type, the last position in the content model is the *topictype*`-info-types` pattern.

For example, the `<concept>` element places the pattern after `<related-links>`:

```
<div>
  <a:documentation>LONG NAME: Concept</a:documentation>
  <define name="concept.content">
    <!-- ... -->
    <optional>
      <ref name="related-links"/>
    </optional>
    <zeroOrMore>
```

```
      <ref name="concept-info-types"/>
    </zeroOrMore>
  </define>
</div>
```

In certain cases, you do not need to use the `info-types` pattern to control topic nesting:

- If a topic type will never permit topic nesting, regardless of context, it can be defined without any pattern or any nested topics.
- If a topic type will only allow specific nesting patterns, such as allowing only other topic types that are defined in the same module, it can nest those topics directly in the same way that other nested elements are defined.

Comment by Kristen J Eberlein on 07 April 2021

Examples? I think the wording needs to be crisped up.

9.4.5 RELAX NG: Coding requirements for element-domain modules

Element-domain modules declare an extension pattern for each element that is extended by the domain. These patterns are used when including the domain module in document-type shells.

Pattern name

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

For example, the name of the pattern for the highlight domain that extends the `<ph>` element is `hi-d-ph`.

Pattern definition

The pattern consists of a choice group that contains references to element-type name patterns. Each extension of the base element type is referenced.

For example:

```
<a:documentation>DOMAIN EXTENSION PATTERNS</a:documentation>

<define name="hi-d-ph">
  <choice>
    <ref name="b.element"/>
    <ref name="i.element"/>
    <ref name="line-through.element"/>
    <ref name="overline.element"/>
    <ref name="sup.element"/>
    <ref name="sub.element"/>
    <ref name="tt.element"/>
    <ref name="u.element"/>
  </choice>
</define>
```

Extension pattern

For each element type that is extended by the element-domain module, the module extends the element-type pattern with a `@combine` value of "choice" that contains a reference to the domain pattern.

For example, the following pattern adds the highlight domain specializations of the `<ph>` element to the content model of the `<ph>` element:

```
<define name="ph" combine="choice">
  <ref name="hi-d-ph"/>
</define>
```

Because the pattern uses a `@combine` value of "choice", the effect is that the domain-provided elements automatically are added to the effective content model of the extended element in any grammar that includes the domain module.

Example

The following code sample shows the extension pattern for the highlight domain, as declared in `highlightDomain.rng`:

```
<div>
  <a:documentation>DOMAIN EXTENSION PATTERNS</a:documentation>

  <define name="hi-d-ph">
    <choice>
      <ref name="b.element"/>
      <ref name="i.element"/>
      <ref name="line-through.element"/>
      <ref name="overline.element"/>
      <ref name="sup.element"/>
      <ref name="sub.element"/>
      <ref name="tt.element"/>
      <ref name="u.element"/>
    </choice>
  </define>

  <define name="ph" combine="choice">
    <ref name="hi-d-ph"/>
  </define>
</div>
```

9.4.6 RELAX NG: Coding requirements for attribute-domain modules

An attribute-domain vocabulary module declares a new attribute specialized from either the `@props` or `@base` attribute.

The name of an attribute domain is the name of the attribute plus "Att". For example, for the attribute named `@deliveryTarget`, the attribute-domain name is "deliveryTargetAtt". The attribute-domain name is used to construct pattern names for the domain.

An attribute-domain module consists of a single file, which has three sections:

Specializations attribute contribution

The contribution to the `@specializations` attribute is documented in the module. The value is constructed according to the rules found in 8.3.7 specializations attribute rules and syntax (149).

The OASIS grammar files use a `<domainsContribution>` element to document the contribution; this element is used to help enable generation of DTD and XSD grammar files. An XML comment or `<a:documentation>` element also can be used.

Attribute declaration pattern

The specialized attribute is declared in a pattern named *domainName*`-d-attribute`. The attribute is defined as optional.

For example, the following code samples shows the the `@audience` specialization of `@props`:

```
<define name="audienceAtt-d-attribute">
  <optional>
    <attribute name="audience" dita:since="2.0">
      <a:documentation>Specifies the audience to which an element applies.</
a:documentation>
    </attribute>
```

```
  </optional>
</define>
```

Attribute extension pattern

The attribute extension pattern extends either the `@props` or `@base` attribute set pattern to include the attribute specialization.

Specializations of @props

The pattern is named `props-attribute-extensions`. The pattern specifies a `@combine` value of "interleave", and the content of the pattern is a reference to the specialized-attribute declaration pattern.

For example:

```
<define name="props-attribute-extensions" combine="interleave">
  <ref name="audienceAtt-d-attribute"/>
</define>
```

Specializations of @base

The pattern is named `base-attribute-extensions`. The pattern specifies a `@combine` value of "interleave", and the content of the pattern is a reference to the specialized-attribute declaration pattern.

For example:

```
<define name="base-attribute-extensions" combine="interleave">
    <ref name="myBaseSpecializationAtt-d-attribute"/>
</define>
```

9.4.7 RELAX NG: Coding requirements for constraint modules

A structural constraint module defines the constraints for a map or topic element type. A domain constraint module defines the constraints for an element or attribute domain.

Implementation of constraint modules

Constraint modules are implemented by importing the constraint module into a document-type shell in place of the module that the constraint modifies. The constraint module itself imports the base module to be constrained; within the import, the module redefines the patterns as needed to implement the constraint.

For example, a constraint module that modifies the `<section>` element imports the base module `topicMod.rng`. Within that import, it constrains the `section.content` pattern:

```
<include href="topicMod.rng">
  <define name="section.content">
    <!-- Define constrained model here -->
  </define>
</include>
```

For a more complete example, see `strictTaskbodyConstraintMod.rng`, delivered with the DITA 1.3 grammar files.

Comment by robander
Need to update the above example statement for 2.0 to refer to the correct name of the tech comm 2.0 spec.

Combining multiple constraints

Because the constraint module imports the module that it modifies, only one constraint module can be used per vocabulary module (otherwise the module being constrained would be imported multiple times). If multiple constraints are combined for a single vocabulary module, they must be implemented in one of the following ways:

Combing constraints into a single module

The constraints can be combined into a single module.

For example, when combining separate constraints for `<section>` and `<shortdesc>`, a single module can be defined as follows:

```
<include href="topicMod.rng">
  <define name="section.content">
    <!-- Constrained model for section -->
  </define>
  <define name="shortdesc.content">
    <!-- Constrained model for shortdesc -->
  </define>
</include>
```

Chaining constraints

Constraints can be chained so that each constraint imports another, until the final constraint imports the base vocabulary module.

For example, when combining separate constraints for `<section>`, `<shortdesc>`, and `<li>` from the base vocabulary, the `<section>` constraint can import the `<shortdesc>` constraint, which in turn imports the `<li>` constraint, which finally imports `topicMod.rng`.

Comment by Kristen J Eberlein on 07 April 2021

Need an example of this

9.4.8 RNG: Coding requirements for expansion modules

An expansion module defines the expanded configuration for a map or topic element type.

Expansion modules

Expansion modules have the following coding requirements:

Comment by Kristen J Eberlein on 20 April 2021

What needs to go here?

10 Element reference

This section of the DITA specification contains a topic for each DITA element, as well as information about DITA attributes.

10.1 DITA elements, A to Z

This topics provides links to all of the DITA elements in alphabetical order.

10.2 DITA attributes, A to Z

This topics provides links to DITA attributes in alphabetical order.

10.3 Topic elements

The base topic elements include elements that make up the core building blocks of the DITA topic, such as topic, body, and related-links, as well as elements like <p> and <ph> that are used in many topic specializations. Some of these elements are also available inside the <topicmeta> map element.

10.3.1 Basic topic elements

The generic topic structure is used for untyped topics.

For an answer to the question "What are topics?" and more details on when to use different information types, see 3.3 DITA topics (19).

10.3.1.1 <abstract>

The <abstract> element occurs between the topic title and the topic body. It is presented as the initial content of a topic. The <abstract> can contain paragraph-level content as well as one or more <shortdesc> elements which can be used for providing link previews or summaries.

Usage information

The <abstract> element cannot be overridden by maps, but its contained <shortdesc> elements can be, for the purpose of creating link summaries or previews.

Use the <abstract> element when the initial paragraph of a topic is unsuitable for use as a link preview or for summaries, because, for example, it contains lists or tables, or because only a portion of the paragraph is suitable. Note that when the initial paragraph is suitable as a summary, that content should be placed in a <shortdesc> element rather than in an <abstract> element. The <abstract> element allows for a wider range of content in your initial paragraph, such as lists and tables, and allows you to identify portions of the <abstract> content as useful for previews or summaries by embedding the <shortdesc> element within <abstract>.

Processing expectations

When the contained <shortdesc> occurs within phrase-level content, it is treated as phrase-level content and should not create a separate paragraph on output of the topic. When the contained <shortdesc> occurs as a peer to paragraph-level content, it is treated as block-level content and should create a separate paragraph on output of the topic. When multiple <shortdesc> elements are included in an <abstract>, they are concatenated in output of link previews or summaries (separated by spaces).

When a <shortdesc> element occurs in a DITA map, it overrides the short description provided in the topic for the purpose of generating link previews, but does not replace the <shortdesc> in the rendered topic itself. This means that generated links to this topic will use the short description from the map for purposes any link previews provided with the link, while the rendered topic continues to use the short description inside the topic. If the <topicref> element in the DITA map also specifies the @copy-to attribute, the content of the <shortdesc> element in the DITA map also overrides the short description provided in the topic. In this case, the rendered topic itself will display the <shortdesc> contents from the map in place of the <shortdesc> originally specified in the topic. Processors might not implement this behavior.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example: <abstract> with phrase-level short description

```
<abstract>The abstract is being used to provide more complex content.
    <shortdesc>The short description must be directly contained by the abstract.</shortdesc>
The abstract can put text around the shortdesc.
</abstract>
```

Topic output

The abstract is being used to provide more complex content. The short description must be directly contained by the abstract. The abstract can put text around the short description.

Preview/summary output

The short description must be directly contained by the abstract.

Example: <abstract> with block-level short description

```
<abstract><p>The abstract is being used to provide more complex content.</p>
    <shortdesc>The short description must be directly contained by the abstract.</shortdesc>
<p>The abstract can put text around the short description.</p>
</abstract>
```

Topic output

The abstract is being used to provide more complex content.

The short description must be directly contained by the abstract.

The abstract can put text around the short description.

Preview/summary output

The short description must be directly contained by the abstract.

Example: <abstract> with multiple short descriptions

```
<abstract>The abstract is being used to provide more complex content.
    <shortdesc>The short description must be directly contained by the abstract.</shortdesc>
    <p>The abstract can put text around the short description.</p>
    <shortdesc>There can be more than one short description.</shortdesc>
</abstract>
```

Topic output

The abstract is being used to provide more complex content. The short description must be directly contained by the abstract.

The abstract can put text around the short description.

There can be more than one short description.

Preview/summary output

The short description must be directly contained by the abstract. There can be more than one short description.

Related reference

shortdesc (206)

A short description describes the purpose or main point of a topic.

10.3.1.2 <body>

The body contains the main content of a topic.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows a DITA topic that contains a title and a body.

```
<topic>
  <title>Mycompany Style Guide: the <xmlelement>b</xmlelement> element</title>
  <body><p>Use the bold tag <b>for visual emphasis only</b>; do not use it if another
phrase-level element better signifies the reason for the emphasis.</p></body>
</topic>
```

10.3.1.3 <bodydiv>

A body division is a grouping of sequential elements within the body of a topic. There is no addition semantic meaning. It is useful primarily as a specialization base and for reuse.

Usage information

The `<bodydiv>` element cannot contain a title. If a title is required, use nested topics.

The `<bodydiv>` element can nest itself, so it can be used as a specialization base. Another common use case for the `<bodydiv>` element is to group a sequence of related elements for reuse, so that another topic can reference the entire set with a single `@conref` or `@conkeyref` attribute.

Because the `<bodydiv>` element allows `<section>`, it cannot be used within `<section>` elements. Use the `<div>` element to group content that might occur in both topic bodies and sections.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows how the `<bodydiv>` element can be used to group a sequence of elements for reuse:

```
<topic id="sample" xml:lang="en">
 <title>Sample for bodydiv</title>
 <body>
  <bodydiv id="div">
   <p>This set of information is reusable as a group.</p>
   <p>Lists of three contain three items.</p>
   <ul>
    <li>This is one item.</li>
    <li>This is another item.</li>
    <li>This is the third item.</li>
   </ul>
  </bodydiv>
  <p>This concludes my topic.</p>
```

```
  </body>
</topic>
```

10.3.1.4 <dita>

The <dita> element is the root element for the ditabase document type.

Comment by Kristen J Eberlein on 09 September 2020

This topic should be removed from the base spec and added to the tech comm spec. It should include info about use cases (and maybe when to use ditabase, and when to NOT use ditabase.)

Usage information

The <dita> element can contain any sequence of topics, and the ditabase document type enables these topic types to nest.

The <dita> element cannot be specialized.

Attributes

The following attributes are available on this element: @xmlns:ditaarch and @DITAArchVersion from Architectural attributes (369), and Localization attributes (366).

Example

The following code sample shows a ditabase document that contains multiple topics:

```
<dita>
  <concept id="batintro">
    <title>Intro to bats</title>
    <conbody>
      <!-- ... -->
    </conbody>
  </concept>
  <task id="batfeeding">
    <title>Feeding a bat</title>
    <taskbody>
      <!-- ... -->
    </taskbody>
  </task>
  <reference id="batparts">
    <title>Parts of bats</title>
    <refbody>
      <!-- ... -->
    </refbody>
  </reference>
      <!-- ... -->
</dita>
```

10.3.1.5 <related-links>

The (<related-links> element contains …. This element follows the body of the topic.

Rendering expectations

After a topic is processed into its final output form, the related links usually are displayed at the end of the topic, although some Web-based help systems might display them in a separate navigation frame.

Links specified within the <related-links> element typically are displayed together with the links that are generated based on the map context.

Processing expectations

The following lists contains processing expectations for the <related-link> element:

1. Links within a <linklist> element appear in the order defined, while those outside of a <linklist> might be sorted and displayed in a different order or location (based upon their role, target, importance, or other qualifiers).
2. PDF or print-oriented output typically ignores hierarchical links such as those with roles of ancestor, parent, child, descendant, next, previous, or sibling, although this behavior is not required.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (apart from @href), and 10.8.3.10 The role and otherrole attributes (390).

Example

The following code sample show how the <related-link> element is used to specify links to Web sites that always will be relevant to the topic:

```
<related-links scope="external" format="html">
  <link href="http://www.example.org">
    <linktext>Example 1</linktext>
  </link>
  <link href="http://www.example.com">
    <linktext>Example 2</linktext>
  </link>
</related-links>
```

10.3.1.6 <shortdesc>

A short description describes the purpose or main point of a topic.

Usage information

When present in topics, the short description is the first paragraph of the topic. It also is used for link previews and search results.

When present in topics, the short description is the first paragraph of the topic. It also is used for link previews and search results.

When present in maps, the <shortdesc> element is associated with <topicref> elements. This enables map authors to accomplish the following goals:

- Associate a short description with a non-DITA object.
- Provide a short description that is specific to the map context and used for link previews.
- Override the short description located in the associated DITA topic, when the @copy-to attribute is specified. Processors might not implement this behavior.

  When a <shortdesc> element applies to an entire DITA map, it serves as description only.

Rendering expectations

Processors *SHOULD* render the content of the <shortdesc> element as the initial paragraph of the topic.

When processors generate link previews that are based on the map context, they *SHOULD* use the content of the `<shortdesc>` that is located in the map rather than the `<shortdesc>` that is located in the DITA topic. However, when processors render the topic itself, they *SHOULD* use the content of the `<shortdesc>` element located in the DITA topic, unless the `@copy-to` attribute is specified on the topic reference in the map.

The short description in the map *MAY* override the short description in the topic if the `<topicref>` element specifies a `@copy-to` attribute.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Examples

Figure 66: Short description in a topic

The following code sample shows how a `<shortdesc>` element can be used in a topic:

```
<topic id="concept">
 <title>Introduction to bird calling</title>
 <shortdesc>If you want to attract more birds to your Acme Bird Feeder,
learn the art of bird calling. Bird calling is an efficient way
to alert more birds to the presence of your bird feeder.</shortdesc>
 <body>
   <p>Bird calling requires learning:</p>
   <ul>
    <li>Popular and classical bird songs</li>
    <li>How to whistle like a bird</li>
   </ul>
 </body>
</topic>
```

Figure 67: Short description in a map

The following code sample shows how a short description can be used in a DITA map to provide information about a non-DITA resource. The content of the `<shortdesc>` element is used when a link preview to the Web site for the American Birding Association is generated.

```
<map>
  <title>Enjoying birds</title>
    ...
    <topicref href="birds-in-colorado.dita"/>
    <topicref href="bird-calling.dita"/>
    <topicref href="https://www.birding.example.com/" format="external" type="html">
      <topicmeta>
        <shortdesc>The American Birding Association is only organization
        in North America that specifically caters to recreational birders.
        Its mission is to "inspire all people to enjoy and protect wild birds."
      </topicmeta>
    </topicref>
    ...
</map>
```

Related reference

abstract (202)

The <abstract> element occurs between the topic title and the topic body. It is presented as the initial content of a topic. The <abstract> can contain paragraph-level content as well as one or more <shortdesc> elements which can be used for providing link previews or summaries.

10.3.1.7 <title>

A title is a heading or label for an object. Titles can be associated with topics, maps, sections, examples, figures, tables, and other structures.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (without the metadata attribute group), plus @base and @rev from the Metadata attributes (366).

Example

The following code sample shows how titles are used for both the topic and a figure within the topic:

```
<topic id="topicid">
  <title>Monitoring your heart rate with ThisDevice</title>
  <body>
    <!-- ... -->
    <fig id="adjust-the-monitor">
      <title>Adjusting your monitor</title>
      <p>If the monitor is not reporting, follow the directions
         in the video to adjust your equipment.</p>
      <!-- ... -->
    </fig>
  </body>
</topic>
```

10.3.1.8 <titlealt>

An alternative title is used to convey information about a document in contexts other than straightforward display.

Usage Information

Alternative titles can be used in topics and in maps. When used directly beneath a root <map> element, the alternative title applies to the map itself. When used inside a <topicref> or specialization of <topicref>, the alternative title applies to the resource referenced by the <topicref>. When the referenced resource is a DITA topic, the alternative titles from the <topicref> are merged with those authored directly in the topic, with the alternative titles from the <topicref> taking higher priority.

The roles of an alternative title are specified by its @title-role attribute. Multiple roles may be specified, separated by white space. An alternative title must specify at least one role.

Some roles may not be meaningful in certain contexts. For example, a navigational alternate title is not meaningful in the context of a <topicgroup> element's <topicmeta>, since the element is not part of a publication's navigational structure. Such alternate titles should be ignored by processors.

Processing expectations

The processing of an alternative title depends on its roles. Processors are required to support the following role tokens.

linking
The title for use in references to the resources generated from DITA map structures, such as hierarchical parent/child/sibling links and links generated from relationship tables. In addition, this is the fallback alternative title for `navigation` and `search` roles. Other, non-standardized title roles meant for use in link generation should also use this as a fallback.

navigation
The title for use in tables of content and other navigation aids. In some cases, when processing a `<topicref>` that has no `@href`, this should also be used as the title of the generated pseudo-topic, if applicable. If not present, this role is fulfilled by the `linking` role.

search
The title for use in search results for systems that support content search. If not present, this role is fulfilled by the `linking` role.

subtitle
A subtitle for the document, generally to be rendered below the primary title in a smaller size.

hint
A hint for the benefit of map authors, allowing them to see the title of a referenced resource without needing to access the resource itself. Does not have an effect on processing or output.

-dita-use-conref-target
When present, instructs processors to use the `@title-role` tokens from the element referenced by `@conref` or `@conkeyref`. All other tokens are ignored, replaced by those on the referenced element.

Other roles may be defined by processors, authors, or content architects for specific purposes. Content architects are encouraged to develop specializations of this element specifying default roles for use by their authors for specific use cases. The base DITA vocabulary contains an alternative titles domain providing specializations of this element that fulfill each of the above roles.

Alternative titles bearing roles that are not recognized by the processor "MUST" be ignored and not appear in output.

Attributes

@title-role
Specifies the role or roles fulfilled by the alternative title. Multiple roles are separated by white space.

Examples

This section contains examples of how the `<titlealt>` element can be used.

Figure 68: Subtitles

The following map specifies a subtitle.

```
<map>
  <title>Publication title</title>
  <topicmeta>
    <titlealt title-role="subtitle">Publication subtitle</titlealt>
```

```
  </topicmeta>
</map>
```

Figure 69: Alternative titles with multiple roles

The following alternative title serves both as a navigation title and a search title.

```
<titlealt title-role="navigation search">Short title</titlealt>
```

Figure 70: Example: Multiple Titles and their Uses

The following topicref contains a number of alternative titles.

```
<topicref keys="about" href="about.dita">
  <topicmeta>
    <titlealt title-role="navigation">About the Product</titlealt>
    <titlealt title-role="linking">About This Product</titlealt>
    <titlealt title-role="search">About</titlealt>
    <titlealt title-role="hint">About the Acme TextMax 5000</titlealt>
  </topicmeta>
</topicref>
```

1. The `navigation` title will be used when rendering the table of contents.
2. The `linking` title will be used when generating parent/child/sibling and relationship table-based related links to this topic.
3. The `search` title will be used to provide the title in systems that support dynamic content searches.
4. The `hint` title describes the actual title of the topic for the benefit of map authors, but will not be used in output.

Figure 71: Custom title roles

A content architect could create a Topic specialization with custom `<titlealt>` specializations called `<windowtitle>` and `<breadcrumbtitle>`. These specializations specify default `@title-role` values of `window` and `breadcrumb`, respectively, so that authors do not have to specify those roles explicitly. Content containing these specializations could look like the following.

```
<helpTopic id="topic167">
  <title>Doing the Thing in the Place where the Stuff Is</title>
  <prolog>
    <windowtitle>Doing Things</windowtitle>
    <breadcrumbtitle>Things</breadcrumbtitle>
  </prolog>
```

They could also incorporate these elements into their map document type shell, enabling map authors to override the values in topics.

```
<topicref href="topic167.dita">
  <topicmeta>
    <breadcrumbtitle>Thing Doing</breadcrumbtitle>
  </topicmeta>
</topicref>
```

Figure 72: Navigation titles and precedence

Consider the following series of topic references:

```
<topicref href="topics.dita#one"/>
<topicref href="topics.dita#two">
  <topicmeta>
    <titlealt title-role="navigation">Topic Two (Map navigation title)</titlealt>
  </topicmeta>
</topicref>
```

```
<topicref href="topics.dita#three">
  <topicmeta>
    <titlealt title-role="linking">Topic Three (Map linking title)</titlealt>
  </topicmeta>
</topicref>
<topicref href="topics.dita#four">
  <topicmeta>
    <titlealt title-role="linking">Topic Four (Map linking title)</titlealt>
  </topicmeta>
</topicref>
```

Here is the ditabase document containing those topics:

```
<dita>
  <topic id="one">
    <title>Topic One</title>
  </topic>
  <topic id="two">
    <title>Topic Two</title>
    <prolog>
      <titlealt title-role="navigation">Topic Two (Topic navigation title)</titlealt>
    </prolog>
  </topic>
  <topic id="three">
    <title>Topic Three</title>
  </topic>
  <topic id="four">
    <title>Topic Four</title>
    <prolog>
      <titlealt title-role="navigation">Topic Four (Topic navigation title)</titlealt>
    </prolog>
  </topic>
</dita>
```

The resulting navigation structure would be as follows:

1. **Topic One** - The navigation title is pulled from the title of the topic, since neither the map nor the topic specify a navigation title.
2. **Topic Two (Map navigation title)** - The navigation title comes from the map, as its navigation title takes precedence over that in the topic.
3. **Topic Three (Map linking title)** - The navigation title comes from the map, which serves as the fallback for navigation titles when no `navigation` alternative title is provided.
4. **Topic Four (Topic navigation title)** - The navigation title comes from the topic. Even though the map specifies a `<titlealt>` with a role of `linking`, and normally maps take precedence, a `linking` alternative title is only used for navigation when there is no `navigation` alternative title available. In this case, the one from the topic is present, and is therefore used. To override the topic's navigation title in this case, the topic reference would have to explicitly provide a `navigation` alternative title. The `linking` title in the map still applies as the resource's linking title, just not its navigation title.

Figure 73: Example: Reconciling Map and Topic Alternative Titles

A `<topicref>` contains the following titles:

```
<topicref href="topic.dita">
  <topicmeta>
    <titlealt title-role="breadcrumbTitle">Doin' Stuff</titlealt>
    <titlealt title-role="longTitle">That thing you do when there's stuff that needs doing.</
titlealt>
  </topicmeta.
</topicref>
```

The referenced topic has the following prolog:

```
<prolog>
  <titlealt title-role="subtitle">Doing Stuff</titlealt>
```

```
  <titlealt title-role="breadcrumbTitle flipbookTitle">Stuff</titlealt>
</prolog>
```

During processing, the two sets of elements will be concatenated together (logically, if not physically), with the map's elements coming first:

```
<titlealt title-role="breadcrumbTitle">Doin' Stuff</titlealt>
<titlealt title-role="longTitle">That thing you do when there's stuff that needs doing.</
titlealt>
<titlealt title-role="subtitle">Doing Stuff</titlealt>
<titlealt title-role="breadcrumbTitle flipbookTitle">Stuff</titlealt>
```

Note that `breadcrumbTitle` is specified in both the map and the topic, and the map's value takes precedence. However, that same alternative title in the topic specifies an additional role of `flipbookTitle`, which is not overridden by the map, and so should be preserved.

The equivalent merged alternative titles, with duplicates removed, would look as follows.

```
<titlealt title-role="breadcrumbTitle">Doin' Stuff</titlealt>
<titlealt title-role="longTitle">That thing you do when there's stuff that needs doing.</
titlealt>
<titlealt title-role="subtitle">Doing Stuff</titlealt>
<titlealt title-role="flipbookTitle">Stuff</titlealt>
```

Figure 74: Example: Conrefs and Title Roles

Consider the following conrefs:

```
<titlealt title-role="linking" conref="titles.dita#someAltTitle">
```

In this case, the roles for the resolved element will be `linking`; any other roles specified on the referenced element will be lost.

```
<titlealt title-role="search -dita-use-conref-target" conref="titles.dita#someAltTitle">
```

In this case, the roles for the resolved element will be whatever is specified on the referenced element.

Note The `search` token has no effect, and will be ignored. The presence of `-dita-use-conref-target` in `@title-role` overrides all other tokens. The roles on the referencing and referenced elements are not merged; only those from the referenced element apply.

Figure 75: Keyrefs and alternative titles

Consider the following two topic references:

```
<topicref keys="a">
  <topicmeta>
    <titlealt title-role="linking">Linking Title from Keyref</titlealt>
    <titlealt title-role="navigation">Navigation Title from Keyref</titlealt>
  </topicmeta>
</topicref>
<topicref keyref="a">
  <topicmeta>
    <titlealt title-role="navigation">Navigation Title</titlealt>
  </topicmeta>
</topicref>
```

The resolved titles would look something like this:

```
<titlealt title-role="navigation">Navigation Title</titlealt>
<titlealt title-role="linking">Linking Title from Keyref</titlealt>
<titlealt title-role="navigation">Navigation Title from Keyref</titlealt>
```

That is, the "local" alternative titles come before those pulled from the key reference. In cases where only a single alternative title of a given role can be used, the first takes precedence, so the `navigation` title from the key reference has no effect.

10.3.1.9 <topic>

A topic is a standalone unit of information.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (with a narrowed definition of `@id`, given below) and Architectural attributes (369).

@id (REQUIRED)

Provides an anchor point. This ID is usually required as part of the `@href` or `@conref` syntax when cross referencing or reusing content within the topic; it also enables `<topicref>` elements in DITA maps to optionally reference a specific topic within a DITA document. This attribute is defined with the XML Data Type ID.

Example

The following code sample shows the primary structural components of a topic: title, short description, prolog, body, and related links.

```
<topic id="topic">
  <title>The basic structure of a topic</title>
  <shortdesc/>
  <prolog/>
  <body/>
  <related-links/>
</topic>
```

10.3.2 Body elements

The body elements support the most common types of content authoring for topics: paragraphs, lists, phrases, figures, and other common types of exhibits in a document.

10.3.2.1 <alt>

Alternate text is a textual description of an image. Systems can display the alternate text when the image cannot be rendered or viewed by the reader.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows how alternate text is associated with an image of a marketing banner:

```
<image href="newCampaign.jpg">
  <alt>Marketing banner for new product campaign</alt>
</image>
```

10.3.2.2 <cite>

A citation indicates the title of a bibliographic resource.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and `@keyref` (388).

Example

In the following code sample, the `<cite>` element is used to mark up the title of an article.

```
<p>The online article <cite>Specialization in the Darwin Information Typing
Architecture</cite> provides a detailed explanation of how to define new
topic types.</p>
```

10.3.2.3 <dd>

The definition description is the definition for a term in a definition list entry.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.3.2.7 dl (216).

10.3.2.4 <ddhd>

The `<ddhd>` element provides an optional heading or title for a column of descriptions or definitions in a definition list (`<dl>`).

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.3.2.7 dl (216).

10.3.2.5 <desc>

A description is a statement that describes or contains additional information about an object.

Usage information

The following list outlines common uses of the `<desc>` element:

<table> and <fig>
: Provides more information than can be contained in the title

<xref> and <link>
: Provides a description of the target

<object>
: Provides alternate content to use when the context does not permit displaying the object

Rendering expectations

When used in conjunction with <fig> or <table> elements, processors *SHOULD* consider the content of <desc> elements to be part of the content flow.

When used in conjunction with <xref> or <link> elements, processors *MAY* choose to render the content of <desc> elements as hover help.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Examples

This section contains examples of how the <desc> element can be used.

Figure 76: Description of a figure

In the following code sample, the <figure> element contains a reference to an image of a famous painting by Leonardo Da Vinci. The <title> element provides the name of the painting, while the <desc> element contains information about when the portrait is thought to have been painted.

```
<fig>
<title>Mona Lisa</title>
<desc>Circa 1503-06, perhaps continuing until 1517</desc>
<image href="mona-lisa.jpg">
 <alt>Photograph of Mona Lisa painting</alt>
</image>
</fig>
```

Figure 77: Description of a cross reference

In the following code sample, the <link> element contains a <desc> element. Some processors might render the content of the <desc> element as hover help.

```
<link keyref="dita-13-02">
  <linktext>DITA 1.3 Errata 02</linktext>
  <desc>Final errata version of DITA 1.3, published 19 June 2018</desc>
</link>
```

10.3.2.6

A division is a grouping of sequential content within a topic. There is no additional semantic meaning.

Usage information

The element is useful primarily as a specialization base and for reuse.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

This section contains examples of how the element can be used.

Figure 78: Using for grouping

In the following example, a element is used to organize several elements together so that they can be referenced by @conref or @conkeyref:

```
...
<div id="div-01">
 <p>The first paragraph</p>
 <p>The second paragraph</p>
 <note>This is a note</note>
</div>
...
```

Without using a element, the content could not be grouped for content referencing since the start and end elements are of different types.

Figure 79: Using for specialization

In the following example, is used as the basis for specializing a new domain element, <pullquote>:

```
<!ENTITY % pullquote.content
  "(%div.cnt;)*"
>
<!ENTITY % pullquote.attributes
             "%univ-atts;"
>
<!ELEMENT pullquote    %pullquote.content;>
<!ATTLIST pullquote    %pullquote.attributes;>

<!ATTLIST pullquote      class CDATA "+ topic/div pubcontent-d/pullquote ">
```

Instances of <pullquote> could then be used in both <body> and <section> contexts:

```
<topic id="article-01">
  <title>My Article</title>
  <body>
    <p>Something pithy someone said</p>
    <pullquote><p>Something Pithy</p></pullquote>
    <!-- ... -->
    <section spectitle="Deep Dive">
      <p>This is really really pithy</p>
      <pullquote><p>Really Pithy</p></pullquote>
      <!-- ... -->
    </section>
  </body>
</topic>
```

10.3.2.7 <dl>

A definition list is a list of terms and corresponding definitions.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Examples

The following code sample shows how a definition list can be used to describe the message levels that are generated by a monitoring application. The `@compact` attribute instructs processors to tighten the vertical spacing.

```
<dl compact="yes">
  <dlentry>
    <dt>Warning</dt>
    <dd>Problems were detected, but the software will continue to monitor activity.</dd>
  </dlentry>
  <dlentry>
    <dt>Error</dt>
    <dd>Problems were detected, and the software is in danger of shutting down.</dd>
  </dlentry>
  <dlentry>
    <dt>Severe</dt>
    <dd>Monitoring activity has ceased.</dd>
  </dlentry>
</dl>
```

10.3.2.8 <dlentry>

A definition list entry is a group within a definition list. It associates a term with its definition.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.3.2.7 dl (216).

10.3.2.9 <dlhead>

The `<dlhead>` element contains optional headings for the term and description columns in a definition list. The definition list heading might contain a heading for the column of terms (`<dthd>`) and a heading for the column of descriptions (`<ddhd>`).

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows a definition list with a header:

```
<dl>
 <dlhead>
  <dthd>Image selection</dthd>
  <ddhd>Resulting information</ddhd>
 </dlhead>
 <dlentry>
  <dt>File Type</dt>
  <dd>The file extension of the image</dd>
 </dlentry>
 <dlentry>
  <dt>Image class</dt>
  <dd>Whether the image is raster, vector, or 3D</dd>
 </dlentry>
 <dlentry>
  <dt>Number of pages</dt>
  <dd>Number of pages in the image</dd>
 </dlentry>
```

```
  <dlentry>
   <dt>Fonts</dt>
   <dd>Names of the fonts contained within a vector image</dd>
  </dlentry>
 </dl>
```

Rendering of definition lists will vary by application and by display format. Processors might render the code sample in the following way:

Image selection	Resulting information
File type	File extension of the image
Image class	Whether the image is raster, vector, or 3D
Number of pages	Number of pages in the image
Fonts	Names of the fonts contained within a vector image

10.3.2.10 <draft-comment>

A draft comment is content that is intended for review and discussion, such as questions, comments, and notes to reviewers. This content is not intended to be included in production output.

Rendering expectations

By default, processors *SHOULD NOT* render `<draft-comment>` elements. Processors *SHOULD* provide a mechanism that causes the content of the `<draft-comment>` element to be rendered in draft output only.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (with a narrowed definition for `@translate`, given below) and the attributes defined below.

@author
Designates the originator of the draft comment.

@disposition
Specifies the status of the draft comment.

@time
Specifies when the draft comment was created.

@translate
Specifies whether the content of the element is translatable. The default value is "no". Setting `@translate` to "yes" overrides the default value. The DITA specification contains a list of each OASIS DITA element and its common processing default for the translate value; because this element uses an actual default, it is always treated as `translate="no"` unless overridden. Available values are:

no
The content of this element is not translatable.

yes
The content of this element is translatable.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

Example

The following example illustrates how an content developer can use a <draft-comment> element to pose a question to reviewers.

```
<draft-comment
  author="EBP"
  time="23 May 2017"
  status="missing-info">
Where's the usage information for this section?
</draft-comment>
```

Processors might render the information from the highlighted attributes at viewing or publishing time. Authors might use the value of the @status attribute to track the work that remains to be done on a content collection.

10.3.2.11 <dt>

A definition term is the term or phrase that is defined in a definition list entry.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.3.2.7 dl (216).

10.3.2.12 <dthd>

The <dthd> element provides an optional heading for the column of terms in a definition list (<dl>).

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.3.2.7 dl (216).

10.3.2.13 <example>

An example helps to illustrate the subject of the topic or a portion of the topic.

Usage information

Use <example> to contain both sample code (or similar artifacts) and the discussion that illustrates the sample. For example, a topic about programming code could use the <example> element to contain both the sample code and the text that describes the code.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and spectitle (specialization attributes) (376).

Example

The following code sample shows an <example> element that contains a code block and a textual explanation of it:

```
<section id="AddingRecord">
   <title>ADD</title>
   <p>New database records are created using the <cmdname>ADD</cmdname> command.</p>
   <example>
      <p>The following example illustrates the creation of a new record.
All parameter settings are strictly optional.</p>
      <codeblock>01 OPTIONS ABC,ADD,DEF,HIJK,LMNO,AOW=25000,HF=2</codeblock>
   </example>
</section>
```

10.3.2.14 <fallback>

The <fallback> element provides content to be presented when multimedia objects cannot be rendered.

Processing expectations

The contents of this element are displayed only when the media that is referenced by the containing <object> element cannot be displayed.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See <object> (231).

10.3.2.15 <fig>

A figure is a container for a variety of objects, including artwork, images, code samples, equations, and tables.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Display attributes (369), and spectitle (specialization attributes) (376).

Example

The following code sample shows how a <fig> element can associate a title and a description with an image.

```
<fig>
<title>The handshake</title>
<desc>This image shows two hands clasped in a formal,
business-like handshake.</desc>
<image href="59j0p66.jpg">
 <alt>A handshake</alt>
</image>
</fig>
```

10.3.2.16 <figgroup>

A figure group organizes segments within a figure.

Comment by Kristen J Eberlein on 09 September 2020

Should this topic be located here in "Body elements," or should it be relocated to "Specialization elements?

Usage information

The `<figgroup>` element is useful primarily as a base for complex specializations, such as nestable groups of syntax within a syntax diagram. The `<figgroup>` element can nest; it also can contain multiple cross-references, footnotes, and keywords.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows how the `<figgroup>` can group content with associated metadata:

```
<fig>
  <title>Sample complex figure</title>
  <figgroup>
    <data name="MetaItem" value="13"/>
    <data name="MetaThing" value="31"/>
    <ph>These elements are grouped with associated metadata</ph>
  </figgroup>
</fig>
```

10.3.2.17 <fn>

A footnote is ancillary information that typically is rendered in the footer of a page or at the end of an online article. Such content is usually inappropriate for inline inclusion.

Usage information

There are two types of footnotes: *single-use footnote* and *use-by-reference footnote*.

Single-use footnote
: This is produced by a `<fn>` element that does not specify a value for the `@id` attribute.

Use-by-reference footnote
: This is produced by a `<fn>` element that specifies a value for the `@id` attribute. It must be used in conjunction with an `<xref>` element with `@type` set to "fn".

To reference a footnote that is located in another topic, the conref or conkeyref mechanism is used.

Rendering expectations

The two footnote types typically produce different types of output:

Single-use footnote
: When rendered, a superscript symbol (numeral or character) is produced at the location of the `<fn>` element. The superscript symbol is hyperlinked to the content of the footnote, which is placed at the bottom of a PDF page or the end of an online article. The superscript symbol can be specified by the value of the `@callout` attribute. When no `@callout` value is specified, footnotes are typically numbered.

Use-by-reference footnote

Nothing is rendered at the location of the <fn> element. The content of a use-by-reference footnote is only rendered when it is referenced by an <xref> with the @type attribute set to "fn". If an <xref> with the @type attribute set to "fn" is present, a superscript symbol is rendered at the location of the <xref> element. Unless conref or conkeyref is used, the <fn> and <xref> must be located in the same topic.

However, the details of footnote processing and formatting are implementation dependent. For example, a tool that renders DITA as PDF might lack support for the @callout attribute, or footnotes might be collected as end notes for certain types of publications.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attribute defined below.

@callout

Specifies the character that is used for the footnote link, for example, a number or an alphabetical character. The attribute also can specify a short string of characters.

Examples

This section contains examples of how the <fn> element can be used.

Figure 80: Example of a single-use footnote

The following code sample shows a single-use footnote. It contains a simple <fn> element, with no @id or @callout attribute.

```
The memory storage capacity of the computer is
2 GB<fn>A GB (gigabyte) is equal to
1000 million bytes</fn> with error correcting support.
```

When rendered, typically a superscript symbol is placed at the location of the <fn> element; this superscript symbol is hyperlinked to the content of the <fn>, which is typically is placed at the bottom of a PDF page or the end of an online article. The type of symbol used is implementation specific.

The above code sample might produce output similar to the following:

The memory storage capacity of the computer is 2 GB[1] with error correcting support.

. . . . . .

[1] A GB (gigabyte) is equal to 1000 million bytes

----- [bottom of page] ----------

Figure 81: Example of a single-use footnote with a @callout attribute

The following code sample shows a single-use footnote that uses a @callout attribute:

```
The memory storage capacity of the computer is
2 GB<fn callout="#">A GB (gigabyte) is equal to
1000 million bytes</fn> with error correcting support.
```

The rendered output is similar to that of the previous example, although processors that support it will render the footnote symbol as # (hashtag).

Figure 82: Example of a use-by-reference footnote

The following code sample shows use-by-reference footnotes. The `<fn>` elements have `@id` attributes, and inline `<xref>` elements reference those `<fn>` elements:

```
<fn id="dog-name">Fido</fn>
<fn id="cat-name">Puss</fn>
<fn id="llama-name">My llama</fn>
...
<p>I like pets. At my house, I have a dog<xref href="#topic/dog-name" type="fn"/>, a
cat<xref href="#topic/cat-name" type="fn"/>, and a
llama<xref href="#topic/llama-name" type="fn"/>.</p>
```

The code sample might produce output similar to the following:

. . . . . .

I like pets. At my house, I have a dog[1], a cat[2], and a llama[3].

. . . . . .

[1]Fido

[2]Puss

[3]My llama

----- [bottom of page] ----------

Figure 83: Example of a single-use footnote that uses conref

The following code sample shows footnotes stored in a shared topic (`footnotes.dita`):

```
<!-- Content from footnotes.dita -->
<topic id="footnotes">
...
  <fn id="strunk">Elements of Style</fn>
  <fn id="DQTI">Developing Quality Technical Information, 2nd edition</fn>
...
</topic>
```

To use those footnotes, authors conref them into the relevant topics:

```
<p>See the online resource<fn conref="footnotes.dita#footnotes/DQTI"/> for more
   information about how to assess the quality of technical documentation ...</p>
```

Figure 84: Example of a use-by-reference footnote that uses conref

The following code sample shows a use-by-reference footnote that uses conref:

```
<topic id="evaluating-quality">
  <title>Evaluating documentation quality</title>
  <body>
  ...
  <fn conref="footnotes.dita#footnotes/DQTI" id="dqti"/>
  ...
  <p>See the online resource<xref="./evaluating-quality/dqti" type="fn"/> for more
     information about how to assess the quality of technical documentation ...</p>
  ...
  </body>
<topic>
```

10.3.2.18 <image>

An image is a reference to artwork that is stored outside of the content.

Rendering expectations

The referenced image typically is rendered in the main flow of the content.

Processors *SHOULD* scale the object when values are provided for the `@height` and `@width` attributes. The following expectations apply:

- If a height value is specified and no width value is specified, processors *SHOULD* scale the width by the same factor as the height.
- If a width value is specified and no height value is specified, processors *SHOULD* scale the height by the same factor as the width.
- If both a height value and width value are specified, implementations *MAY* ignore one of the two values when they are unable to scale to each direction using different factors.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), `@keyref` (388), and the attributes defined below.

@align
Controls the horizontal alignment of an image when `@placement` is specified as "break". Common values include "left", "right", and "center".

@format
Identifies the format of the resource that is referenced. See 10.8.3.5 The format attribute (386) for details on supported values.

@height
Indicates the vertical dimension for the resulting display. The value of this attribute is a real number (expressed in decimal notation) optionally followed by a unit of measure from the set of pc, pt, px, in, cm, mm, em (picas, points, pixels, inches, centimeters, millimeters, and ems respectively). The default unit is px (pixels). Example values include "5", "5in", and "10.5cm".

@href
Provides a reference to the image. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications.

@placement
Indicates whether an image is displayed inline or on a separate line. The default value is inline. Allowable values are "inline", "break", or and "-dita-use-conref-target" (385).

@scale
Specifies a percentage as an unsigned integer by which to scale the image in the absence of any specified image height or width; a value of 100 implies that the image should be presented at its intrinsic size. If a value has been specified for the `@height` or `@width` attribute (or both), the `@scale` attribute is ignored.

It is an error if the value of this attribute is not an unsigned integer. In this case, the implementation might give an error message and might recover by ignoring this attribute.

@scalefit
Allows an image to be scaled up or down to fit within available space. Allowable values are" yes", "no", and "-dita-use-conref-target" (385). If `@height`, `@width`, or `@scale` is specified, those attributes determine the graphic size, and the `@scalefit` attribute is ignored. If none of those

attributes are specified and `scalefit="yes"`, then the image is scaled (the same factor in both dimensions) so that the graphic will just fit within the available height or width (whichever is more constraining).

The available width would be the prevailing column (or table cell) width—that is, the width a paragraph of text would have if the graphic were a paragraph instead. The available height is implementation dependent, but if feasible, it is suggested to be the page (or table cell) height or some other reasonable value.

@scope
Identifies the closeness of the relationship between the current document and the target resource. Allowable values are "local", "peer", "external", and "-dita-use-conref-target" (385). See 10.8.3.11 The scope attribute (391) for more information on values.

@width
Indicates the horizontal dimension for the resulting display. The value of this attribute is a real number (expressed in decimal notation) optionally followed by a unit of measure from the set of pc, pt, px, in, cm, mm, em (picas, points, pixels, inches, centimeters, millimeters, and ems respectively). The default unit is px (pixels). Example values include "5", "5in", and "10.5cm".

Example

The following code sample shows how an image is referenced. The `@placement` attribute is set to "break"; this ensures that the image is not rendered inline.

```
<image href="bike.gif" placement="break">
  <alt>Two-wheeled bicycle</alt>
</image>
```

10.3.2.19 <include>

The `<include>` element specifies that non-DITA content from a resource outside the current document should be placed at that location. The resource is specified using either a URI or a key reference. Processing expectations for the referenced data can also be specified.

Comment by Kristen J Eberlein on 09 September 2020

Should this topic be located here in "Body elements," or should it be relocated to "Specialization elements?

Usage information

The `<include>` element is intended as a specialization base and for the following use cases:

- The transclusion of non-DITA XML within `<foreign>` element using `parse="xml"`
- The transclusion of preformatted textual content within `<pre>` element using `parse="text"`
- The transclusion of plain-text prose within DITA elements using `parse="text"`

It is an error when the `<include>` element is used to reference DITA content. Authors should use `@conref` or `@conkeyref` to reuse DITA content.

Processing expectations

Comment by Kristen J Eberlein on 29 April 2019

What of the content in this section (not yet edited) should be normative statements?

The <include> element instructs processors to insert the contents of the referenced resource at the location of the <include> element. If the content is unavailable to the processor or cannot be processed using the specified @parse value, the contents of the <fallback> element, if any, are presented instead.

If the processor cannot process the referenced content using the rules implied by the @parse attribute, either because the referenced scheme is not supported or because there was a processing error, processors should issue a warning or error. All processors are expected to support the @parse values "text" and" xml".

Processors are expected to detect the encoding of the referenced document based on the rules described for the @encoding (372) attribute.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Inclusion attributes (369), Link-relationship attributes (369), and 10.8.3.7 The keyref attribute (388).

Examples

For the most part, <include> should be used as a basis for specialization. The following examples use it directly for purposes of illustration.

This section contains examples of how the <include> element can be used.

Figure 85: Inclusion of XML markup other than SVG or MathML

In the following code sample, the <include> element references a tag library descriptor file:

```
<fig>
  <title>JSP Tag Library Elements and Attributes</title>
  <foreign outputclass="tld">
    <include href="../src/main/webapp/WEB-INF/jsp-tag-library.tld"
             parse="xml" format="tld"/>
  </foreign>
</fig>
```

Figure 86: Inclusion of README text into a DITA topic, with fallback

```
<shortdesc>
  <include href="../src/README.txt" parse="text" encoding="UTF-8">
    <fallback>This topic describes XYZ.</fallback>
  </include>
</shortdesc>
```

Figure 87: Inclusion of preformatted text

In the following code sample, the <include> element references a JSON file:

```
<pre>
```

```
  <include href="../src/config.json" format="json" parse="text" encoding="UTF-8"/>
</pre>
```

Figure 88: Inclusion of README as Markdown converted to DITA using a proprietary @parse value, with fallback

```
<section>
  <include href="about.md" encoding="UTF-8"
    parse="http://www.example.com/dita/includeParsers/markdown-to-dita">
      <fallback>This section not available.</fallback>
  </include>
</section>
```

Figure 89: Proprietary vendor handling for CSV tables

```
<fig>
  <title>Data Table</title>
  <include href="data.csv"  encoding="UTF-8"
    parse="http://www.example.com/dita/includeParsers/csv-to-simpletable"/>
</fig>
```

10.3.2.20 <keyword>

A keyword is text or a token that has a unique or key-like value, such as a product name or unit of reusable text.

Processing expectations

When used within the `<keywords>` element, the content of a `<keyword>` element is considered to be metadata and should be processed as appropriate for the given output medium.

Elements that are specialized from the `<keyword>`element might have extended processing, such as specific formatting or automatic indexing.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and `@keyref` (388).

Example

This section contains examples of how the `<keyword>` element can be used.

Figure 90: <keyword> element used to store a product name

In the following code sample, the `<keyword>` element holds a product name.

```
<keyword id="acme-bird-feeder">ACME Bird Feeder</keyword>
```

The product name can be referenced using one of the DITA reuse mechanisms: content reference (conref), content key reference (conkeyref), or key reference (keyref).

Figure 91: <keyword> element as metadata

In the following code sample, "Big data" is specified as metadata that applies to the topic.

```
<prolog>
  <metadata>
    <keywords>
```

```
      <keyword>Big data</keyword>
    </keywords>
  </metadata>
</prolog>
```

10.3.2.21

A list item is an item in either an ordered or unordered list.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.3.2.28 ol (234) or 10.3.2.41 ul (242)

10.3.2.22 <lines>

Lines are lines of text where white space is significant. It can be used to represent dialogs, poetry, or other text fragments where line breaks are significant.

Rendering expectations

Processors *SHOULD* preserve or otherwise indicate white space within the `<lines>` element.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Display attributes (369), xml:space (377), and spectitle (specialization attributes) (376).

Example

In the following code sample, a `<lines>` element contains an excerpt from Sonnet 18, one of the best-known of the 154 sonnets written by the English playwright and poet William Shakespeare:

```
<lines>
Shall I compare thee to a summer's day?
Thou art more lovely and more temperate:
Rough winds do shake the darling buds of May,
and summer's lease hath all too short a date:
...</lines>
```

10.3.2.23 <longdescref>

A long description reference is a reference to a textual description of a graphic or object.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369), and `@keyref` (388).

Example

This section contains examples of how the <longdescref> element can be used.

Figure 92: <longdescref> which references a local DITA description

In the following code sample, the <longdescref> references a detailed image description that is stored in a DITA topic.

```
<image href="llama.jpg">
  <alt>Llama picture</alt>
  <longdescref href="my-pet-llama.dita"/>
</image>
```

Figure 93: <longdescref> which references an external description

In this code sample, the long description is stored remotely, on a external Web site.

```
<image href="puffin.jpg">
  <alt>Puffin pigure</alt>
  <longdescref href="http://www.example.org/birds/puffin.html"
               scope="external"
               format="html"/>
</image>
```

10.3.2.24 <longquoteref>

A long quotation reference is a reference to the source of a lengthy quotation.

Processing expectations

Rendering of this element is left up to DITA processors. Depending on the presentation format, it might be appropriate to ignore the element, present it as a link, use it to turn the entire quotation into a link, or do something else.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369), and @keyref (388).

Example

Comment by Kristen J Eberlein on 05 August 2018

When trying to craft a new example for this element, I realized I don't understand what value it provides that is not already available in <lq>.

In the following code sample, the <longdescref> element references an online version of *As You Like It*.

```
<p>The following is one of the most frequently used quotations from a Shakespearean play:
  <lq>All the world's a stage, and all the men and women merely players. They have
      their exits and their entrances; And one man in his time plays many parts.’
    <longquoteref href="http://www.example.org/shakespeare/as-you-like-it" scope="external"/>
  </lq>
</p>
```

10.3.2.25 <lq>

A long quotation is a quotation that contains one or more paragraphs. The title and source of the document that is being quoted can be specified.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with a narrowed definition for `@type`, given below), and `@keyref` (388), and the attributes defined below.

@reftitle
The title of the document or topic that is quoted.

@type
Indicates the location of the source of the quote. Note that this differs from the `@type` attribute on many other DITA elements. See 10.8.3.12 The type attribute (391) for detailed information on the usual supported values and processing implications.

Example

The following code sample contains a quotation. The `@reftitle` attribute specifies the title of the document that is quoted, and the `@href` attribute indicates a Web site where the full text of the address can be accessed.

```
<p>This is the first line of the address that Abraham Lincoln delivered
on November 19, 1863 for the dedication of the cemetery at Gettysburg, Pennsylvania.</p>
<lq reftitle="Gettysburg address"
href="https://en.wikisource.org/wiki/Gettysburg_Address_(Nicolay_draft)" format="html"
scope="external">Four score and seven years ago our fathers brought forth on this continent
a new nation, conceived in liberty, and dedicated to the proposition that all men
are created equal.</lq>
```

10.3.2.26 <note>

A note contains information that expands on or calls attention to a particular point.

Usage information

Variant types of notes (caution, danger, warning, etc.) can be indicated through values selected for the `@type` attribute.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), spectitle (specialization attributes) (376), and the attributes defined below.

@othertype
Specifies an alternate note type. This value is used as the user-provided note title when the `@type` attribute value is set to "other".

@type
Specifies the type of a note. Note that this differs from the `@type` attribute on many other DITA elements. See 10.8.3.12 The type attribute (391) for detailed information on supported values and processing implications. Available values are "note", "tip", "fastpath", "restriction", "important", "remember", "attention", "caution", "notice", "danger", "warning", "trouble", "other", and "-dita-use-conref-target" (385).

Example

The following code sample shows a <note> with @type set to "tip":

```
<note type="tip">Thinking of a seashore, green meadow, or cool
mountain overlook can help you to relax and be more
patient.</note>
```

10.3.2.27 <object>

The DITA <object> element corresponds to the HTML <object> element, and attribute semantics derive from their HTML definitions. For example, the @type attribute differs from the @type attribute on many other DITA elements.

Usage information

The <object> element enables authors to include animated images, applets, plug-ins, ActiveX controls, video clips, and other multimedia objects in a topic.

Rendering expectations

Processors *SHOULD* scale the object when values are provided for the @height and @width attributes. The following expectations apply:

- If a height value is specified and no width value is specified, processors *SHOULD* scale the width by the same factor as the height.
- If a width value is specified and no height value is specified, processors *SHOULD* scale the height by the same factor as the width.
- If both a height value and width value are specified, implementations *MAY* ignore one of the two values when they are unable to scale to each direction using different factors.

When an object cannot be rendered in a meaningful way, processors *SHOULD* present the contents of the <fallback> element, if it is present.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attributes defined below.

@archive
Specifies a space-separated list of URIs indicating resources needed by the object. These resources might include those URIs specified by the @classid and @data attributes. Preloading these resources usually results in faster load times for objects. The URIs in the list should be relative to the URI specified in the @codebase attribute.

@archivekeyrefs
Key references to one or more archives, as for @archive. The value is a space-separated list of key names. Each resolvable key reference is treated as a URI as though it had been specified on the @archive attribute. When specified, and at least one key name is resolvable, the key-provided archive list is used. If @archive is specified, it is used as a fallback when no key names can be resolved to a URI.

@classid
Contains a URI that specifies the location of an object's implementation. It can be used together with the @data attribute which is specified relative to the value of the @codebase attribute.

@classidkeyref
Key reference to the URI that specifies the location of an object's implementation, as for `@classid`. When specified, and the key is resolvable, the key-provided class ID URI is used. If `@classid` is specified, it is used as a fallback when the key cannot be resolved to a URI.

@codebase
Specifies the base URI used for resolving the relative URI values given for `@classid`, `@data`, and `@archive` attributes. If `@codebase` is not set, the default is the base URI of the current element.

@codebasekeyref
Key reference to the base URI used for resolving other attributes, as for `@codebase`. When specified, and the key is resolvable, the key-provided code base URI is used. If `@codebase` is specified, it is used as a fallback if the key cannot be resolved to a URI. If no URI results from processing `@codebasekeyref` and `@codebase` is not specified, the default is the base URL of the current element.

@data
Contains a reference to the location of an object's data. If this attribute is a relative URL, it is specified relative to the value of the `@codebase` attribute. If this attribute is set, the `@type` attribute should also be set.

@datakeyref
Provides a key reference to the object. When specified and the key is resolvable, the key-provided URI is used. A key that has no associated resource, only link text, is considered to be unresolved. If `@data` is specified, it is used as a fallback when the key cannot be resolved to a resource.

@declare
When this attribute is set to "declare", the current object definition is a declaration only. The object must be instantiated by a later nested object definition referring to this declaration. The only allowable value is "declare".

@type
Indicates the content type (MIME type) for the data specified by the `@data` or `@datakeyref` attribute. This attribute should be set when the `@data` attribute is set to avoid loading unsupported content types. Note that this differs from the `@type` attribute on many other DITA elements (it specifies a MIME type rather than a content type). If `@type` is not specified, the effective type value for the key named by the `@datakeyref` attribute is used as the this attribute's value.

@standby
Contains a message to be displayed while an object is loading.

@height
Specifies the vertical dimension for the resulting display. The value of this attribute is a real number (expressed in decimal notation) optionally followed by a unit of measure from the set of pc, pt, px, in, cm, mm, em (picas, points, pixels, inches, centimeters, millimeters, and ems respectively). The default unit is px (pixels). Possible values include: "5", "5in", and "10.5cm".

@width
Specifies the horizontal dimension for the resulting display. The value of this attribute is a real number (expressed in decimal notation) optionally followed by a unit of measure from the set of pc, pt, px, in, cm, mm, em (picas, points, pixels, inches, centimeters, millimeters, and ems respectively). The default unit is px (pixels). Possible values include: "5", "5in", and "10.5cm".

@usemap
Indicates that a client-side image map is to be used. An image map specifies active geometric regions of an included object and assigns a link to each region. When a link is selected, a document might be retrieved or a program might run on the server.

@name
Defines a unique name for the object.

@tabindex
Position the object in tabbing order.

Example

Output processors might need to modify data in order to enable compatible function across various browsers, so these examples are only representative:

```
<p>Cutting the keys from the system unit:</p>
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
 codebase="http://download.macromedia.com/pub/shockwave/cabs/
flash/swflash.cab#version=6,0,0,0"
 data="cutkey370.swf"
 type="application/x-shockwave-flash"
 height="280"
 width="370"
 id="cutkey370">
 <desc>A description of the task</desc>
 <fallback>Media not available.</fallback>
 <param name="movie" value="cutkey370.swf"/>
 <param name="quality" value="high"/>
 <param name="bgcolor" value="#FFFFFF"/>
</object>
```

```
<p>What's EIM?</p>
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
 codebase="http://download.macromedia.com/pub/shockwave/cabs/
flash/swflash.cab#version=6,0,0,0"
 data="eim.swf"
 height="400"
 width="500"
 id="eim">
 <desc>Some great, glorious info</desc>
 <fallback><image href="media-not-available.png"/></fallback>
 <param name="movie" value="eim.swf"/>
 <param name="quality" value="high"/>
 <param name="bgcolor" value="#FFFFFF"/>
 <param name="pluginspace"
 value="http://www.macromedia.com/go/getflashplayer"/>
</object>
```

Figure 94: Object with reference to video using key reference on the <param> elements

```
<object
  id="E5123_026.mp4"
  width="300"
  height="300">
  <fallback>Media not available.</fallback>
  <param name="poster"
     keyref="E5123_026_poster"
  />
  <param name="source"
     keyref="E5123_026_video"
  />
</object>
```

Where the keys could be:

```
<map>
  <!-- ... -->
  <keydef keys="E5123_026_poster"
      href="../images/E5123_026_poster.png"
      type="video/mp4"
  />
  <keydef keys="E5123_026_video"
```

```
      href="../media/E5123_026_poster.mp4"
      type="video/mp4"
  />
  <!-- ... -->
</map>
```

Figure 95: Object with indirect reference to a flash file and fallback @data value

```
<object
 classidkeyref="video_classid"
 codebasekeyref="video_codebase"
 datakeyref="cutkey370"
 height="280"
 width="370"
 id="cutkey370">
 <desc>A description of the task</desc>
 <fallback>Media not available.</fallback>
 <param name="movie" keyref="cutkey370"/>
 <param name="quality" value="high"/>
 <param name="bgcolor" value="#FFFFFF"/>
</object>
```

Where the key could be:

```
<map>
  <!-- ... -->
  <!-- NOTE: Using @scope="external" because
       the class ID is a URI that is not intended to
       be directly resolved.
    -->
  <keydef keys="video_classid"
    href="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    scope="external"
  />
  <!-- NOTE: Using @scope="external" to avoid systems trying to
             download this file when they don't need to.
    -->
  <keydef keys="video_codebase"
    href="http://download.macromedia.com/pub/shockwave/cabs/
flash/swflash.cab#version=6,0,0,0"
    format="shockwave"
    scope="external"
  />
  <!-- Using @scope="external" here because the referenced URL
       is not intended to be resolved in isolation but relative
       to the codebase URI.
    -->
  <keydef keys="cutkey370"
      href="cutkey370.swf"
      type="application/x-shockwave-flash"
      scope="external"
  />
  <!-- ... -->
</map>
```

10.3.2.28

An ordered list is a list of items that are sorted by sequence or order of importance.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), compact (371), and spectitle (specialization attributes) (376).

Example

The following code sample shows the use of an ordered list:

```
<p>Here are Rotten Tomatoes' five best movies of all time:</p>
<ol>
 <li>The Wizard of Oz (1939)</li>
 <li>Citizen Kane (1941)</li>
 <li>Get Out (2017)</li>
 <li>The Third Man (1949)</li>
 <li>Mad Max: Fury Road (2015)</li>
</ol>
```

10.3.2.29 <p>

A paragraph is a single unit of text that contains a main idea.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample contains a paragraph:

```
<p>A paragraph is a group of related sentences that support a central
idea. Paragraphs typically consist of three parts: a topic sentence, body sentences,
and a concluding or bridging sentence.</p>
```

10.3.2.30 <param>

The `<param>` (parameter) element specifies a set of values that might be required by an `<object>` at runtime.

Usage information

Any number of `<param>` elements might appear in the content of an `<object>` in any order, but must be placed at the start of the content of the enclosing object. This element is comparable to the XHMTL `<param>` element, and its attributes' semantics derive from their HTML definitions. For example, the `@type` attribute differs from the `@type` attribute on many other DITA elements.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attributes defined below.

@name (REQUIRED)
The name of the parameter.

@value
Specifies the value of a run-time parameter that is specified by the `@name` attribute.

@valuetype
Specifies the type of the `@value` attribute. Allowed values are:

data
A value of data means that the value will be evaluated and passed to the object's implementation as a string.

ref
A value of ref indicates that the value of the `@value` attribute is a URL that designates a resource where run-time values are stored. This allows support tools to identify URLs that are given as parameters.

object
A value of object indicates that the value of `@valuetype` is an identifier that refers to an object declaration in the document. The identifier must be the value of the ID attribute set for the declared object element.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

@type
This attribute specifies for a user agent the type of values that will be found at the URI designated by `@value`. Note that this differs from the `@type` attribute on many other DITA elements.

1. When `@valuetype` is set to "ref", this attribute directly specifies the content type of the resource designated by `@value`.
2. Otherwise, if `@type` is specified and `@keyref` is specified and resolvable, this attribute specifies the content type of the resource designated by `@keyref`.
3. Otherwise, if `@type` is not specified and `@keyref` is specified and is resolvable, the effective type value specified for the key that is named by the `@keyref` attribute is used as the value of the `@type` attribute.

@keyref
Key reference to the thing the parameter references. If `@valuetype` is specified but is not set to "ref", this attribute is ignored. When `@valuetype` is not specified and `@keyref` is specified, it implies a setting of `valuetype="ref"`. When `@keyref` is specified and the effective value of `@valuetype` is "ref":

1. When the key specified by `@keyref` is resolvable and has an associated URI, that URI is used as the value of this element (overriding `@value`, if that is specified).
2. When the key specified by `@keyref` is resolvable and has no associated resource (only link text), the `@keyref` attribute is considered to be unresolvable for this element. If `@value` is specified, it is used as fallback.
3. When the key specified by `@keyref` is not resolvable, the value of the `@value` attribute is used as a fallback target for the `<param>` element.

Example

See 10.3.2.27 object (231).

10.3.2.31 <ph>

A phrase is a small group of words that stand together as a unit, typically forming a component of a clause.

Usage information

The `<ph>` element often is used to enclose a phrase for reuse or conditional processing.

The `<ph>` element frequently is used as a specialization base, to create phrase-level markup that can provide additional semantic meaning or trigger specific processing or formatting. For example, all highlighting domain elements are specializations of `<ph>`.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and `@keyref` (388).

Example

The following code sample shows `<ph>` elements that are used for conditional processing:

```
<p>The Style menu is the <ph product="Software1000"/>third item</ph>
<ph product="Software9000"/>fourth item</ph> from the left on the menu bar.</p>
```

10.3.2.32 <pre>

Preformatted text is text that contains line breaks and spaces that are intended to be preserved at publication time.

Rendering expectations

Processors *SHOULD* preserve line the breaks and spaces that are present in the content of a `<pre>` element.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Display attributes (369), xml:space (377), and spectitle (specialization attributes) (376).

Example

The following code sample shows preformatted text that contains white space and line breaks. When the following code sample is published, the white space and line breaks are preserved.

```
<pre>
          MEMO: programming team fun day
Remember to bring a kite, softball glove, or other favorite
outdoor accessory to tomorrow's fun day outing at Zilker Park.
Volunteers needed for the dunking booth.
</pre>
```

10.3.2.33 <q>

A quotation is a small group of words that is taken from a text or speech and repeated by someone other than the original author or speaker.

Rendering expectations

Processors add appropriate styling, such as locale-specific quotation marks, around the contents of the `<q>` element and render it inline.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

In the following code sample, the <q> element contains a quotation. Note that no quotation marks are included; locale-specific quotation marks will be generated during processing.

```
<p>
George said, <q>Disengage the power supply before servicing the unit.</q>
</p>
```

10.3.2.34 <section>

A section is an organizational division in a topic. Sections are used to organize subsets of information that are directly related to the topic; they can have titles.

Usage information

Multiple sections within a single topic do not represent a hierarchy, but rather peer divisions of that topic. Sections cannot be nested.

Note For maximum flexibility in creating specialization, sections allow plain text as well as phrase and block level elements. Because of the way XML grammars are defined within a DTD, any element that allows plain text cannot restrict the order or frequency of other elements. As a result, the <section> element allows <title> to appear anywhere as a child of <section>. However, the intent of the specification is that <title> should only be used once in any <section>, and when used, it should precede any other text or element content.

Rendering expectations

Processors *SHOULD* treat the presence of more than one <title> element in a <section> element as an error.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and spectitle (specialization attributes) (376).

Example

The following code sample shows how element-reference topics in the DITA specification use titled sections to provide a consistent structure for grouping information:

```
<reference id="sub" xml:lang="en-us">
  <title>p</title>
  <shortdesc conkeyref="library-short-descriptions/p"/>
  <refbody>
    <section><title>Usage information</title>
      <p>...</p>
    </section>
    <section><title>Rendering expectations</title>
      <p>...</p>
    </section>
    <section><title>Processing expectations</title>
      <p>...</p>
    </section>
    <section><title>Specialization hierarchy</title>
      <p>...</p>
    </section>
    <section><title>Attributes</title>
      <p>...</p>
    </section>
    <example><title>Example</title>
```

```
      <p>...</p>
    </example>
  </refbody>
</reference>
```

10.3.2.35 <sectiondiv>

A section division is a grouping of sequential content within a section. There is no additional semantic meaning attached. It is useful primarily as a specialization base and for reuse.

Usage information

The `<sectiondiv>` element cannot contain a title. If a title is required, use nested topics.

The `<sectiondiv>` element can nest itself, so it can be used as a specialization base. Another common use case for the `<sectiondiv>` element is to group a sequence of related elements for reuse, so that another topic can reference the entire set with a single `@conref` or `@conkeyref` attribute.

The `<sectiondiv>` element can only be used within `<section>` elements. Use the `<div>` element to group content that might occur in both topic bodies and sections.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

In the example below, the `<sectiondiv>` element is used to group content that can be reused elsewhere.

```
<section>
  <title>Nice pets</title>
  <sectiondiv id="smallpets">
    <p>Cats are nice.</p>
    <p>Dogs are nice.</p>
    <p>Friends of mine really love their hedgehogs.</p>
  </sectiondiv>
  <sectiondiv id="biggerpets">
    <p>Lots of people want ponies when they grow up.</p>
    <p>Llamas are also popular.</p>
  </sectiondiv>
</section>
```

10.3.2.36 <sl>

A simple list is a list that contains a few items of short, phrase-like content.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), compact (371), and spectitle (specialization attributes) (376).

Example

In a topic that discusses related modules, the following markup could be used:

```
<section>
  <title>Messages</title>
  <p>Messages from the ags_open module are identical with messages from:</p>
  <sl>
    <sli>ags_read</sli>
```

```
    <sli>ags_write</sli>
    <sli>ags_close</sli>
  </sl>
</section>
```

10.3.2.37 <sli>

A simple list item is a component of a simple list. A simple list item contains a brief phrase or text content, adequate for describing package contents, for example.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.3.2.36 sl (239).

10.3.2.38 <term>

The `<term>` element identifies words that might have or require extended definitions or explanations.

Usage information

The `@keyref` attribute can be used to associate a term with a resource, typically a definition of the term. The `@keyref` attribute can also be used to supply the text content for `<term>` using standard `@keyref` processing for variable text.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and `@keyref` (388).

Example

The following code sample shows how the `<term>` element can be used:

```
<p>A <term>reference implementation</term> of DITA implements the standard,
fallback behaviors intended for DITA elements.</p>
```

10.3.2.39 <text>

The `<text>` element serves as a container for text. It has no associated semantics.

Usage information

The `<text>` element is primarily used as a specialization base or to enable reuse. The `<text>` can contain only text or nested `<text>` elements.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

In the following code sample, the <text> element is used to contain text that is intended to be reused:

```
<p>This an example of <text id="reuse">Text
   that is reusable</text>, with no extra
   semantics attached to the text.</p>
```

10.3.2.40 <tm>

The <tm> element identifies a term or phrase that is trademarked. Trademarks include registered trademarks, service marks, slogans, and logos.

Usage information

The business rules for indicating and displaying trademarks differ from company to company. These business rules can be enforced by either authoring policy or processing.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attributes defined below.

@tmtype (REQUIRED)
Specifies the trademark type. Allowable values are:

tm
Trademark

reg
Registered trademark

service
Service mark

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

@trademark
Specifies the trademarked term.

@tmowner
Specifies the trademark owner, for example "OASIS".

@tmclass
Specifies the classification of the trademark. This can be used to differentiate different groupings of trademarks.

Example

The following code sample shows how IBM uses the <tm> element:

```
<p>The advantages of using <tm trademark="DB2 Universal Database" tmtype="tm">
<tm trademark="DB2" tmtype="reg" tmclass="ibm">DB2</tm> Universal Database</tm> are
well known.</p>
```

10.3.2.41

An unordered list is a list in which the order of items is not significant.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), compact (371), and spectitle (specialization attributes) (376).

Example

The following code sample shows a list in which the order of items is unimportant:

```
<p>Here are the countries that I have visited:</p>
<ul>
 <li>Germany</li>
 <li>France</li>
 <li>Japan</li>
 <li>Mexico</li>
</ul>
```

10.3.2.42 <xref>

A cross reference is an inline link. A cross reference can link to a different location within the current topic, another topic, a specific location in another topic, or an external resource such as a PDF or Web page.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369), and `@keyref` (388).

Examples

This section contains examples of how the `<xref>` element can be used.

Figure 96: Cross reference to another topic, without link text

The following code sample shows a cross reference to another topic; link text is not provided. Processor typically use the topic title as the link text.

```
<p>Background information about DITA is provided in
<xref href="overview-of-dita.dita"/>.</p>
```

The same cross reference could be created using `@keyref` instead of `@href`; using `@keyref` allows the link to be redirected to different resources when the topic is used in different contexts.

Figure 97: Cross references with link text specified

The following code sample shows a cross reference that specifies link text:

```
<p>While this set of tutorials gives several simple examples of
<xref keyref="markup-examples">common DITA features</xref>, a comprehensive
```

```
list of DITA features is available in the DITA specification
<xref keyref="dita-conformance">conformance clause</xref>.</p>
```

Figure 98: Cross reference to a URI that contains an ampersand

The following code sample shows a cross reference that contains an ampersand:

```
<xref href="https://www.example.com/docview.wss?rs=757&context=SSVNX5"
scope="external" format="html">Part number SSVNX5</xref>
```

Because the `@href` attribute value needs to be a valid URI, the ampersand must be escaped, as shown in the revised code sample below:

```
<xref href="https://www.example.com/docview.wss?rs=757&amp;context=SSVNX5"
scope="external" format="html">Part number SSVNX5</xref>
```

Although the entity is in the DITA source, the entity might not show up when the link target is displayed in an editor or a Web browser; the URI might be shown as the following:

```
https://www.example.com/docview.wss?rs=757&context=SSVNX5
```

Related concepts

DITA addressing (73)

DITA provides two addressing mechanisms. DITA addresses either are direct URI-based addresses, or they are indirect key-based addresses. Within DITA documents, individual elements are addressed by unique identifiers specified on the `@id` attribute. DITA defines two fragment-identifier syntaxes; one is the full fragment-identifier syntax, and the other is an abbreviated fragment-identifier syntax that can be used when addressing non-topic elements from within the same topic.

Related reference

link (253)

A link is a reference to another DITA topic or a non-DITA resource.

10.3.3 Multimedia elements

The multimedia elements are used to reference audio or video content. The elements in this domain are modeled on the HTML5 `<audio>` and `<video>` elements.

10.3.3.1 <audio>

Audio is sound that the human ear is capable of hearing.

Usage information

The `<audio>` element is modeled on the HTML5 `<audio>` element.

An audio resource can be referenced by `@href`, `@keyref`, and nested `<media-source>` elements.

Behaviors such as auto-playing, looping, and muting are determined by attributes. When not specified, the default behavior is determined by the user agent that is used to present the media.

Comment by Kristen J Eberlein on 22 April 2019

If we keep the above wording, we need to define the term *user agent*.

Rendering expectations

When an audio resource cannot be rendered in a meaningful way, processors *SHOULD* present the contents of the `<fallback>` element, if it is present.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attributes defined below.

@autoplay
Specifies whether the resource automatically plays when it is presented. The following values are recognized: "true", "false", and "-dita-use-conref-target ". The default value is "true".

@controls
Specifies whether the presentation of the resource includes user interface controls. The following values are recognized: "true", "false", and "-dita-use-conref-target ". The default value is "true".

@format
Specifies the MIME type for the resource. This attribute enables processors to avoid loading unsupported resources. If `@format` is not specified and `@keyref` is specified, the effective type for the key named by the `@keyref` attribute is used as the value. If an explicit `@format` is not specified on either the `<audio>` element or key definition, processors can use other means, such the URI file extension, to determine the effective MIME type of the resource.

@href
Specifies the absolute or relative URI of the audio resource. If `@href` is specified, specify `@format` also.

@keyref
Specifies a key reference to the audio resource.

@loop
Specifies whether the resource loops when played. The following values are recognized: "true", "false", and "-dita-use-conref-target ". The default value is "true".

@muted
Specifies whether the resource is muted. The following values are recognized: "true", "false", and "-dita-use-conref-target ". The default value is "true".

@scope
The `@scope` attribute describes the closeness of the relationship between the current document and the target resource. Resources in the same information unit are considered `"local"`; resources in the same system as the referencing content but not part of the same information unit are considered `"peer"`; and resources outside the system, such as Web pages, are considered `"external"`.

@tabindex
Specifies whether the audio resource can be focused and where it participates in sequential keyboard navigation. See `@tabindex` in the HTML specification (WHATWG version).

Examples

Figure 99: An <audio> element that uses direct addressing

In the following code sample, an audio resource is referenced using direct addressing. The `@type` attribute specifies the MIME type of the audio resource.

```
<audio href="message.mp3" format="audio/mp3"/>
```

Figure 100: An <audio> element that uses indirect addressing

In the following code sample, the audio resource is addressed using a key reference:

```
<audio keyref="message"/>
```

Both the URI and the MIME type are specified on the key definition:

```
<keydef keys="message" href="message.mp3" format="audio/mp3"/>
```

Figure 101: An <audio> element with multiple formats

In the following code sample, `<media-source>` elements are used to specify the different audio formats that are available.

```
<audio>
  <media-source href="message.mp3" format="audio/mp3"/>
  <media-source href="message.wav" format="audio/wav"/>
</audio>
```

Figure 102: Example of a complex <audio> element

The following code sample specifies an audio resource and defines multiple presentational details; it also provides fallback behavior for when the audio resource cannot be rendered.

```
<audio autoplay="true"
               controls="true"
               loop="false"
               muted="false">
  <desc>A sound file narrating the performance of this procedure.</desc>
  <fallback>The audio track walking through this procedure is not available.</fallback>
  <!-- Multiple formats, with URI and MIME type referenced using a key -->
  <media-source keyref="walkthrough-mp3"/>
  <media-source keyref="walkthrough-wav"/>
</audio>
```

10.3.3.2 <media-source>

The media source specifies the location of an audio or video resource.

Usage information

The media source is modeled on the `<source>` element used in HTML5 media elements.

Rendering expectations

When multiple `<media-source>` elements are present, the user agent evaluates them in document order and selects the first resource that can be played.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attributes defined below.

@format
Specifies the format of the resource being addressed.

@href
Specifies the URI of the media resource.

@keyref
Specifies a key reference to the media resource.

@scope
The `@scope` attribute describes the closeness of the relationship between the current document and the target resource. Resources in the same information unit are considered `"local"`; resources in the same system as the referencing content but not part of the same information unit are considered `"peer"`; and resources outside the system, such as Web pages, are considered `"external"`.

Example

See 10.3.3.1 audio (243) and 10.3.3.4 video (247).

10.3.3.3 <media-track>

Media track settings specify the location of supplemental text-based data for the referenced media, for example, subtitles or descriptions.

Usage information

The media track settings are modeled on the `<track>` element used in HTML5 media elements. They refer to track resources that use Web Video Text Track Format (WebVTT).

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attributes defined below.

@format
Specifies the format of the resource being addressed.

@href
Specifies the URI of the track resource.

@keyref
Specifies a key reference to the track resource.

@kind
Specifies the usage for the track resource. This attribute is modeled on the `@kind` attribute on the HTML5 `<track>` element, as described by the *HTML specification, WHATWG version*. The values for this attribute are derived from the HTML5 standard:

captions
Transcription or translation of the dialogue, sound effects, relevant musical cues, and other relevant audio information. This is intended for use when the soundtrack is unavailable, for example, because it is muted or because the user is hard-of-hearing. This information is rendered over the video and labeled as appropriate for hard-of-hearing users.

chapters
Chapter titles, which are intended to be used for navigating the media resource. The chapter titles are rendered as an interactive list in the interface for the user agent.

descriptions
Textual descriptions of the video component of the media resource. This is intended for audio synthesis when the visual component is unavailable, for example, because the user is interacting with the application without a screen or because the user is blind. Descriptions are synthesized as separate audio tracks.

metadata
Tracks intended for use from script. This metadata is not displayed by the user agent.

subtitles
Transcription or translation of the dialogue, suitable for when the sound is available but not understood, for example, because the user does not understand the language of the soundtrack. Subtitles are rendered over the video.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

@scope
The `@scope` attribute describes the closeness of the relationship between the current document and the target resource. Resources in the same information unit are considered `"local"`; resources in the same system as the referencing content but not part of the same information unit are considered `"peer"`; and resources outside the system, such as Web pages, are considered `"external"`.

@srclang
Specifies the language of the track resource.

Example

See 10.3.3.4 video (247).

10.3.3.4 <video>

A video is a recording of moving visual images.

Usage information

The `<video>` element is modeled on the HTML5 `<video>` element.

A video resource can be referenced by `@href`, `@keyref`, and nested `<media-source>` elements.

Behaviors such as auto-playing, looping, and muting are determined by attributes. When not specified, the default behavior is determined by the user agent that is used to present the media.

Rendering expectations

The video resource typically is rendered in the main flow of the content.

Processors *SHOULD* scale the video resource when values are provided for the `@height` and `@width` attributes. The following expectations apply:

- If a height value is specified and no width value is specified, processors *SHOULD* scale the width by the same factor as the height.
- If a width value is specified and no height value is specified, processors *SHOULD* scale the height by the same factor as the width.

- If both a height value and width value are specified, implementations *MAY* ignore one of the two values when they are unable to scale to each direction using different factors.

When a video resource cannot be rendered in a meaningful way, processors *SHOULD* render the contents of the `<fallback>` element, if it is present.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attributes defined below.

Comment by rodaande
This list was significantly reworked when the element moved into the base vocabulary; need to validate that the list is still correct for LwDITA.

@autoplay
Specifies whether the resource automatically plays when it is presented. The following values are recognized: "true", "false", and "-dita-use-conref-target ". The default value is "true".

@controls
Specifies whether the presentation of the resource includes user interface controls. The following values are recognized: "true", "false", and "-dita-use-conref-target ". The default value is "true".

@format
Specifies the MIME type for the resource. This attribute enables processors to avoid loading unsupported resources. If `@format` is not specified and `@keyref` is specified, the effective type for the key named by the `@keyref` attribute is used as the value. If an explicit `@format` is not specified on either the `<video>` element or key definition, processors can use other means, such the URI file extension, to determine the effective MIME type of the resource.

@height
Indicates the vertical dimension for the resulting display. The value of this attribute is a real number (expressed in decimal notation) optionally followed by a unit of measure from the set of cm, em, in, mm, pc, pt, px, and Q (centimeters, ems, inches, picas, points, pixels, millimeters, and quarter-millimeters, respectively). The default unit is px (pixels). Possible values include: "5", "5in", and "10.5cm".

@href
Specifies the absolute or relative URI of the video resource. If `@href` is specified, specify `@format` also.

@keyref
Specifies a key reference to the video resource.

@loop
Specifies whether the resource loops when played. The following values are recognized: "true", "false", and "-dita-use-conref-target ". The default value is "true".

@muted
Specifies whether the resource is muted. The following values are recognized: "true", "false", and "-dita-use-conref-target ". The default value is "true".

@poster
Specifies the absolute or relative URI of the image that is rendered before video playback begins.

@posterkeyref
Specifies a key reference for the poster image.

@scope

The `@scope` attribute describes the closeness of the relationship between the current document and the target resource. Resources in the same information unit are considered `"local"`; resources in the same system as the referencing content but not part of the same information unit are considered `"peer"`; and resources outside the system, such as Web pages, are considered `"external"`.

@tabindex

> **Comment by rodaande**
> Need to add the linked version of the HTML spec into our resources.

Specifies whether the video resource can be focused and where it participates in sequential keyboard navigation. See `@tabindex` in the HTML specification (WHATWG version).

@width

Indicates the horizontal dimension for the resulting display. The value of this attribute is a real number (expressed in decimal notation) optionally followed by a unit of measure from the set of cm, em, in, mm, pc, pt, px, and Q (centimeters, ems, inches, picas, points, pixels, millimeters, and quarter-millimeters, respectively). The default unit is px (pixels). Possible values include: "5", "5in", and "10.5cm".

Examples

This section contains examples of how the `<video>` element can be used.

Figure 103: A <video> element that uses direct addressing

In the following code sample, a video resource is referenced using direct addressing. The `@format` attribute specifies the MIME type of the video.

```
<video href="video.mp4" format="video/mp4"/>
```

Figure 104: A <video> element that uses indirect addressing

In the following code sample, the video resource is addressed using a key reference:

```
<video keyref="video"/>
```

Both the URI and the MIME type are specified on the key definition:

```
<keydef keys="video" href="video.mp4" format="video/mp4"/>
```

Figure 105: A <video> element with multiple formats

In the following code sample, `<media-source>` elements are used to specify the different video formats that are available.

```
<video>
  <media-source href="video.mp4" format="video/mp4"/>
  <media-source href="video.ogg" format="video/ogg"/>
```

```
  <media-source href="video.webm" format="video/webm"/>
</video>
```

Figure 106: Example of a <video> element with multiple formats and multilingual subtitles

The following code sample defines multiple presentational details for a video that is available in multiple formats. The video is referenced using key reference and a fallback image is provided for use when the video cannot be displayed.

```
<video height="300px"
               loop="false"
               muted="false"
               poster="demo1-video-poster"
               width="400px">
  <desc>A video illustrating this procedure.</desc>
  <fallback>
    <image href="video-not-available.png">
      <alt>This video cannot be displayed.</alt>
    </image>
  </fallback>
  <!-- Multiple formats, referenced via key. The key definition
       specifies both the URI and the MIME type -->
  <media-source keyref="demo1-video-mp4"/>
  <media-source keyref="demo1-video-ogg"/>
  <media-source keyref="demo1-video-webm"/>
  <!-- Subtitle tracks in English, French and German.
       Each key definition provides a URI and sets type="subtitles". -->
  <media-track srclang="en" keyref="demo1-video-subtitles-en"/>
  <media-track srclang="fr" keyref="demo1-video-subtitles-fr"/>
  <media-track srclang="de" keyref="demo1-video-subtitles-de"/>
</video>
```

10.3.4 Indexing elements

The indexing elements provide content that a processor can use to generate an index.

10.3.4.1 <index-see>

An `<index-see>` element directs the reader to an index entry that the reader should use instead of the current one.

Usage information

There can be multiple `<index-see>` elements within an `<indexterm>` element.

Processing expectations

Processors should ignore an `<index-see>` element if its parent `<indexterm>` element contains any `<indexterm>` children.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and `@keyref` (388).

Examples

This section contains examples of how <index-see> elements can be used.

Figure 107: Use of an <index-see> element

The following code sample shows how an <index-see> element is used to refer readers to the preferred term:

```
<indexterm>Carassius auratus
  <index-see>goldfish</index-see>
</indexterm>
```

This markup will generate an index entry without a page reference.

Figure 108: Use of an <index-see> element to redirect to a multi-level index entry

The following code sample shows how an <index-see> is used to redirect to a multilevel index entry:

```
<indexterm>feeding goldfish
  <index-see>goldfish
    <indexterm>feeding</indexterm>
  </index-see>
</indexterm>
```

10.3.4.2 <index-see-also>

An <index-see-also> element directs the reader to an index entry that the reader should use in addition to the current one.

Usage information

There can be multiple <index-see-also> elements within a single <indexterm> element.

Processing expectations

Processors should ignore an <index-see-also> element if its parent <indexterm> element contains any <indexterm> children.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and @keyref (388).

Examples

This section contains examples of how <index-see-also> elements can be used.

Figure 109: Use of an <index-see-also> element

The following code sample shows the use of an <index-see-also> element to generate a "see also" reference to the index entry for "goldfish".

```
<indexterm>carp
  <index-see-also>goldfish</index-see-also>
</indexterm>
```

This markup generates a primary index entry for "carp" and a redirection that instructs the reader to "see also goldfish".

Figure 110: Use of an <index-see-also> element to redirect to a multilevel index entry

The following code sample shows the use of an `<index-see-also>` element to redirect to a multilevel `<indexterm>` element:

```
<indexterm>feeding
  <index-see-also>goldfish
    <indexterm>feeding</indexterm>
  </index-see-also>
</indexterm>
```

10.3.4.3 <indexterm>

An `<indexterm>` element contains content that is used to produce an index entry in a generated index. Nested `<indexterm>` elements create multi-level indexes.

Rendering expectations

The content of `@indexterm` entries is not rendered in the flow of body text; it is rendered only as part of a generated index.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), `@keyref` (388), and the attributes defined below.

@start
Specifies an identifier that indicates the start of an index range.

@end
Specifies an identifier that indicates the end of an index range.

Examples

This section contains examples of how `<indexterm>` elements can be used.

Figure 111: Index reference to a point within in a topic

When index entries are placed in the body of a topic, they serve as point references to their location in the topic.

In the following code sample, the `<indexterm>` element provides a point reference to the beginning of the paragraph.

```
<p><indexterm>databases</indexterm>Databases are used to ...</p>
```

Figure 112: Index entries within topic prologues or DITA maps

When index entries are located within the `<prolog>` element in a topic or the `<topicmeta>` element in a DITA map, they serve as point references to the start of the topic title.

In the following code sample, the `<indexterm>` element provides a reference to the topic as a whole; the generated index entry is associated with the start of the `<title>` element.

```
<topic id="about-databases">
  <title>About databases</title>
  <prolog>
    <metadata>
```

```
      <keywords>
        <indexterm>databases</indexterm>
      </keywords>
    </metadata>
  </prolog>
  <body>
    <!-- content... -->
  </body>
</topic>
```

The effect is the same as if the <indexterm> element had been located in the map:

```
<map>
  <topicref href="aboutdatabases.dita">
    <topicmeta>
      <keywords>
        <indexterm>databases</indexterm>
      </keywords>
    </topicmeta>
  </topicref>
  <!-- ... -->
</map>
```

Figure 113: A simple index range

A simple index range will look something like this:

```
<indexterm start="cheese">cheese</indexterm>
<!-- ... additional content -->
<indexterm end="cheese"/>
```

This markup will generate a top-level index term for "cheese" that covers a series of pages, such as:

cheese 18-24

Figure 114: A more complex index range

Specifying a range for nested terms is similar. In this sample, the range is specified for the tertiary index entry "pecorino":

```
<indexterm>cheese
  <indexterm>sheeps milk
    <indexterm start="level-3-pecorino">pecorino</indexterm>
  </indexterm>
</indexterm>
 <!-- ... additional content ... -->
<indexterm end="level-3-pecorino"/>
```

10.3.5 Related links elements

Related links are topic-to-topic connections or connections from DITA topics to non-DITA resources. These embedded references in DITA topics establish dependencies from the topics to the referenced resources.

10.3.5.1 <link>

A link is a reference to another DITA topic or a non-DITA resource.

Processing expectations

When displayed, links typically are sorted based on their attributes, which define the type or role of the link target in relation to the current topic.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369), `@keyref` (388), and 10.8.3.10 The role and otherrole attributes (390).

Example

The following code sample shows a simple collection of links in a DITA topic:

```
<related-links>
  <link href="covid-19.dita"/>
  <link href="covid-19-testing.dita"/>
  <link format="html" href="covid-19-nc.html">
    <linktext>COVID-19 in North Carolina</linktext>
  </link>
  <link format="html" href="239fh49.html#resources">
    <linktext>Public health resources in Durham, NC</linktext>
    <desc>When you work as a contact tracer, you need to know ...</desc>
  </link>
</related-links>
```

Related concepts

DITA addressing (73)

DITA provides two addressing mechanisms. DITA addresses either are direct URI-based addresses, or they are indirect key-based addresses. Within DITA documents, individual elements are addressed by unique identifiers specified on the `@id` attribute. DITA defines two fragment-identifier syntaxes; one is the full fragment-identifier syntax, and the other is an abbreviated fragment-identifier syntax that can be used when addressing non-topic elements from within the same topic.

Related reference

xref (242)

A cross reference is an inline link. A cross reference can link to a different location within the current topic, another topic, a specific location in another topic, or an external resource such as a PDF or Web page.

10.3.5.2 <linkinfo>

The `<linkinfo>` element specifies a paragraph that describes or provides additional information about the links that are contained in a `<linklist>` element.

Processing expectations

Comment by Kristen J Eberlein on 01 June 2018

What processing expectations do we have for this element? I'm assuming that we expect processors to render the content somewhere and probably in a paragraph format, based on the short description.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows how the <linkinfo> element provides additional information about the links related to "Repairing widgets":

```
<linklist>
  <title>Repairing widgets</title>
  <link href="debug.dita" type="task"/>
  <link href="repair.dita" type="task"/>
  <link href="test.dita" type="task"/>
  <linkinfo>To repair a reciprocating widget,follow the instructions carefully
            and in the specified order.</linkinfo>
</linklist>
```

10.3.5.3 <linklist>

A link list is an author-ordered group of links.

Usage information

Attributes that are set on the <linklist> element cascade to the contained links.

Rendering expectations

When rendering links, processors *MUST* preserve the order of links that are specified within <linklist> elements.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), collection-type (common map attributes) (371), 10.8.3.10 The role and otherrole attributes (390), spectitle (specialization attributes) (376), and the attributes defined below. This element also uses @format, @scope, and @type from Link-relationship attributes (369).

@duplicates
Specifies whether duplicate links are removed from a group of links. Duplicate links are links that address the same resource using the same properties, such as link text and link role. How duplicate links are determined is processor-specific. The following values are permitted:

yes
Specifies that duplicate links are retained.

no
Specifies that duplicate links are removed.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

The suggested processing default is "yes" within <linklist> elements and "no" for other links.

@collection-type
See collection-type (common map attributes) (371) for a full definition and list of supported values.

Example

The following code sample shows how the <linklist> element is used to construct an author-defined group of links. The @format and @scope attributes are set on the <linklist> element and cascade to the contained links. The order of links is preserved in the output.

```
<related-links>
  <linklist format="html" scope="external">
```

```
    <link href="http://www.example.org">
      <linktext>Example 1</linktext>
    </link>
    <link href="http://www.example.com">
      <linktext>Example 2</linktext>
    </link>
  </linklist>
</related-links>
```

10.3.5.4 <linkpool>

A link pool is a group of links; the order that the links are rendered in the output is determined by the processor.

Usage information

Attributes that are set on the `<linkpool>` element cascade to the contained links.

Rendering expectations

The order in which links in a `<linkpool>` element are rendered is processor-specific. A processor might order links based on role or type. A processor might move or remove links based on the context; for example, prerequisite links might be rendered at the beginning of a Web page, or links to the next topic might be removed if the two topics are rendered on the same page in a PDF. Processors might automatically sort some links, while others are left ordered as authored.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), collection-type (common map attributes) (371), 10.8.3.10 The role and otherrole attributes (390), and the attributes defined below. This element also uses `@format`, `@scope`, and `@type` from Link-relationship attributes (369).

@duplicates
: Specifies whether duplicate links are removed from a group of links. Duplicate links are links that address the same resource using the same properties, such as link text and link role. How duplicate links are determined is processor-specific. The following values are permitted:

  yes
  : Specifies that duplicate links are retained.

  no
  : Specifies that duplicate links are removed.

  -dita-use-conref-target
  : See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

  The suggested processing default is "yes" within `<linklist>` elements and "no" for other links.

@collection-type
: See collection-type (common map attributes) (371) for a full definition and list of supported values.

Example

The following code sample shows how a `<linkpool>` element is used to group a set of conceptual information. The order in which the links are rendered in the output is processor-dependent.

```
<related-links>
  <linkpool type="concept">
    <link href="czez.dita#czez" role="next"/>
```

```
    <link href="czunder.dita"/>
    <link format="html" href="czover.htm#sqljsupp" role="parent">
      <linktext>Overview of the CZ</linktext>
    </link>
    <link format="html" href="czesqlj.htm#sqljemb">
      <linktext>Working with CZESQLJ</linktext>
      <desc>When you work with CZESQLJ, you need to know...</desc>
    </link>
  </linkpool>
</related-links>
```

10.3.5.5 <linktext>

Link text is the label for a link or resource.

Usage information

The `<linktext>` element can provide descriptive text for a link when the target cannot be resolved during processing, for example, as when the link is to a peer, external, or non-DITA resource. When used in conjunction with a local DITA resource, the content of the `<linktext>` is used as a label for the generated link.

Comment by Kristen J Eberlein on 31 October 2020

Is the above correct and adequately resolves your draft comment from 2018?

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Examples

This section contains examples of how the `<linktext>` element can be used.

Figure 115: Link text within a topic

The following code sample shows how a `<linktext>` element can be used within a topic to provide text for a related link to a non-DITA resource:

```
<related-links>
  <link href="SQLJ-example.html" format="html" scope="local">
    <linktext>Accessing relational data with SQLJ</linktext>
  </link>
</related-links>
```

10.3.6 Table elements

DITA topics support two types of tables. The `<table>` element uses the OASIS Exchange Table Model (formerly known as the CALS table model). The OASIS table supports the spanning of multiple rows or columns for special layout or organizational needs, and provides a wide variety of controls over the display properties of the data and even the table structure itself.

The other table structure in DITA is called `<simpletable>`. As the name implies, it is structurally less complex than the OASIS table, and it can be used as a very simple, regular table for which close control of formatting is not as important. The main advantage of `<simpletable>` is for describing lists of data with regular headings, such as telephone directory listings, display adapter configuration data, or API properties.

10.3.6.1 <colspec>

A column specifications provides information about a single column in a table from on the OASIS Exchange Table Model, such as a column name and number, cell content alignment, or a column width.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (without the Metadata attribute group), `@base` from the Metadata attributes (366), and the attributes defined below. This element also uses `@align`, `@char`, `@charoff`, `@colsep`, `@rowsep`, and `@rowheader` from Complex table attributes (369).

@colnum
: Indicates the number of a column in the table, counting from the first logical column to the last column.

@colname
: Specifies a name for the column defined by this element. The `<entry>` element can use `@colname` to refer to the name of this column.

@colwidth
: Describes the column width.

Example

See 10.3.6.8 table (262).

10.3.6.2 <entry>

A table entry represents a single cell in a table based on the OASIS Exchange Table Model.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (without the Metadata attribute group), `@base` and `@rev` from the Metadata attributes (366), and the attributes defined below. This element also uses `@align`, `@char`, `@charoff`, `@colsep`, `@rowsep`, and `@valign` from the Complex table attributes (369).

@rotate
: Specifies whether the contents of the entry is rotated. Supported values are:

  1
  : The contents of the cell are rotated 90 degrees counterclockwise.

  0
  : No rotation occurs.

  -dita-use-conref-target
  : See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

  If this attribute is not specified, no rotation occurs. In situations where a stylesheet or other formatting mechanism specifies table cell orientation, the `@rotate` attribute can be ignored.

@colname
: Specifies the column name in which an entry is found. The value is a reference to the `@colname` attribute on the `<colspec>` element.

@namest
: Specifies the first logical column that is included in a horizontal span. The value is a reference to the `@colname` attribute on the `<colspec>` element.

@nameend
Specifies the last logical column that is included in a horizontal span. The value is a reference to the `@colname` attribute on the `<colspec>` element.

@morerows
Specifies the number of additional rows to add in a vertical span.

@scope
Specifies that the current entry is a header for other table entries. Allowable values are:

row
The current entry is a header for all cells in the row.

col
The current entry is a header for all cells in the column.

rowgroup
The current entry is a header for all cells in the rows spanned by this entry.

colgroup
The current entry is a header for all cells in the columns spanned by this entry.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

@headers
Specifies one or more `<entry>` headers that apply to the current entry. The `@headers` attribute contains an unordered set of unique space-separated tokens, each of which is an ID reference of an entry from the same table.

Example

See 10.3.6.8 table (262).

10.3.6.3 <row>

A table row is a single row in a table based on the OASIS Exchange Table Model.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), along with `@rowsep` and `@valign` from Complex table attributes (369).

Example

See 10.3.6.8 table (262).

10.3.6.4 <simpletable>

Simple tables are a basic tabular environment to present organized content.

Usage information

Choose the `<simpletable>` element when control of tabular formatting is not as important. `<simpletable>` is designed for closer compatibility with HTML5 tables. For example, multi-column tabular data such as phone directory listings or parts lists are good candidates for `<simpletable>`. Another good use of `<simpletable>` is for information that seems to beg for a three-part definition list; the `@keycol` attribute can be used to indicate which column represents the "key" or term-like column of the structure.

The `@keycol` attribute indicates which column represents the "key" or term-like column of the structure.

The close match of `<simpletable>` to tabular, regular data makes `<simpletable>` suitable as the basis for specialized structures such as `<properties>` (for programming information) and choice tables (for tasks).

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Display attributes (369), Simpletable attributes (369), and spectitle (specialization attributes) (376).

Examples

Figure 116: Example of a simple table

The following code sample shows a simple table that is used to represent a truth table from Boolean logic:

```
<simpletable>
  <sthead>
    <stentry>P</stentry>
    <stentry>not P</stentry>
  </sthead>
  <strow>
    <stentry>true</stentry>
    <stentry>false</stentry>
  </strow>
  <strow>
    <stentry>false</stentry>
    <stentry>true</stentry>
  </strow>
</simpletable>
```

Figure 117: Example with column and row spanning

The following code sample shows a simple table that tracks meals:

```
<simpletable>
  <title>Food log for Wednesday</title>
  <sthead>
    <stentry>Meal</stentry>
    <stentry>Food</stentry>
  </sthead>
  <strow>
    <stentry colspan="2">Fasting period</stentry>
  </strow>
  <strow>
    <stentry>Lunch</stentry>
    <stentry rowspan="2">Pasta</stentry>
  </strow>
  <strow>
    <stentry>Dinner</stentry>
  </strow>
</simpletable>
```

Figure 118: Example using @keycol

The following code sample shows how the `@keycol` attribute can be used. The value of the `@keycol` attribute specifies that the first column is the header column. This indicates that items in the first column are headers for the row. Rendering of the header column is left up to the implementation.

```
<simpletable keycol="1">
 <sthead>
  <stentry>Term</stentry>
  <stentry>Categorization</stentry>
```

```
   <stentry>Definition</stentry>
  </sthead>
  <strow>
   <stentry>Widget</stentry>
   <stentry>noun</stentry>
   <stentry>Thing that is used for something</stentry>
  </strow>
  <strow>
   <stentry>Frustration</stentry>
   <stentry>noun</stentry>
   <stentry>What you feel when you drop the widget</stentry>
  </strow>
 </simpletable>
```

10.3.6.5 <stentry>

A simple table entry represents a single cell within a simple table.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), specentry (specialization attributes) (376), and the attributes defined below.

@colspan
Specifies the number of columns that a cell is to span inside a simple table.

@rowspan
Specifies the number of rows that a cells is to span inside a simple table.

@scope
Specifies that the current entry is a header for other table entries. Allowable values are:

row
The current entry is a header for all cells in the row.

col
The current entry is a header for all cells in the column.

rowgroup
The current entry is a header for all cells in the rows spanned by this entry.

colgroup
The current entry is a header for all cells in the columns spanned by this entry.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

@headers
Specifies one or more `<entry>` headers that apply to the current entry. The `@headers` attribute contains an unordered set of unique space-separated tokens, each of which is an ID reference of an entry from the same table.

Example

See 10.3.6.4 simpletable (259).

10.3.6.6 <sthead>

A simple table header is an optional header row for a simple table.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.3.6.4 simpletable (259).

10.3.6.7 <strow>

A simple table row is a single row in a simple table.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.3.6.4 simpletable (259).

10.3.6.8 <table>

A table based on the OASIS Exchange Table Model organizes arbitrarily complex relationships of tabular information. This standard table markup allows column or row spanning and table captions or descriptions.

Usage information

The DITA table is based on the OASIS Exchange Table Model, augmented with DITA attributes that enable it for accessibility, specialization, conref, and other DITA processing. An optional `<title>` inside the `<table>` element provides a caption to describe the table. In addition, the optional `<desc>` element enables table description that is parallel with figure description.

See 10.3.6.4 simpletable (259) for a simplified table model that is closer to the HTML5 table model, and can be specialized to represent more regular relationships of data.

In DITA tables, in place of the `@expanse` attribute used by other DITA elements, the `@pgwide` attribute is used in order to conform to the OASIS Exchange Table Model. The `@pgwide` attribute has a similar semantic ("1"=page width; "0"=resize to galley or column).

> **Note** The `@scale` attribute represents a stylistic markup property that is currently maintained in tables for legacy purposes. External stylesheets should enable less dependency on this attribute. Use the `@scale` attribute judiciously.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), `@frame` and `@scale` from Display attributes (369), and the attributes defined below. This element also uses `@colsep`, `@rowsep`, and `@rowheader` from Complex table attributes (369).

Comment by Robert
Looking for consistency on orient, pgwide, and entry/@rotate. Should they mention "not enforced by dtd / rng"? Should they say "Supported values" (seems to imply "processors MUST support at least these") or "Allowed values" (implies no other values are legal)?

@orient
Specifies the orientation of the table in page-based outputs. This attribute is primarily useful for print-oriented display. Allowable values are:

port
The same orientation as the text flow.

land
90 degrees counterclockwise from the text flow.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

In situations where a stylesheet or other formatting mechanism specifies table orientation based on other criteria, or for non-paginated outputs, the `@orient` attribute can be ignored.

@pgwide
Specifies the horizontal placement of the element. Supported values are 1 and 0, although these are not mandated by the DTD or RNG grammar files.

For print-oriented display, the value "1" places the element on the left page margin; "0" aligns the element with the left margin of the current text line and takes indention into account.

Example

Figure 119: DITA source

The following code sample shows a table that is used to provide reference information about animals and gestation:

```
<table>
<tgroup cols="2">
<colspec colwidth="121*"/>
<colspec colwidth="76*"/>
<thead>
<row>
<entry colname="COLSPEC0" valign="top">Animal</entry>
<entry colname="COLSPEC1" valign="top">Gestation</entry>
</row>
</thead>
<tbody>
<row>
<entry>Elephant (African and Asian)</entry>
<entry>19-22 months</entry>
</row>
<row>
<entry>Giraffe</entry>
<entry>15 months</entry>
</row>
<row>
<entry>Rhinoceros</entry>
<entry>14-16 months</entry>
</row>
<row>
<entry>Hippopotamus</entry>
<entry>7 1/2 months</entry>
</row>
</tbody>
</tgroup>
</table>
```

The formatted output might be displayed as follows:

Animal	Gestation
Elephant (African and Asian)	19-22 months
Giraffe	15 months
Rhinoceros	14-16 months
Hippopotamus	7 1/2 months

In this example, the use of the <thead> element for the header allows processors or screen readers to identify a header relationship between any cell in the table body and the matching header cell above that column.

Example: Complex table with implied accessibility markup

Comment by Robert
Discussed with Kris October 5 2020: would be good to have a section on accessibility in the architectural spec; if that is created, some of these examples might be better placed there.

In the following example, the table uses <thead> to identify header rows and @rowheader to identify a header column. This header relationship can be used to automatically create renderings of the table in other formats, such as HTML, that can be navigated using a screen reader or other assistive technology.

Figure 120: DITA source

```
<table frame="all" rowheader="firstcol">
  <title>Sample of automated table accessibility</title>
<desc>Names are listed in the column c1. Points are listed in both data columns, with
expected points in column c2 and actual points in column c3.</desc>
  <tgroup cols="3">
    <colspec colname="c1"/>
    <colspec colname="c2"/>
    <colspec colname="c3"/>
    <thead>
      <row>
        <entry morerows="1">Name</entry>
        <entry namest="c2" nameend="c3">points</entry>
      </row>
      <row>
        <entry>expected</entry>
        <entry>actual</entry>
      </row>
    </thead>
    <tbody>
      <row>
        <entry>Mark</entry>
        <entry>10,000</entry>
        <entry>11,123.45</entry>
      </row>
      <row>
        <entry>Peter</entry>
        <entry>9,000</entry>
        <entry>11,012.34</entry>
      </row>
      <row>
        <entry>Cindy</entry>
        <entry>10,000</entry>
        <entry>10,987.64</entry>
      </row>
     </tbody>
   </tgroup>
</table>
```

In this sample, navigation information for assistive technology is derived from two sources:

- The <thead> element contains two rows, and indicates that each <entry> in those rows is a header cell for that column. This means that each body cell can be associated with the header cell or cells above the column. For example, in the second body row, the entry "Peter" can be associated with the header "Name"; similarly, the entry "9,000" can be associated with the headers "expected" and "points".
- The @rowheader attribute implies that the first column plays a similar roll as a header. This means that each body cell in columns two and three can be associated with the header cell in column one. For example, in the second body row, the entry "9,000" can be associated with the header "Peter".

As a result of these two sets of headers, a rendering of the table can associate the entry "9,000" with three headers: "Peter", "expected", and "points", making it fully navigable by a screen reader or other assistive technology.

The formatted output might be displayed as follows:

Table 4: Sample of automated table accessibility

Names are listed in the column c1. Points are listed in both data columns, with expected points in column c2 and actual points in column c3.

Name	**points**	
	expected	**actual**
Mark	10,000	11,123.45
Peter	9,000	11,012.34
Cindy	10,000	10,987.64

Complex table with some manually specified accessibility markup

In some complex tables, the `<thead>` element and `@rowheader` attribute might not be enough to support all accessibility needs. Assume that the table above is flipped so that the names are listed across the top row, instead of in the first column, as follows:

Table 5: Sample with two header columns

Name		**Mark**	**Peter**	**Cindy**
points	**expected**	10,000	9,000	10,000
	actual	11,123.45	11,012.34	10,987.64

In this case the `@rowheader` attribute cannot be used, because it is only able to specify the first column as a header column. In this case, the `@scope` attribute can be used to indicate that entries in the first and second columns function as headers for the entire row (or row group, in the case of a cell that spans more than one row). The following code sample demonstrates the use of `@scope` to facilitate navigation of these rows by a screen reader or other assistive technology; note that the `<thead>` element is still used to imply a header relationship with the names at the top of each column.

```
<table frame="all">
 <title>Sample with two header columns</title>
 <tgroup cols="5">
  <colspec colname="c1"/>
  <colspec colname="c2"/>
  <colspec colname="c3"/>
  <colspec colname="c4"/>
  <colspec colname="c5"/>
  <thead>
   <row>
    <entry namest="c1" nameend="c2">Name</entry>
    <entry>Mark</entry>
    <entry>Peter</entry>
    <entry>Cindy</entry>
   </row>
  </thead>
  <tbody>
   <row>
    <entry morerows="1" scope="rowgroup"><b>points</b></entry>
    <entry scope="row"><b>expected</b></entry>
    <entry>10,000</entry>
    <entry>9,000</entry>
    <entry>10,000</entry>
```

```
   </row>
   <row>
    <entry scope="row"><b>actual</b></entry>
    <entry>11,123.45</entry>
    <entry>11,012.34</entry>
    <entry>10,987.64</entry>
   </row>
  </tbody>
 </tgroup>
</table>
```

Example: complex table with manual accessibility markup

In extremely complex tables, such as those with a single header cell in the middle of the table, extremely fine grained accessibility controls are available to explicitly associate any content cell with any header cell. This might also be useful for cases where processors do not support the implied accessibility relationships described above.

In the following sample, header cells are identified using the `@id` attribute, which is referenced using the `@headers` attribute on appropriate content cells. This makes all header relationships in the table explicit. Note that this sample ignores the `@scope` attribute, which could be used to exercise manual control without setting as many attribute values; it also ignores the fact that `<thead>` creates a header relationship even when the `@id` and `@headers` attributes are not used.

Figure 121: DITA source

```
<table frame="all">
  <title>Sample with fully manual accessibility control</title>
<desc>Names are listed in the column c1. Points are listed in both data columns, with
expected points in column c2 and actual points in column c3.</desc>
  <tgroup cols="3">
    <colspec colname="c1"/>
    <colspec colname="c2"/>
    <colspec colname="c3"/>
    <thead>
      <row>
        <entry morerows="1"> </entry>
        <entry namest="c2" nameend="c3" id="pts">points</entry>
      </row>
      <row>
        <entry id="exp" headers="pts">expected</entry>
        <entry id="act" headers="pts">actual</entry>
      </row>
    </thead>
    <tbody>
      <row>
        <entry id="name1">Mark</entry>
        <entry headers="name1 exp pts">10,000</entry>
        <entry headers="name1 act pts">11,123.45</entry>
      </row>
      <row>
        <entry id="name2">Peter</entry>
        <entry headers="name2 exp pts">9,000</entry>
        <entry headers="name2 act pts">11,012.34</entry>
      </row>
      <row>
        <entry id="name3">Cindy</entry>
        <entry headers="name3 exp pts">10,000</entry>
        <entry headers="name3 act pts">10,987.64</entry>
      </row>
     </tbody>
   </tgroup>
</table>
```

The formatted output might be displayed as follows:

Table 6: Sample with fully manual accessibility control

Names are listed in the column c1. Points are listed in both data columns, with expected points in column c2 and actual points in column c3.

	points	
	expected	**actual**
Mark	10,000	11,123.45
Peter	9,000	11,012.34
Cindy	10,000	10,987.64

10.3.6.9 <tbody>

A table body is a group of rows in a table based on the OASIS Exchange Table Model.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and `@valign` from Complex table attributes (369).

Example

See 10.3.6.8 table (262).

10.3.6.10 <tgroup>

A table group is a grouping of a table header and table body, based on the OASIS Exchange Table Model.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attribute defined below. This element also uses `@colsep`, `@rowsep`, and `@align` from Complex table attributes (369).

@cols (REQUIRED)
Indicates the number of columns in a `<tgroup>`.

Example

See 10.3.6.8 table (262).

10.3.6.11 <thead>

A table header contains one or more header rows in a table based on the OASIS Exchange Table Model.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and `@valign` from Complex table attributes (369).

Example

See 10.3.6.8 table (262).

10.4 Map elements

Map elements include the core components of DITA maps, such as `<topicref>` and `<reltable>`, as well as general purpose map specializations in the map group domain.

10.4.1 Basic map elements

DITA maps are built from a few core elements that are used for referencing and organizing topics. The `<topicmeta>` element is also available to specify metadata for the map, for individual topics, or for groups of topics. Many elements inside `<topicmeta>` are also available inside the topic prolog.

10.4.1.1 <map>

A DITA map is the mechanism for aggregating topic references and defining a context for those references. It contains references to topics, maps, and other resources; these references are organized into hierarchies, groups, and tables.

Usage information

A map describes the relationships among a set of DITA topics. The following are types of relationships that can be described in a map:

Hierarchical
: Nested topics create a hierarchical relationship. The topic that does the nesting is the parent, and the topics that are nested are the children.

Ordered
: Child topics can be labeled as having an ordered relationship, which means they are referenced in a definite sequence.

Family
: Child topics can be labeled as having a family relationship, which means they all refer to each other.

Rendering expectations

When rendering a map, processors might make use of the relationships defined in the map to create a Table of Contents (TOC), aggregate topics into a PDF document, or create links between topics in the output.

The `<title>` element can be used to provide a title for the map. In some scenarios the title is purely informational; it is present only as an aid to the author. In other scenarios, the title might be useful or even required. In a map referenced by another map, the title might be discarded as topics from the submap are aggregated into a larger publication.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (with a narrowed definition of `@id`, given below), Common map attributes (369), Architectural attributes (369), and the attributes defined below. This element also uses `@type`, `@scope`, and `@format` from Link-relationship attributes (369).

@anchorref
: Identifies a location within another map document where this map will be anchored. Resolution of the map is deferred until the final step in the delivery of any rendered content. For example, `anchorref="map1.ditamap#a1"` allows the map with `@anchorref` to be pulled into the location of the anchor point "a1" inside `map1.ditamap` when `map1.ditamap` is rendered for delivery.

@id

Allows an ID to be specified for the map. Note that maps do not require IDs (unlike topics), and the map ID is not included in references to elements within a map. This attribute is defined with the XML Data Type ID.

Example

The following code sample contains six `<topicref>` elements. The `<topicref>` elements are nested and have a hierarchical relationship. The file `bats.dita` is the parent topic and the other topics are its children. The hierarchy could be used to generate a PDF, a navigation pane in an information center, a summary of the topics, or related links between the parent topic and its children.

```
<map id="mybats">
  <title>Bats</title>
  <topicref href="bats.dita">
    <topicref href="batcaring.dita"/>
    <topicref href="batfeeding.dita"/>
    <topicref href="batsonar.dita"/>
    <topicref href="batguano.dita"/>
    <topicref href="bathistory.dita"/>
  </topicref>
</map>
```

10.4.1.2 <topicref>

A topic reference is the mechanism for referencing a topic (or another resource) from a DITA map. It can nest, which enables the expression of navigation and table-of-content hierarchies, as well as containment hierarchies and parent-child relationships.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with a narrowed definition of `@href`, given below), Common map attributes (369), Topicref-element attributes (369), `@keys` (389), and `@keyref` (388).

@href

Points to the resource that is represented by the `<topicref>`. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications. References to DITA content cannot be below the topic level: that is, you cannot reference individual elements inside a topic. References to content other than DITA topics should use the `@format` attribute to identify the kind of resource being referenced.

Example

Figure 122: Common map hierarchy using <topicref> elements

The following code sample shows a simple map that organizes several topics about the software product Example Tool Builder. The `<topicref>` that refers to `setup.dita` uses the `@collection-type` attribute to indicate that the order of three sub-topics in that section is important.

```
<map>
  <title>Example Tool Builder version 1.2.3</title>
  <topicref href="setup.dita" collection-type="sequence">
    <topicref href="prerequisites.dita"/>
    <topicref href="installing.dita"/>
    <topicref href="validating.dita"/>
  </topicref>
  <topicref href="everyday-use.dita">
    <!-- ... -->
  </topicref>
  <topicref href="troubleshooting.dita">
```

```
    <!-- ... -->
  </topicref>
</map>
```

10.4.1.3 <topicmeta>

Topic metadata is metadata that applies to a topic based on its context in a map.

Usage information

The metadata specified in a <topicmeta> element is specific to a given context within a map. If a reference to a single resource appears more than once in a map or set of maps, unique metadata can be specified in each instance. For example, when the parent <topicref> element results in a link, elements within the <topicmeta> element can be used to provide context-specific information about the link, such as link text, a short description, or a navigation title.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following example shows how the <topicmeta> element can contain a metadata element:

```
<map>
  <title>Indexing elements</title>
  <topicref href=indexing.dita">
    <topicmeta>
      <audience>Indexing specialists</audience>
    </topicmeta>
    <!-- Adding topic references -->
  </topicref>
</map>
```

The <audience> element indicates that the topic (and its child topics) are of interest to the indexing specialists within the company.

Related concepts

Cascading of metadata attributes in a DITA map (49)

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.

Related reference

Reconciling topic and map metadata elements (52)

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.

<anchor>

An anchor within a map is an integration point that another map can reference in order to insert its navigation into the referenced map's navigation tree.

Usage information

The <anchor> element is typically used to allow integration of run-time components. For build-time integration, you can use a <topicref> element to reference another map, or use the @conref or @conkeyref attribute on an element inside the map.

Processing expectations

The mechanism by which map processors discover maps to be anchored is processor specific.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (with a narrowed definition of @id, given below).

@id (REQUIRED)

Provides an integration point that another map can reference in order to insert its navigation into the current navigation tree. The @anchorref attribute on a map can be used to reference this attribute. See 6.1 ID attribute (73) for more details.

Example

The following code sample shows how a map creates an <anchor> element with an @id attribute set to "a1".

Figure 123: DITA map that contains an anchor

```
<map>
  <title>MyComponent tasks</title>
  <topicref href="start.dita" toc="yes">
    <navref mapref="othermap2.ditamap"/>
    <navref mapref="othermap3.ditamap"/>
    <anchor id="a1"/>
  </topicref>
</map>
```

The @id on an <anchor> element can be referenced by the @anchorref attribute on another map's <map> element. For example, the map to be integrated at that spot could be defined as follows.

Figure 124: DITA map that references an anchor

```
<map anchorref="map1.ditamap#a1">
  <title>This map is can be rendered at the "a1" anchor
         in the MyComponent task map</title>
  <!-- ... -->
</map>
```

10.4.1.5 <navref>

A navigation reference is a reference to another map which is preserved as a transcluding link in the result deliverable rather than resolved when the deliverable is produced. Output formats that support such linking can integrate the referenced resource when displaying the referencing map to an end user.

Usage information

The `<navref>` element is intended as a reference to a navigation resource that can be resolved at rendering time. It allows for maps to be published into a help system where the referenced navigation is published independently (or may not be avilable at all). If available, the referenced navigation can then be resolved at render time within a help system.

In order to include another map directly without depending on the output format or help system, use a `<topicref>` element with the `@format` attribute set to `ditamap`. The effect is similar to using a `@conref` attribute. For example, the following markup represents a literal inclusion of the map `other.ditamap`:

```
<topicref href="other.ditamap" format="ditamap"/>
```

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attribute defined below.

@mapref
: Specifies the URI of the map file or non-DITA resource to be referenced. It might reference a DITA map or a resource that is appropriate for a target help system. For example, it could reference an XML TOC file for use with Eclipse help.

Example

In the following code sample, the map titled "MyComponent tasks" references the maps `othermap2.ditamap` and `othermap3.ditamap`.

```
<map title="MyComponent tasks">
 <navref mapref="../com.example.plugin.xml.doc/othermap1.ditamap"/>
 <navref mapref="../com.example.plugin.xml.doc/othermap2.ditamap"/>
</map>
```

10.4.1.6 <reltable>

A relationship table is a mechanism that creates among topics, based on the familiar table model of rows, columns, and cells.

Usage information

Each column in a relationship table typically represents a specific role in a set of relationships. For example, a frequently-used type of relationship table uses the first column to contain references to tasks, while the second and third columns reference concept and reference topics. The relationship table rows define relationships between the resources referenced in different cells of the same row; in this example, each row establishes relationships between tasks and the concept and reference topics that support the tasks. When used in this manner, relationship tables make it easy to determine where related information is missing or undefined.

Relationship tables can be used in conjunction with hierarchies and groups to manage all the related links in an information set.

When a title is associated with a relationship table, the title typically is used as an authoring convenience and is not displayed in generated publications.

Processing expectations

By default, the contents of a `<reltable>` element are not output for navigation or TOC purposes; they are used only to define relationships that can be expressed as topic-to-topic links. The `<relcell>` elements can contain `<topicref>` elements, which are then related to other `<topicref>` elements in the same row (although not necessarily in the same cell).

Within a map tree, the effective relationship table is the union of all relationship tables in the map.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Common map attributes (369) (without `@keyscope` or `@collection-type`, and with a narrowed definition of `@toc`, given below), and the attributes defined below. This element also uses `@type`, `@scope`, and `@format` from Link-relationship attributes (369).

@toc
: Specifies whether the resource appears in the table of contents (TOC). If the value is not specified locally, but is specified on an ancestor, the value cascades from the closest containing element. By default, the value for `@toc` is set to "no". See Common map attributes (369) for a complete definition of `@toc`.

@title
: An identifying title for this element.

Example

In this example, a relationship table is defined with three columns; one for "concept", one for "task", and one for "reference". Three cells are defined within one row. The first cell contains one concept topic: `about-MyDevice.dita`. The second cell contains two task topics: `setting-up-MyDevice.dita` and `operating-MyDevice.dita`. The third cell contains two reference topics: `MyDevice-settings.dita` and `MyDevice-version-info.dita`.

```
<map>
  <reltable>
    <relheader>
      <relcolspec type="concept"/>
      <relcolspec type="task"/>
      <relcolspec type="reference"/>
    </relheader>
    <relrow>
      <relcell>
        <topicref href="about-MyDevice.dita"/>
      </relcell>
      <relcell>
        <topicref href="setting-up-MyDevice.dita"/>
        <topicref href="operating-MyDevice.dita"/>
      </relcell>
      <relcell>
        <topicref href="MyDevice-settings.dita"/>
        <topicref href="MyDevice-version-info.dita"/>
      </relcell>
    </relrow>
  </reltable>
</map>
```

A graphical version of the relationship table in an editor might look like this:

type="concept"	type="task"	type="reference"
about-MyDevice.dita	setting-up-MyDevice.dita operating-MyDevice.dita	MyDevice-settings.dita MyDevice-version-info.dita

On output, links should be added to topics that are in the same row, but not in the same cell. This allows simple maintenance of parallel relationships: for example, in this case, `setting-up-MyDevice.dita` and `operating-MyDevice.dita` are two tasks that require the same supporting information (concept and reference topics) but might otherwise be unrelated. When topics in the same cell are in fact related, the cell's `@collection-type` attribute can be set to family. If some cells or columns are intended solely as supporting information and should not link back to topics in other cells, you can set the `@linking` attribute on the `<relcell>` or `<relcolspec>` to "targetonly".

In this example, the related links would be as follows:

about-MyDevice.dita
`setting-up-MyDevice.dita`, `operating-MyDevice.dita`, `MyDevice-settings.dita`, `MyDevice-version-info.dita`

setting-up-MyDevice.dita
`about-MyDevice.dita`, `MyDevice-settings.dita`, `MyDevice-version-info.dita`

operating-MyDevice.dita
`about-MyDevice.dita`, `MyDevice-settings.dita`, `MyDevice-version-info.dita`

MyDevice-settings.dita
`about-MyDevice.dita`, `setting-up-MyDevice.dita`, `operating-MyDevice.dita`

MyDevice-version-info.dita
`about-MyDevice.dita`, `setting-up-MyDevice.dita`, `operating-MyDevice.dita`

Although such tables can initially take some time to learn and manipulate, they are inherently an efficient way to manage these links. In particular, they increase the prospect for reuse among topics, because those topics do not contain context-specific links. A relationship table also makes it easy to see and manage patterns; for example, the fact that `operating-MyDevice.dita` and `setting-up-MyDevice.dita` have the same relationships to supporting information is clear from the table, but would require some comparison and counting to determine from the list summary just before this paragraph.

10.4.1.7 <relrow>

A row in a relationship table creates a relationship between the cells in that row, which is expressed in output as links between the topics or resources referenced in those cells.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.4.1.6 reltable (272).

10.4.1.8 <relcell>

A relationship table cell is a group of one or more topic references that are related to topic references in other cells of the same row. A relationship table cell does not imply a relationship between topics or

resources that are referenced in the same cell, unless the `@collection-type` attribute cell indicates that they are related.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and Common map attributes (369) (without `@keyscope`). This element also uses `@type`, `@scope`, and `@format` from Link-relationship attributes (369).

Example

See 10.4.1.6 reltable (272).

10.4.1.9 <relheader>

A relationship table header row is a group of column definitions for a relationship table.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.4.1.6 reltable (272).

10.4.1.10 <relcolspec>

A relationship table column specification provides default attribute values for references in a single column of a relationship table.

Usage information

You can use the `<relcolspec>` element to set default values for the attributes of the topics that are referenced in the column. For example, when you set the `@type` attribute to "concept," all `<topicref>` elements in the column that do not have a `@type` attribute specified are treated as concepts.

Adding a `<topicref>` element to the `<relcolspec>` element defines a relationship between the topic (or topics) contained within the `<relcolspec>` element and the topics that are referenced in the column of the relationship table. Note that this does not define a relationship between two cells in the same column; the only new relationship is between `<topicref>` targets in a `<relcell>` and `<topicref>` targets in that column's `<relcolspec>`.

Rendering expectations

When a `<title>` element exists inside of the `<relcolspec>` element, the content of the `<title>` element is intended to be used as the label for the related links that are defined and generated by the column. If the `<title>` element is not present, the labels for the related links are generated in the following ways:

- If the `<relcolspec>` element contains a `<topicref>` element that specifies a navigation title, that navigation title is used for the label.
- If the `<relcolspec>` element contains a `<topicref>` element that does not specify a navigation title but does reference a DITA topic, the label is derived from the navigation title of the referenced topic or, lacking that, the title of the topic.

- If no title is specified and no <topicref> is present in the <relcolspec>, a rendering tool might choose to generate a title for the links generated from that column.

Processing expectations

When values are specified for attributes of <relcell> or <relrow> elements, those values are inherited before those defined for <relcolspec> attributes. Values specified for attributes of <relcolspec> elements are inherited before those defined for the <reltable> element.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and Common map attributes (369) (without @keyscope or @collection-type). This element also uses @type, @scope, and @format from Link-relationship attributes (369).

Example

The following section shows how a simple relationship table uses column specifications to validate topic types.

Figure 125: Enforcing concept, type, and reference types with <relcolspec>

The following code sample shows how a relationship table groups relationships between concepts, tasks, and references. Three cells are defined within one row. The first cell contains one concept topic: puffins.dita. The second cell contains two task topics: puffinFeeding.dita and puffinCleaning.dita. The third cell contains a reference topic: puffinHistory.dita. Setting the @type on each column allows (but does not require) processors to validate that the topics in each column are of the expected type.

```
<map>
 <reltable>
  <relheader>
   <relcolspec type="concept"/>
   <relcolspec type="task"/>
   <relcolspec type="reference"/>
  </relheader>
  <relrow>
   <relcell><topicref href="puffins.dita"/></relcell>
   <relcell>
     <topicref href="puffinFeeding.dita"/>
     <topicref href="puffinCleaning.dita"/>
   </relcell>
   <relcell>
     <topicref href="puffinHistory.dita"/>
   </relcell>
  </relrow>
 </reltable>
</map>
```

The following section shows how relationship table columns can reference topics and specify titles for use when generating groups of links.

Figure 126: Relationship table column headers with topics and titles

The following code sample shows how topics and titles can be specified in a relationship table column header.

```
<reltable>
  <relheader>
    <relcolspec type="task">
      <topicref href="tbs.dita">
        <topicmeta><navtitle>Troubleshooting</navtitle></topicmeta>
```

```
      </topicref>
    </relcolspec>
    <relcolspec type="reference">
      <topicref href="msg.dita">
        <topicmeta><navtitle>Messages</navtitle></topicmeta>
      </topicref>
    </relcolspec>
  </relheader>
  <relrow>
    <relcell>
      <topicref href="debug_login.dita"/>
        <topicmeta><linktitle>Debugging login errors</linktitle></topicmeta>
      </topicref>
    </relcell>
    <relcell>
      <topicref href="login_error_1.dita">
        <topicmeta><linktitle>Login not found</linktitle></topicmeta>
      </topicref>
    </relcell>
  </relrow>
  <relrow>
    <relcell>
      <topicref href="checking_access.dita">
        <topicmeta><linktitle>Checking access controls</linktitle></topicmeta>
      </topicref>
    </relcell>
    <relcell>
      <topicref href="login_error_2.dita">
        <topicmeta><linktitle>Login not allowed</linktitle></topicmeta>
      </topicref>
    </relcell>
  </relrow>
</reltable>
```

In addition to the relationships defined by the rows in the relationship table, the following relationships are now defined by the columns in the relationship table:

- `tbs.dita` **<–>** `debug_login.dita`
- `tbs.dita` **<–>** `checking_access.dita`
- `msg.dita` **<–>** `login_error_1.dita`
- `msg.dita` **<–>** `login_error_2.dita`

Ignoring the headers for a moment, the `<reltable>` here would ordinarily define a two-way relationship between `debug_login.dita` and `login_error1.dita`. This will typically be expressed as a link from each to the other. An application might render the link with a language-appropriate heading such as "Related reference", indicating that the target of the link is a reference topic.

The headers change this by specifying a new title. In the second column, the `<topicref>` specifies a title of "Messages", which should now be used together with the link to anything in that column. So, a generated link from `debug_login.dita` to `login_error1.dita` should be rendered together with the title of "Messages". How this is rendered together with the link is up to the application.

<anchor>

An anchor within a map is an integration point that another map can reference in order to insert its navigation into the referenced map's navigation tree.

Usage information

The `<anchor>` element is typically used to allow integration of run-time components. For build-time integration, you can use a `<topicref>` element to reference another map, or use the `@conref` or `@conkeyref` attribute on an element inside the map.

Processing expectations

The mechanism by which map processors discover maps to be anchored is processor specific.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (with a narrowed definition of `@id`, given below).

@id (REQUIRED)
: Provides an integration point that another map can reference in order to insert its navigation into the current navigation tree. The `@anchorref` attribute on a map can be used to reference this attribute. See 6.1 ID attribute (73) for more details.

Example

The following code sample shows how a map creates an `<anchor>` element with an `@id` attribute set to "a1".

Figure 127: DITA map that contains an anchor

```
<map>
  <title>MyComponent tasks</title>
  <topicref href="start.dita" toc="yes">
    <navref mapref="othermap2.ditamap"/>
    <navref mapref="othermap3.ditamap"/>
    <anchor id="a1"/>
  </topicref>
</map>
```

The `@id` on an `<anchor>` element can be referenced by the `@anchorref` attribute on another map's `<map>` element. For example, the map to be integrated at that spot could be defined as follows.

Figure 128: DITA map that references an anchor

```
<map anchorref="map1.ditamap#a1">
  <title>This map is can be rendered at the "a1" anchor
         in the MyComponent task map</title>
  <!-- ... -->
</map>
```

10.4.1.12 <keytext>

A `<keytext>` element specifies variable or link text; it also specifies alternate text for images that are referenced by keys.

Processing expectations

See 6.4.9 Processing key references to generate text or link text (85).

Comment by Kristen J Eberlein on 31 October 2020

Should this be a related link instead of a cross reference?

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Examples

The section contains examples of how the <keytext> element can be used.

Figure 129: Simple example

The following code sample shows how variable text can be defined using the <keytext> element:

```
<keydef keys="company-name">
  <topicmeta>
    <keytext translate="no">Acme Widget Company</keytext>
  </topicmeta>
</keydef>
```

Figure 130: More complex example

The following code sample shows a variable-text definition that includes highlighting elements:

```
<keydef keys="company-name">
  <topicmeta>
    <keytext translate="no">
      <i>Super</i> Widget Squared<sup>2</sup>
    </keytext>
  </topicmeta>
</keydef>
```

Figure 131: Alternate text for an image

DITA implementations often reference images using keys. In such cases, the <keytext> element provides the alternate text for the image. The following code sample shows the markup for the <keytext> element

```
<keydef keys="company-logo" href="images/logo.jpg" format="jpg">
  <topicmeta>
    <keytext>Blue Acorn logo</keytext>
  </topicmeta>
</keydef>
```

The image can be referenced by <image keyref="company-logo"/>. When rendered to mediums that support alternate text, the effective alternative text for the image is "Blue Acorn," as though a literal <alt>element had been a child of the <image>

Figure 132: Variable text that is conditionally processed

DITA implementations often need to conditionally process product names. The following code sample shows a <keytext> element that contains <ph> elements that are conditionally processed:

```
<keydef keys="company-name">
  <topicmeta>
    <keytext translate="no">
      <ph product="cat">Acme Widgets for Cats</ph
      <ph product="dog">Acme Widgets for Dogs</ph>
      <ph product="pig">Acme Widgets for Pigs</ph>
    </keytext>
  </topicmeta>
</keydef>
```

Figure 133: Processing logic

The following sample shows a key definition that includes several elements within the <topicmeta> element:

```
<keydef href="http://www.example.com" keys="company-name">
  <topicmeta>
```

```
    <keytext>Acme Tools</keytext>
    <navtitle>Acme Tools web site</navtitle>
    <linktext>Acme Tools Web Portal</linktext>
  </topicmeta>
</keydef>
```

Once processed, the effective text content of both `<ph keyref="company-name"/>` and `<xref keyref="company-name"/>` is "Acme Tools".

To set distinct text values for both the company name and the link text that is associated with the company Web site, best practices call for using two different key definitions.

10.4.2 Subject scheme elements

Subject scheme maps are used to define sets of controlled values and taxonomic subjects, bind controlled values to attributes as enumerations, and specify relationships among taxonomic subjects.

10.4.2.1 <attributedef>

The `<attributedef>` element specifies an attribute that is bound to a set of controlled values. This binding restricts the permissible values for the attribute to the set of controlled values.

Specialization hierarchy

The `<attributedef>` element is specialized from `<data>`. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: ID attributes (366), `@status` and `@base` from Metadata attributes (366), outputclass (368), class (not for use by authors) (367), and the attributes defined below.

@name (REQUIRED)
Specifies the attribute that is bound to a set of enumerated values

@translate
Specifies whether the content of the element is translatable. The default value is "no". Setting `@translate` to "yes" overrides the default value. The DITA specification contains a list of each OASIS DITA element and its common processing default for the translate value; because this element uses an actual default, it is always treated as `translate="no"` unless overridden. Available values are:

no
The content of this element is not translatable.

yes
The content of this element is translatable.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

Example

In the following code sample, the enumeration definition for the `@otherprops` attribute limits the permissible values to the subject "values-otherprops". This means that the only valid value for the `@otherprops` attribute is "examples".

```
<subjectScheme>

    <!-- DEFINE SETS OF CONTROLLED VALUES -->
```

```
    <!-- Values for @otherprops -->
    <subjectdef keys="values-otherprops">
        <subjectdef keys="examples"/>
    </subjectdef>

    <!-- BINDS SETS OF CONTROLLED VALUES -->

    <!-- Binding for @otherprops -->
    <enumerationdef>
        <attributedef name="otherprops"/>
        <subjectdef keyref="values-otherprops"/>
    </enumerationdef>

</subjectScheme>
```

10.4.2.2 <defaultSubject>

The `<defaultSubject>` element specifies the default value for the attribute in cases where no value is specified. The default value must be one of the controlled values that are bound to the attribute.

Specialization hierarchy

The `<defaultSubject>` element is specialized from `<topicref>`. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with a narrowed definition of `@href`, given below), Topicref-element attributes (369), `@keys` (389), and `@keyref` (388). This element also uses `@processing-role` and `@toc` from Common map attributes (369).

@href
: Specifies the referenced resource. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications. References to non-DITA content need to use the `@format` attribute to identify the kind of resource.

Example

The following code sample limits the values for `@platform` to the "os" subject. The result is that the following values are permitted for `@platform`:

- `linux`
- `mswin`
- `zos`
- `macos`

If no value is specified for the `@platform` attribute in the DITA source, the default is "linux".

```
<subjectScheme>
  <subjectdef keys="os">
    <subjectdef keys="linux"/>
    <subjectdef keys="mswin"/>
    <subjectdef keys="zos"/>
    <subjectdef keys="macos"/>
  </subjectdef>
  <enumerationdef>
    <attributedef name="platform"/>
    <defaultSubject keyref="linux"/>
    <subjectdef keyref="os"/>
```

```
  </enumerationdef>
</subjectScheme>
```

10.4.2.3 <elementdef>

The <elementdef> element specifies an element to which an attribute and set of controlled values are bound.

Specialization hierarchy

The <elementdef> element is specialized from <data>. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: ID attributes (366), @status and @base from Metadata attributes (366), outputclass (368), class (not for use by authors) (367), and the attributes defined below.

@name (REQUIRED)
Specifies the element to which the set of controlled values are bound

@translate
Specifies whether the content of the element is translatable. The default value is "no". Setting @translate to "yes" overrides the default value. The DITA specification contains a list of each OASIS DITA element and its common processing default for the translate value; because this element uses an actual default, it is always treated as translate="no" unless overridden. Available values are:

no
The content of this element is not translatable.

yes
The content of this element is translatable.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

Example

In this example, the @type attribute for the <note> element is bound to the specified set of values. Processors should limit the values for the @type attribute on the <note> element to the following set of values: attention, caution, and danger. Other elements that have a @type attribute are not affected.

```
<subjectScheme>
  <subjectdef keys="note-values">
    <subjectdef keys="attention"/>
    <subjectdef keys="caution"/>
    <subjectdef keys="danger"/>
  </subjectdef>
  <!-- ... -->
  <enumerationdef>
    <elementdef name="note"/>
    <attributedef name="type"/>
    <subjectdef keyref="note-values"/>
  </enumerationdef>
</subjectScheme>
```

10.4.2.4 <enumerationdef>

The <enumerationdef> contains an enumeration definition. An enumeration definition specifies an attribute, a set of controlled values (optional), and the element to which the attribute and controlled values pair are bound (optional).

Usage information

An enumeration definition can accomplish the following goals:

Bind a set of controlled values to an attribute

When the <enumerationdef> element contains an <attributedef> and a <subjectdef> element, the set of controlled values bound to the attribute apply to all elements.

For example, when <enumerationdef> contains only <attributedef name="value"/>, the @value attribute is limited to the specified enumeration for all elements.

Limit a set of controlled values to a specific element and attribute pair

When the <enumerationdef> element contains an <attributedef>, a <subjectdef>, **and** an <elementdef> element, the enumeration applies to the specified attribute **only** on the specified element. The enumeration does not apply to the attribute on other elements.

For example, when the <enumerationdef> element contains both <attributedef name="type"/> and <elementdef name="note"/>, only the @type attribute on the <note> element is limited to the specified enumeration. The possible values for the @type attribute on other elements are not affected.

Specify that an attribute is not valid.

When the <enumerationdef> element is empty, no value is valid for the attribute.

Whether an attribute takes a single value or multiple values from the enumeration is part of the structural definition of the element. An attribute that is defined as CDATA can take multiple values, while an attribute defined as NMTOKEN can take only one.

Specialization hierarchy

The <enumerationdef> element is specialized from <topicref>. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: ID attributes (366), @status and @base from Metadata attributes (366), outputclass (368), and class (not for use by authors) (367).

Example

The following subject scheme map contains three enumeration definitions:

1. The permissible values for the @audience attribute on the <draft-comment> element are restricted to the subject "values-audience-draft-comment". This means that the only allowed values are "spec-editors" and "tc-reviewers".
2. The permissible values for @otherprops are restricted to the subject "values-otherprops". This means that the only valid value for @otherprops is "examples".

3. The enumeration for the `@props` element is empty. No values are valid.

```
<subjectScheme>

    <!-- DEFINE SETS OF CONTROLLED VALUES -->

    <!-- 1. Values for @audience on <draft-comment> -->
    <subjectdef keys="values-audience-draft-comment">
        <subjectdef keys="spec-editors"/>
        <subjectdef keys="tc-reviewers"/>
    </subjectdef>

    <!-- 2. Values for @otherprops -->
    <subjectdef keys="values-otherprops">
        <subjectdef keys="examples"/>
    </subjectdef>

    <!-- BINDS SETS OF CONTROLLED VALUES -->

    <!-- 1. Binding for @audience on <draft-comment> -->
    <enumerationdef>
        <elementdef name="draft-comment"/>
        <attributedef name="audience"/>
        <subjectdef keyref="values-audience-draft-comment"/>
    </enumerationdef>

    <!-- 2. Binding for @otherprops -->
    <enumerationdef>
        <attributedef name="otherprops"/>
        <subjectdef keyref="values-otherprops"/>
    </enumerationdef>

    <!-- 3. Binding for @props -->
    <enumerationdef>
        <attributedef name="props"/>
    </enumerationdef>

</subjectScheme>
```

10.4.2.5 <hasInstance>

The `<hasInstance>` element specifies that the contained subjects have an INSTANCE-OF relationship with the container subject. In an INSTANCE-OF hierarchy, the child subject is a specific entity or object and the parent subject is a type, kind, or class of entity or object.

Specialization hierarchy

The `<hasInstance>` element is specialized from `<topicref>`. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with a narrowed definition of `@href`, given below), `@processing-role` from Common map attributes (369), `@keys` (389), and `@keyref` (388).

@href

Specifies the referenced resource. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications. References to non-DITA content need to use the `@format` attribute to identify the kind of resource.

Example

The following code sample specifies that New York City, Reykjavik, and Moscow are each specific instances of a city.

```
<subjectScheme>
  <hasInstance>
    <subjectdef keys="city">
      <subjectdef keys="nyc"/>
      <subjectdef keys="reykjavik"/>
      <subjectdef keys="moscow"/>
    </subjectdef>
  </hasInstance>
</subjectScheme>
```

10.4.2.6 <hasKind>

The `<hasKind>` element specifies that the contained hierarchy expresses KIND-OF relationships between subjects. In a KIND-OF hierarchy, the child subject is a particular variety of the parent subject. A KIND-OF hierarchy is sometimes known as an IS-A, generic, or subsumption hierarchy.

Specialization hierarchy

The `<hasKind>` element is specialized from `<topicref>`. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with a narrowed definition of `@href`, given below), `@processing-role` from Common map attributes (369), `@keys` (389), and `@keyref` (388).

@href
: Specifies the referenced resource. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications. References to non-DITA content need to use the `@format` attribute to identify the kind of resource.

Example

The following code sample specifies that cities, towns, and villages are each a kind of settlement. Additionally, big-city, medium-city, and small-city are each a kind of city.

```
<subjectScheme>
  <hasKind>
    <subjectdef keys="settlement">
      <subjectdef keys="city">
        <subjectdef keys="big-city"/>
        <subjectdef keys="medium-city"/>
        <subjectdef keys="small-city"/>
      </subjectdef>
      <subjectdef keys="town"/>
      <subjectdef keys="village"/>
    </subjectdef>
  </hasKind>
</subjectScheme>
```

10.4.2.7 <hasNarrower>

The <hasNarrower> element specifies that the contained subjects have a narrower focus than the container subject. The way in which a relationship is narrower is not specified.

Specialization hierarchy

The <hasNarrower> element is specialized from <topicref>. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with a narrowed definition of @href, given below), @processing-role from Common map attributes (369), @keys (389), and @keyref (388).

@href

Specifies the referenced resource. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications. References to non-DITA content need to use the @format attribute to identify the kind of resource.

Example

The following code sample specifies that "Planting roses" is a narrower subject category than "Horticulture", although it is part of the "Horticulture" subject.

```
<subjectScheme>
  <hasNarrower>
    <subjectdef keys="horticulture">
      <subjectdef keys="planting-roses"/>
    </subjectdef>
  </hasNarrower>
</subjectScheme>
```

10.4.2.8 <hasPart>

The <hasPart> element specifies that the contained hierarchy expresses PART-OF relationships between subjects.

Specialization hierarchy

The <hasPart> element is specialized from <topicref>. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with a narrowed definition of @href, given below), @processing-role from Common map attributes (369), @keys (389), and @keyref (388).

@href

Specifies the referenced resource. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications. References to non-DITA content need to use the @format attribute to identify the kind of resource.

Example

The following code sample specifies that a tire and a horn are each a part of a car.

```
<subjectScheme>
  <hasPart>
    <subjectdef keys="car">
      <subjectdef keys="tire"/>
      <subjectdef keys="horn"/>
    </subjectdef>
  </hasPart>
</subjectScheme>
```

10.4.2.9 <hasRelated>

The `<hasRelated>` element specifies that the contain subjects have an associative relationship with the container subject.

Specialization hierarchy

The `<hasRelated>` element is specialized from `<topicref>`. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with a narrowed definition of `@href`, given below), processing-role from Common map attributes (369), Topicref-element attributes (369), `@keys` (389), and `@keyref` (388).

@href
: Specifies the referenced resource. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications. References to non-DITA content need to use the `@format` attribute to identify the kind of resource.

Example

This example specifies that myProgram runs on Linux and Windows.

```
<subjectScheme>
  <subjectdef keys="myProgram">
    <hasRelated keys="platforms">
      <subjectdef keys="linux">
      <subjectdef keys="windows"/>
    </hasRelated>
  </subjectdef>
</subjectScheme>
```

10.4.2.10 <relatedSubjects>

The `<relatedSubjects>` element specifies that associative relationships exist between all contained subjects. The utility of this relationship is primarily for content delivery platforms with advanced capabilities.

Usage information

The information architect can identify the relationship by specifying a `@keys` attribute, label the relationship by specifying a `<titlealt>` with a `@title-role` of `navigation` or `linking` such as `<navtitle>` or `<linktitle>`, and provide a consensus definition of the associative relationship by

referencing a topic. If the relationship has an identifying key, the content provider can use the `@keyref` attribute to specify the same relationship for different subjects.

Processing expectations

Comment by Robert
Wonder if it should instead just be removed? As someone who works on a processor, this is not really telling me anything that I can use. Same for the second sentence of the short description above, which mirrors this: "The utility of this relationship is primarily for content delivery platforms with advanced capabilities."

For filtering and flagging, processors need only inspect the subordinate hierarchies under category subjects that are bound to attributes. Filtering and flagging processors do not have to understand specific types of relationships. Explicit relationships are useful primarily for information viewers with advanced capabilities.

Specialization hierarchy

The `<relatedSubjects>` element is specialized from `<>`. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with a narrowed definition of `@href`, given below), `@keys` (389), and `@keyref` (388). This element also uses `@processing-role`, `@collection-type`, and a narrowed definition of `@linking` (given below) from Common map attributes (369).

@href
Specifies the referenced resource. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications. References to non-DITA content need to use the `@format` attribute to identify the kind of resource.

@linking
On this element, the `@linking` attribute has a default value of "normal". Otherwise, the attribute is the same as defined in Common map attributes (369).

Example

The following code sample specifies that the Linux, the Apache Web Server, and the MySQL Database subjects are related.

```
<subjectScheme>
  <!-- ... -->
  <relatedSubjects>
    <subjectdef keys="linux">
    <subjectdef keys="apache-web-server"/>
    <subjectdef keys="my-sql"/>
  </relatedSubjects>
  <!-- ... -->
</subjectScheme>
```

10.4.2.11 <schemeref>

A `<schemeref>` element references another subject scheme map. Typically, the referenced subject scheme defines a base set of controlled values; the referencing map then extends the base set of

controlled values. The values specified in the subject scheme maps are merged; the result is equivalent to specifying all of the values in a single map.

Specialization hierarchy

The <schemeref> element is specialized from <topicref>. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with narrowed definitions of @href, @format, and @type, given below), @keys (389), and @keyref (388).

@href
Specifies the referenced resource. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications. References to non-DITA content need to use the @format attribute to identify the kind of resource.

@format
Specifies the format of the resource. By default, it is set to "ditamap". Otherwise, the attribute is the same as described in Link-relationship attributes (369).

@type
Describes the target of a reference. For the <schemeref> element, this value defaults to "scheme", because the element is expected to point to another subject scheme.

See 10.4.2.19 subjectScheme (295).

10.4.2.12 <subjectdef>

The <subjectdef> element defines a subject. A subject can be used for either taxonomic classification or as a controlled value.

Usage information

The <subjectdef> can use a <titlealt> element with a @title-role of navigation, such as <navtitle>, to supply a label for the subject; the @href attribute can be used to reference a topic that captures the consensus definition for the subject.

Specialization hierarchy

The <subjectdef> element is specialized from <topicref>. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Topicref-element attributes (369), Link-relationship attributes (369) (with a narrowed definition of @href, given below), @keys (389), and @keyref (388). This element also uses @processing-role, @toc, @collection-type, and @linking from Common map attributes (369).

@href
Specifies the referenced resource. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications. References to non-DITA content need to use the @format attribute to identify the kind of resource.

Example

```
<subjectScheme>
  <!-- Pull in a scheme that defines unix OS values -->
  <schemeref href="unixOS.ditamap"/>
  <!-- Define new OS values that are merged with those in the unixOS scheme -->
  <subjectdef keys="operating-systems">
    <subjectdef keys="linux"/>
    <subjectdef keys="windows"/>
    <subjectdef keys="zOS"/>
  </subjectdef>
  <!-- Define application values -->
  <subjectdef keys="applications">
    <subjectdef keys="apache-server" href="subject/apache.dita"/>
    <subjectdef keys="my-sql"      href="subject/sql.dita"/>
  </subjectdef>

  <!-- Define an enumeration of the platform attribute, equal to
       each value in the OS subject. This makes the following values
       valid for the platform attribute: linux, windows, zOS -->
  <enumerationdef>
    <attributedef name="platform"/>
    <subjectdef keyref="os"/>
  </enumerationdef>
  <!-- Define an enumeration of the otherprops attribute, equal to
       each value in the application subjects.
       This makes the following values valid for the otherprops attribute:
       apache-server, my-sql -->
  <enumerationdef>
    <attributedef name="otherprops"/>
    <subjectdef keyref="applications"/>
  </enumerationdef>
</subjectScheme>
```

10.4.2.13 <subjectHead>

The <subjectHead> element provides a heading for a group of subjects, for use if the scheme is displayed. \

Usage information

For example, the <subjectHead> could be displayed as part of a scheme that is rendered to let a user select subjects as part of faceted browsing.

The <subjectHead> element itself does not reference a file and cannot be referenced as a key, so it does not define any controlled values.

Specialization hierarchy

The <subjectHead> element is specialized from <topicref>. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366). This element also uses @processing-role, @toc, and narrowed definitions of @collection-type and @linking from Common map attributes (369).

@collection-type

Collection types describe how links relate to each other. The processing default is "unordered", although no default is specified in the DTD or Schema. Allowable values for @collection-type on this element are:

unordered
Indicates that the order of the child topics is not significant.

sequence
Indicates that the order of the child topics is significant; output processors will typically link between them in order.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

@linking
Defines some specific linking characteristics of subject topics. "normal" is the only valid value, and is specified as the default in the DTD, XSD Schema, and RELAX NG implementations. When attribute values cascade, this causes a linking value of "normal" to cascade to the subjects.

Example

In this example the “Server setup” label doesn't classify content but, when selected, is equivalent to the union of its child subjects. That is, the heading covers content about planning for any application, installing for any application, any task for web servers, or any task for database servers.

```
<subjectScheme toc="yes" search="no">
  <!-- ... -->
  <subjectHead>
    <subjectHeadMeta>
      <navtitle>Server setup</navtitle>
    </subjectHeadMeta>
    <subjectdef href="planningTaskType.dita"/>
    <subjectdef href="installingTaskType.dita"/>
    <subjectdef href="webServerApp.dita"/>
    <subjectdef href="databaseApp.dita"/>
  </subjectHead>
  <!-- ... -->
</subjectScheme>
```

10.4.2.14 <subjectHeadMeta>

The <subjectHeadMeta> element allows a navigation title and short description to be associated with a subject heading.

Specialization hierarchy

The <subjectHeadMeta> element is specialized from <topicmeta>. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

In this example the “Server setup” label doesn't classify content but, when selected, is equivalent to the union of its child subjects. That is, the heading covers content about planning for any application, installing for any application, any task for web servers, or any task for database servers.

```
<subjectScheme toc="yes" search="no">
  <!-- ... -->
  <subjectHead>
    <subjectHeadMeta>
      <navtitle>Server setup</navtitle>
    </subjectHeadMeta>
    <subjectdef href="planningTaskType.dita"/>
```

```
    <subjectdef href="installingTaskType.dita"/>
    <subjectdef href="webServerApp.dita"/>
    <subjectdef href="databaseApp.dita"/>
  </subjectHead>
  <!-- ... -->
</subjectScheme>
```

10.4.2.15 <subjectRel>

The <subjectRel> element contains a set of subjects that are related in some manner. Each group of subjects is contained in a <subjectRole> element; the associations between different columns in the same row are evaluated in the same way as those in a <relrow> (from which <subjectRel> is specialized) but define relationships between the subjects instead of links between topic documents.

Specialization hierarchy

The <subjectRel> element is specialized from <relrow>. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

For a complete example in the context of the <subjectRelTable>, see Example (293).

10.4.2.16 <subjectRelTable>

The <subjectRelTable> element is a specialized relationship table which establishes relationships between the subjects in different columns of the same row.

Usage information

This element provides an efficient way to author non-hierarchical relationships between subjects. Tools (such as search tools) that use subject relationships to find related content might use these associative relationships in a similar way to the hierarchical relationships.

Specialization hierarchy

The <subjectRelTable> element is specialized from <reltable>. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and Common map attributes (369) (with a narrowed definition of @toc, given below). This element also uses @type, @scope, and @format from Link-relationship attributes (369).

@toc

Specifies whether the resource appears in the table of contents (TOC). If the value is not specified locally, but is specified on an ancestor, the value cascades from the closest containing element. By default, the value for @toc is set to "no". See Common map attributes (369) for a complete definition of @toc.

Example

The subject relationship table in this example establishes relationships between operating systems and applications. Based on the `<subjectRole>` element, subjects in the first column are operating systems which are the environment for an application, while subjects in the second column are applications that run in that environment. For a user interested in content about the operating system, content about the applications might also be relevant.

```
<subjectScheme>
  <hasKind>
    <subjectdef keys="operatingSystem">
        <subjectdef keys="linuxOS"/>
        <subjectdef keys="windowsOS"/>
    </subjectdef>
    <subjectdef keys="application">
        <subjectdef keys="IDE">
            <subjectdef keys="eclipseIDE"/>
            <subjectdef keys="visualStudioIDE"/>
        </subjectdef>
        <subjectdef keys="webBrowser">
            <subjectdef keys="firefoxBrowser"/>
            <subjectdef keys="ieBrowser"/>
        </subjectdef>
    </subjectdef>
  </hasKind>
  <!-- ... -->
  <subjectRelTable>
    <subjectRelHeader>
      <subjectRole>
        <subjectdef keyref="operatingSystem">
          <hasRelated keyref="environmentFor">
            <subjectdef keyref="application"/>
          </hasRelated>
        </subjectdef>
      </subjectRole>
      <subjectRole>
        <subjectdef keyref="application"/>
      </subjectRole>
    </subjectRelHeader>
    <subjectRel>
      <subjectRole>
        <subjectdef keyref="linuxOS"/>
        <subjectdef keyref="windowsOS"/>
      </subjectRole>
      <subjectRole>
        <subjectdef keyref="eclipseIDE"/>
        <subjectdef keyref="firefoxBrowser"/>
      </subjectRole>
    </subjectRel>
    <subjectRel>
      <subjectRole>
        <subjectdef keyref="windowsOS"/>
      </subjectRole>
      <subjectRole>
        <subjectdef keyref="ieBrowser"/>
        <subjectdef keyref="visualStudioIDE"/>
      </subjectRole>
    </subjectRel>
  </subjectRelTable>
</subjectScheme>
```

A table view of the `<subjectRelTable>` might look like this; each `<subjectRel>` represents a single row, and each `<subjectRole>` represents a cell.

Table 7: <subjectRelTable> as a table

<subjectdef keyref="operatingSystem"> <hasRelated keyref="environmentFor"> <subjectdef keyref="application"/> </hasRelated> </subjectdef>	<subjectdef keyref="application"/>
<subjectdef keyref="linuxOS"/> <subjectdef keyref="windowsOS"/>	<subjectdef keyref="eclipseIDE"/> <subjectdef keyref="firefoxBrowser"/>
<subjectdef keyref="windowsOS"/>	<subjectdef keyref="ieBrowser"/> <subjectdef keyref="visualStudioIDE"/>

10.4.2.17 <subjectRelHeader>

The <subjectRelHeader> element specifies the roles played by subjects in associations established by a <subjectRelTable>.

Usage information

You use the <subjectRelHeader> element to supply a header row for a subject relationship table when you want to identify the roles played by the subjects in each column. Each cell in the header row identifies a subject topic that defines a role. When specializing the <subjectRelTable> element, you can accomplish the same purpose by specializing the cells within the rows to enforce the roles.

Specialization hierarchy

The <subjectRelHeader> element is specialized from <relrow>. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

For a complete example in the context of the <subjectRelTable>, see Example (293).

10.4.2.18 <subjectRole>

The <subjectRole> element, when used within a <subjectRel> element, contains a set of subjects that are related to other subjects in the same row of the current <subjectRelTable>.

Usage information

By default, no relationship is defined between multiple subjects in the same <subjectRole> element. When used within the <subjectRelHeader>, the <subjectRole> element defines the category of subject or relationship provided by that column.

Specialization hierarchy

The <subjectRole> element is specialized from <relcell>. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and Topicref-element attributes (369). This element also uses @scope, @format, and @type from Link-relationship attributes (369).

Example

For a complete example in the context of the <subjectRelTable>, see Example (293).

10.4.2.19 <subjectScheme>

The <subjectScheme> element defines taxonomic subjects and controlled values.

Specialization hierarchy

The <subjectScheme> element is specialized from <map>. It is defined in the subject scheme module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (with a narrowed definition of @id, given below), Common map attributes (369) (with narrowed definitions of @processing-role and @toc, given below), Architectural attributes (369), and the attributes defined below. This element also uses @type, @scope, and @format from Link-relationship attributes (369).

@id
: Specifies an ID for the map. Note that maps do not require IDs (unlike topics), and the map ID is not included in references to elements within a map. This attribute is defined with the XML Data Type ID.

@anchorref
: Specifies a location within another map document where this map will be anchored. Resolution of the map is deferred until the final step in the delivery of any rendered content. For example, anchorref="map1.ditamap#a1" allows the map with @anchorref to be pulled into the location of the anchor point "a1" inside map1.ditamap when map1.ditamap is rendered for delivery.

@processing-role
: Specifies the role that the resource plays in processing. By default, this is set to "resource-only". Otherwise, the definition matches the one in Common map attributes (369).

@toc
: For this element, the default value for @toc is "no". Otherwise, the definition matches the one found in Common map attributes (369).

Example

```
<subjectScheme>
  <!-- Pull in a scheme that defines unix OS values -->
  <schemeref href="unixOS.ditamap"/>
  <!-- Define new OS values that are merged with those in the unixOS scheme -->
  <subjectdef keys="operating-systems">
    <subjectdef keys="linux"/>
    <subjectdef keys="windows"/>
    <subjectdef keys="zOS"/>
  </subjectdef>
```

```
  <!-- Define application values -->
  <subjectdef keys="applications">
    <subjectdef keys="apache-server" href="subject/apache.dita"/>
    <subjectdef keys="my-sql"      href="subject/sql.dita"/>
  </subjectdef>

  <!-- Define an enumeration of the platform attribute, equal to
       each value in the OS subject. This makes the following values
       valid for the platform attribute: linux, windows, zOS -->
  <enumerationdef>
    <attributedef name="platform"/>
    <subjectdef keyref="os"/>
  </enumerationdef>
  <!-- Define an enumeration of the otherprops attribute, equal to
       each value in the application subjects.
       This makes the following values valid for the otherprops attribute:
       apache-server, my-sql -->
  <enumerationdef>
    <attributedef name="otherprops"/>
    <subjectdef keyref="applications"/>
  </enumerationdef>
</subjectScheme>
```

10.5 Metadata elements

Metadata elements include information that is located within the `<topicmeta>` element (in maps) or `<prolog>` element (in topics), as well as indexing elements that can be placed in additional locations within topic content.

10.5.1 Prolog (metadata) elements

The prolog elements represent the metadata associated with a document. Most of the metadata in a topic prolog can also be authored in a DITA map, in the map's `<topicmeta>` element.

10.5.1.1 <audience>

An `<audience>` element specifies an audience for a topic.

Usage information

A topic can indicate multiple audiences by including multiple `<audience>` elements. For each audience specified, the high-level task they are trying to accomplish can be identified with the `@job` attribute, and the level of experience expected can be specified with the `@experiencelevel` attribute. The `<audience>` element can be used to provide a more detailed definition of values used throughout the map or topic on the `@audience` attribute.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attributes defined below.

@type
: Specifies the type of audience for whom the content of the topic is intended. Note that this differs from the `@type` attribute on many other DITA elements. For example, possible values enumerated in earlier versions of DITA included "user", "purchaser", "administrator", "programmer", "executive", and "services".

@job
: Specifies the high-level task the audience for the topic is trying to accomplish. Different audiences might read the same topic in terms of different high-level tasks; for example, an administrator might read the topic while administering, while a programmer might read the same topic while customizing.

For example, possible values enumerated in earlier versions of DITA included "installing", "customizing", "administering", " programming", " using", "maintaining", "troubleshooting", "evaluating", "planning", and "migrating".

@experiencelevel
Indicates the level of experience the audience is assumed to possess. Different audiences might have different experience levels with respect to the same topic; for example, a topic might require general knowledge from a programmer, but expert knowledge from a user. For example, possible values enumerated in earlier versions of DITA included "novice", "general", and "expert".

@name
Specifies a name for the audience, which can be used in `@audience` attributes.

Example

When used in a reference topic, the following code sample declares that the topic is intended for experienced programmers.

```
<audience type="programmer" job="programming" experiencelevel="expert"/>
```

10.5.1.2 <author>

An `<author>` element specifies the author of a topic.

Usage information

The author is usually the person, organization, or application that created the content. This element is equivalent to the `<Creator>` element in Dublin Core.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with a narrowed definition for `@type`, given below), and `@keyref` (388).

@type
Describes the type of author. Note that this differs from the `@type` attribute on many other DITA elements. For example, possible values enumerated in earlier versions of DITA included "creator" and "contributor".

Example

The following code sample shows two authors, with Jane listed as a creator of the topic and John listed as a contributor to the topic.

```
<prolog>
  <author type="creator">Jane</author>
  <author type="contributor">John</author>
</prolog>
```

10.5.1.3 <brand>

A `<brand>` element specifies the manufacturer or brand associated with the product described by the parent `<prodinfo>` element.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows that product "MyMedDevice" is associated with the brand "ExampleCo".

```
<prodinfo>
  <prodname>MyMedDevice</prodname>
  <vrmlist><vrm version="1"/></vrmlist>
  <brand>ExampleCo</brand>
</prodinfo>
```

10.5.1.4 <category>

A `<category>` element specifies any category by which a topic might be classified for retrieval or navigation.

Usage information

Topics can belong to multiple categories. Such classifications are likely to come from an enumerated or hierarchical set. For example, the categories could be used to group topics in a generated navigation bar.

This element is equivalent to both the `<Coverage>` element and the `<Subject>` element in Dublin Core.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows how a topic is associated with three categories.

```
<prolog>
 <metadata>
  <category>History</category>
  <category>Non-fiction</category>
  <category>Exciting</category>
 </metadata>
</prolog>
```

10.5.1.5 <component>

A `<component>` element specifies the component of the product that this topic is concerned with.

Usage information

A product might be made up of many components, each of which is installable separately. Components might also be shared by several products so that the same component is available for installation with many products.

Processing expectations

An implementation might use this identification to check cross-component dependencies when some components are installed, but not others. An implementation might use the identification to make sure that topics are hidden, removed, or flagged in some way when the component they describe isn't installed.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows that the topic is associated with the "TCP/IP" component of the product "MyAdapter".

```
<prodinfo>
 <prodname>MyAdapter</prodname>
 <vrmlist><vrm version="2"/></vrmlist>
 <component>TCP/IP</component>
</prodinfo>
```

10.5.1.6 <copyrholder>

A <copyrholder> element specifies the copyright holder that holds legal rights to the material contained in the topic.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows that OASIS Open holds the copyright to the material in a topic.

```
<copyright>
  <copyryear year="2020"></copyryear>
  <copyrholder>OASIS Open</copyrholder>
</copyright>
```

10.5.1.7 <copyright>

A <copyright> element specifies legal ownership of the content.

Usage information

The <copyright> element is used for a single copyright entry. It includes the copyright years and the copyright holder. Multiple <copyright> statements are allowed.

This element is equivalent to the <Rights> element in Dublin Core.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attribute defined below.

@type
: Indicates the legal status of the copyright holder. Note that this differs from the @type attribute on many other DITA elements. For example, possible values enumerated in earlier versions of DITA included "primary" and "secondary".

Example

The following code sample shows that OASIS Open holds the copyright to material in a topic, with a copyright year of 2020.

```
<prolog>
 <copyright>
  <copyryear year="2020"></copyryear>
  <copyrholder>OASIS Open</copyrholder>
```

```
  </copyright>
</prolog>
```

10.5.1.8 <copyryear>

A <copyryear> element specifies the copyright year for a topic using the @year attribute.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attribute defined below.

@year
Specifies the year in YYYY format.

Example

The following code sample shows that OASIS Open holds the copyright to material in a topic, with a copyright year of 2020.

```
<prolog>
 <copyright>
  <copyryear year="2020"/>
  <copyrholder>OASIS Open</copyrholder>
 </copyright>
</prolog>
```

10.5.1.9 <created>

A <created> element specifies the document creation date using the @date attribute.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Date attributes (369), and the attribute defined below.

@date (REQUIRED)
Specifies the document creation date. The format is YYYY-MM-DD where YYYY is the year, MM is the month from 01 to 12, and DD is the day from 01-31. See A Summary of the International Standard Date and Time Notation for background.

Example

The following code sample shows that the topic was created on the twelth day of June in 2020.

```
<prolog>
 <critdates>
  <created date="2020-06-12"></created>
  <revised golive="2020-08-20" modified="2020-07-01"></revised>
 </critdates>
</prolog>
```

10.5.1.10 <critdates>

A <critdates> element specifies the critical dates in a document life cycle, such as the creation date and revision dates.

Usage information

This element is equivalent to the <Date> element in Dublin Core.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows that the topic was created on the twelth day of June in 2020, and updated on the 20th of August in 2020.

```
<prolog>
  <critdates>
    <created date="2020-06-12"></created>
    <revised modified="2020-08-20"></revised>
  </critdates>
</prolog>
```

10.5.1.11 <featnum>

A <featnum> element specifies the feature number of a product.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows that the topic is associated with feature number 135 of the product "MyController".

```
<prodinfo>
  <prodname>MyController</prodname>
  <vrmlist><vrm version="2"/></vrmlist>
  <featnum>135</featnum>
  <component>TCP/IP</component>
</prodinfo>
```

10.5.1.12 <keywords>

A <keywords> element contains a list of terms that applies to the topic or map.

Usage information

Key words in <keywords> are used to help find content in the topic. For example, <keyword> elements can be supplied as key words to search engines, and <indexterm> elements can be collected for a document index.

While the <keyword> (227) element can be used inline, the <keywords> element is not an inline element. The <keywords> element only appears in the <topicmeta> or <prolog>, and is used to specify keywords that apply to the topic.

Processing expectations

All <keyword> and/or <indexterm> elements in the <keywords> element are considered part of the topic's metadata and should be reflected in the output as appropriate for the output medium.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows how several key words can be associated with a topic related to installing software.

```
<prolog>
 <metadata>
  <keywords>
   <keyword>installing</keyword>
   <keyword>uninstalling</keyword>
   <keyword>prerequisites</keyword>
   <keyword>helps</keyword>
   <keyword>wizards</keyword>
  </keywords>
 </metadata>
</prolog>
```

10.5.1.13 <metadata>

A <metadata> section is a contaner that groups metadata such as a topic's audience and key words.

Usage information

Elements inside of <metadata> provide information about the content and subject of a topic. When used in topics, other metadata elements outside of the <metadata> container generally provide lifecycle information for the content unit (such as the author or copyright), which are unrelated to the subject.

When used in maps, several metadata elements are allowed both inside and outside of the <metadata> container, with the container made available to provide parity with topic prologs.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows that a topic about jet packs is only intended for advanced users.

```
<prolog>
  <metadata>
    <audience type="user" job="flying" experiencelevel="advanced"/>
    <keywords>
      <keyword>jet pack</keyword>
      <keyword>danger</keyword>
      <keyword>don't try this</keyword>
    </keywords>
  </metadata>
</prolog>
```

10.5.1.14 <othermeta>

An <othermeta> element specifies properties not otherwise included in <metadata> using name/content values.

Processing expectations

Comment by rodaande
Is it useful to say that metadata is considered part of the topic metadata?

All <othermeta> elements are considered part of the topic's metadata and should be reflected in the output as appropriate for the output medium.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attributes defined below.

@name (REQUIRED)
Specifies the name of the metadata property.

@content (REQUIRED)
Specifies the value for the property named in the @name attribute.

@translate-content
Indicates whether the @content attribute of the defined metadata property should be translated or not. Allowable values are "yes", "no", and -dita-use-conref-target (385).

Example

The following code sample shows that for the current topic, the metadata ThreadWidthSystem has a value of metric.

```
<othermeta name="ThreadWidthSystem" content="metric"/>
```

10.5.1.15 <permissions>

A <permissions> element specifies the level of entitlement needed to access content.

Usage information

The <permissions> element indicates any preferred controls for access to content.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attribute defined below.

@view
Specifies the classifications of viewers allowed to view the document. For example, possible values enumerated in earlier versions of DITA included "internal", "classified", "all", and "entitled".

Example

The following code sample shows that the topic is only intended for entitled readers.

```
<prolog>
  <permissions view="entitled"/>
</prolog>
```

10.5.1.16 <platform>

A <platform> element specifies the operating system or hardware related to the product being described by the parent <prodinfo> element.

Usage information

The <platform> element can be used to provide a more detailed definition of values used throughout the map or topic on the @platform attribute.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.5.1.17 prodinfo (304).

10.5.1.17 <prodinfo>

A <prodinfo> element contains information about the product or products that are the subject matter of the current topic.

Usage information

The <prodinfo> element also can be used to provide a more detailed definition of values used throughout the map or topic on the @product attribute.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows that a product is about the product "Transcription Assistant". That product, as described in this topic, is at version number 1.3.1, operates on the Linux platform, and is has the program number SN-12345T.

```
<prolog>
 <metadata>
  <prodinfo>
   <prodname>Transcription Assistant</prodname>
   <vrmlist><vrm version="1" release="3" modification="1"/></vrmlist>
   <platform>Linux</platform>
   <prognum>SN-12345T</prognum>
  </prodinfo>
 </metadata>
</prolog>
```

10.5.1.18 <prodname>

A <prodname> element specifies the name of the product that is supported by the information in this topic.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.5.1.17 prodinfo (304).

10.5.1.19 <prognum>

A program number is an order number or a product tracking code

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.5.1.17 prodinfo (304).

10.5.1.20 <prolog>

The prolog contains metadata about the topic, for example, author information or subject category.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows a <prolog> element that contains common metadata items:

```
<prolog>
  <author>Paul Norman</author>
  <copyright>
    <copyryear year="1930"/>
    <copyrholder>Paul Norman</copyrholder>
  </copyright>
</prolog>
```

10.5.1.21 <publisher>

A publisher is the person, company, or organization who publishes the content.

Usage information

This element is equivalent to the <Publisher> element in Dublin Core.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369), and @keyref (388).

Example

The following code sample shows that the content is published by OASIS Open Printing, Inc.

```
<prolog>
  <author>Ivan</author>
  <publisher>OASIS Open Printing, Inc.</publisher>
</prolog>
```

10.5.1.22 <resourceid>

A resource ID provides an identifier for applications that must use their own identifier scheme, such as context-sensitive help systems and databases.

Usage information

The `@appid` and `@appname` attributes are available to associate an ID with an application. Multiple `@appid` values can be associated with a single `@appname` value, and multiple `@appname` values can be associated with a single `@appid` value. Because the values for the `@appid` and `@appname` attributes work in combination to specify a specific ID for a specific application, each combination of values for the `@appid` and `@appname` attributes should be unique within the context of a single root map.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attributes defined below.

@appname
Specifies a name for the external application that references the topic.

@appid
Specifies an ID used by an application to identify the topic.

@ux-context-string
Specifies the value of a user-assistance context-string that is used to identify the topic.

@ux-source-priority

Specifies precedence for handling `<resourceid>` definitions that exist in both a map and a topic. This attribute only is valid when used within a `<topicref>` element in a map. The allowable values are "-dita-use-conref-target" and the following:

topic-and-map
Use IDs from both the topic and map.

topic-only
Use IDs from the topic only.

map-only
Use IDs from the map only.

map-takes-priority
Use the IDs from the map (if they exist); otherwise, use IDs from the topic.

topic-takes-priority
Use the IDs from the topic (if they exist); otherwise, use IDs from the map.

@ux-windowref
References the `@name` attribute on the `<ux-window>` element that is used to display the topic when called from a help API.

Example

In the following example, user-assistance context hooks are applied to three topics that are referenced from a DITA map. The second topic has two hooks for the same topic.

```
<map title="Widget Help">
 <topicref href="file_ops.dita" type="concept">
   <topicref href="saving.dita" type="task">
     <topicmeta>
     <resourceid appname="ua" appid="1234" ux-context-string="idh_filesave"
     ux-source-priority="topic-only" />
     </topicmeta>
   </topicref>
   <topicref href="deleting.dita" type="task">
     <topicmeta>
      <resourceid appname="ua"
           appid="2345" ux-context-string="idh_filedelete" />
      <resourceid appname="ua"
           appid="6789" ux-context-string="idh_filekill" />
     </topicmeta>
   </topicref>
   <topicref href="editing.dita" type="task">
     <topicmeta>
       <resourceid appname="ua"
            appid="5432" ux-context-string="idh_fileedit" ux-windowref="csh"  />
     </topicmeta>
   </topicref>
</topicref>
</map>
```

In the following example, a user-assistance context hook is defined in the prolog of a task topic. The context hook is made up of a context ID (value for `@appid` attribute) and a context string (value for `@ux-context-string` attribute). A user-assistance window profile also is referenced for this topic.

```
<task id="fedt">
 <title>Editing a File</title>
 <prolog>
   <resourceid appname="ua"
         appid="5432" ux-context-string="idh_fileedit" ux-windowref="csh" />
 </prolog>
 <taskbody>
  <context>After you have created a new file, you can edit it.</context>
  <steps>
   <step><cmd>Open...</cmd></step>
   <step><cmd>Edit...</cmd></step>
   <step><cmd>Save...</cmd></step>
  </steps>
 </taskbody>
</task>
```

Related concepts

Context hooks and window metadata for user assistance (35)

Context hook information specified in the `<resourceid>` element in the DITA map or in a DITA topic enables processors to generate the header, map, alias and other types of support files that are required to integrate the user assistance with the application. Some user assistance topics might need to be displayed in a specific window or viewport, and this windowing metadata can be defined in the DITA map within the `<ux-window>` element.

10.5.1.23 <revised>

Revision information is used to maintain tracking dates that are important in a development cycle, such as the date of the last modification, the original availability date, and the expiration date.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Date attributes (369), and the attribute defined below.

@modified (REQUIRED)

The last modification date, entered as YYYY-MM-DD, where YYYY is the year, MM is the month from 01 to 12, and DD is the day from 01-31.

Example

In the following example, an author specifies revision information within a topic.

```
<prolog>
  <critdates>
    <created date="2019-01-01" golive="2019-02-15" expiry="9999-09-09"/>
    <revised modified="2020-03-03" golive="2020-02-03" expiry="9999-09-09"/>
 </critdates>
</prolog>
```

The same information could be specified in a map.

```
<topicmeta>
  <critdates>
    <created date="2019-01-01" golive="2019-02-15" expiry="9999-09-09"/>
    <revised modified="2020-03-03" golive="2020-02-03" expiry="9999-09-09"/>
 </critdates>
</topicmeta>
```

10.5.1.24 <series>

A `<series>` element specifies information about the product series that the topic supports.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows that the topic is associated with series X of the product "myLearningApp".

```
<prodinfo>
 <prodname>myLearningApp</prodname>
 <vrmlist><vrm version="5"/></vrmlist>
 <series>X</series>
 <component>TCP/IP</component>
</prodinfo>
```

10.5.1.25 <source>

Source information identifies a resource from which the present topic is derived, either completely or in part.

Usage information

The `<source>` element contains a description of the resource. Alternatively, the `@href` or `@keyref` attributes can be used to reference a description of the resource.

This element is equivalent to the `<Source>` element in Dublin Core.

Processing expectations

It is implementation-dependent what it means when the element has both content and an attribute-based reference to another resource.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with a narrowed definition for `@href`, given below), and `@keyref` (388).

@href
: Provides a reference to a resource from which the present resource is derived. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications.

Example

The following code sample shows that the content is based on information from "Somewhere, someplace".

```
<prolog>
 <source>Somewhere, someplace</source>
</prolog>
```

10.5.1.26 <vrmlist>

A `<vrmlist>` element contains one or more `<vrm>` elements for logging the version, release, and modification information for products to which the topic applies.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows the version numbers associated with "Widge-o-matic", as described in the topic.

```
<prolog>
 <metadata>
  <prodinfo>
   <prodname>Widge-o-matic</prodname>
   <vrmlist>
     <vrm version="1" release="2" modification="0"/>
     <vrm version="1" release="2" modification="1"/>
   </vrmlist>
  </prodinfo>
```

```
  </metadata>
 </prolog>
```

This indicates that the topic covers Version 1, release 2, modification levels 0 and 1 (often expressed as version 1.2.0 and 1.2.1).

10.5.1.27 <vrm>

A <vrm> element specifies information about a product's version, modification, and release, to which the current topic applies.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attributes defined below.

@version (REQUIRED)
Specifies the released version number of the product(s) that the document describes.

@release
Specifies the product release identifier.

@modification
Specifies the modification level of the current version and release.

Example

The following code sample shows the version numbers associated with "Widge-o-matic", as described in the topic.

```
<prolog>
 <metadata>
  <prodinfo>
   <prodname>Widge-o-matic</prodname>
   <vrmlist>
     <vrm version="1" release="2" modification="0"/>
     <vrm version="1" release="2" modification="1"/>
   </vrmlist>
  </prodinfo>
 </metadata>
</prolog>
```

This indicates that the topic covers Version 1, release 2, modification levels 0 and 1 (often expressed as version 1.2.0 and 1.2.1).

10.5.2 Specialization elements

Several DITA elements exist either for architectural reasons or for support of specialized markup yet to be designed. Although there is little need to use these elements unless you are directed to, some of them, such as <state>, can be used if your content makes use of these semantic distinctions. For example, a discussion of signals on a gate of an integrated logic circuit might use the <state> element to represent either on or off conditions of that gate.

10.5.2.1 <data>

Data is a generic component that represents metadata within a topic or map. Complex metadata is represented by nested data structures.

Usage information

The primary purpose of the <data> element is as a specialization base. Because it can nest, it can be used to create complex metadata structures. Since it is available in both block and inline contexts, the <data> element can specify properties for most element types.

A metadata property specified using a <data> element usually applies to the structure that contains the <data> element.

When located in <prolog> and <metadata> elements, the property applies to the topic as a whole. When located in the <topicmeta> element, the property applies to the referenced topic.

CAUTION By default, processors do not render the content of the <data> element. Use the <data> element only for properties; do not use it to embed text as part of the content flow.

Rendering expectations

By default, processors *SHOULD* treat a <data> element as unknown metadata; the contents of the <data> element *SHOULD NOT* be rendered.

Processors that recognize a particular <data> element *MAY* make use of it to trigger specialized rendering.

Attributes

The following attributes are available on this element: Data-element attributes (369), Link-relationship attributes (369), 10.8.1 Universal attribute group (366), and @keyref (388).

Examples

Figure 134: Using the @name attribute on unspecialized <data> elements

The following code sample shows how the <data> element can be used to provide metadata. Rendering tools that recognize this metadata can automatically apply highlighting to the code.

```
<codeblock>
  <data name="codestyle" value="javascript"/>
  <!-- JavaScript code block -->
</codeblock>
```

Figure 135: Nesting <data> elements for complex metadata

The following code sample shows how nested <data> elements can provide complex inventory metadata for a part that is described in the topic.

```
<topic id="sample">
  <title>How to purchase items from the warehouse</title>
  <prolog>
    <data name="inventory">
      <data name="aisle" value="4"/>
      <data name="bin" value="13"/>
      <data name="restock" value="weekly"/>
    </data>
  </prolog>
  <body>
```

```
    <!-- ... -->
  </body>
</topic>
```

Figure 136: Specializing <data> for structured metadata

The following code sample contains specializations of the <data> element: <change-item>, <change-completed>, and <change-summary>. These specialized elements each provide a default for @name inside the grammar files, so that processors can work with the data without authors having to specify the attribute.

```
<topic id="data">
  <title><xmlelement>data</xmlelement></title>
  <prolog>
    <change-historylist>
      <change-item>
        <change-completed>2017-08-20</change-completed>
        <change-summary>Refactored topic to use new template</change-summary>
      </change-item>
      <change-item>
        <change-completed>2018-06-06</change-completed>
        <change-summary>Created new examples</change-summary>
      </change-item>
    </change-historylist>
  </prolog>
  <body>
    <!-- ... -->
  </body>
</topic>
```

10.5.2.2 <foreign>

The <foreign> element allows the introduction of non-DITA content, such as MathML, SVG, or Rich Text Format (RTF).

Usage information

The <foreign> element or a specialization can contain more than one type of non-DITA content or a mix of DITA and non-DITA content. Specialization of the <foreign> element is typically implemented as a domain, but it is also possible to implement foreign vocabularies as structural specializations.

If alternate content is desired, use or specialize the <desc> element inside of the <foreign> specialization. Such alternate content needs to be valid wherever the <foreign> specialization is valid. This way, if a processor is not able to recognize or render the <foreign> content itself, it can use the alternate content from <desc>.

Processing expectations

Processors should attempt to display <foreign> content unless otherwise instructed. If the processor cannot render the content, it *MAY* issue a warning.

The enabler of the foreign vocabulary must provide the processing and override the base processing for <foreign>.

- If <foreign> contains more than one alternative content element, they should all be processed. In the case of <desc> they should be concatenated in a similar way to <section>, but with no title (analogous to in HTML).
- If no <desc>, <object>, or <image> element is found within an instance of the <foreign> element, the base processing can emit a warning about the absence of processable content.

- The base processing for <object> might emit the content of <foreign> as a file at the location specified by the @data attribute of the <object> element. The <object> element should have a data attribute or a <foreign> sub-element but not both. In the event that an <object> element contains both a data attribute and an <foreign> sub-element the processing system should ignore one of them.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows a specialization of <foreign> used to hold SVG elements. For this to work, the specialization module that defines <svg> also needs to include definitions for the SVG elements.

```
<p>... as in the formula
  <svg>
    <svg:svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">

<ellipse cx="300" cy="150" rx="200" ry="80"
style="fill:rgb(200,100,50);
stroke:rgb(0,0,100);stroke-width:2"/>

    </svg:svg>
  </svg>.
</p>
```

10.5.2.3 <no-topic-nesting>

The <no-topic-nesting> element is a placeholder in the DITA architecture.

Usage information

The <no-topic-nesting> element is not intended for use in DITA source files.

The <no-topic-nesting> element is designed for use only when configuring a document-type shell where the information designer wants to eliminate the ability to nest topics. When a specialized topic is set up to allow topic nesting, the <no-topic-nesting> element can be used in a shell as a way to disallow that nesting.

Processing expectations

The <no-topic-nesting> element has no associated output processing.

Attributes

The following attribute is available on this element: class (not for use by authors) (367).

Example

This element is not intended to be used in source files.

In the "base topic" DTD document-type shell distributed by OASIS, the `%topic-info-types;` entity is set to `topic`, meaning that `<topic>` elements can nest other `<topic>` elements. The following code sample shows how `<no-topic-nesting>` is used instead to disallow nesting.

```
<!ENTITY % topic-info-types
             "no-topic-nesting"
>
```

In that shell, topics can no longer nest other topics. DTD grammar rules require that some element be specified in this entity, so `<no-topic-nesting>` is used as a placeholder to replace any nested topics.

10.5.2.4 <state>

The `<state>` element specifies a name/value pair whenever it is necessary to represent a named state that has a variable value.

Usage information

The element is primarily intended for use in specializations to represent specific states. For example, it could be used to represent logic circuit states, chemical reaction states, or airplane instrumentation states.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attributes defined below.

@name (REQUIRED)
Specifies the name of the property whose state is being described.

@value (REQUIRED)
Specifies the state of the property identified by the `@name` attribute.

Example

The following code sample shows how `<state>` can be used to indicate a "high" value for a flow rate in a device.

```
<note type="warning">If the rate of flow reaches "high" on your FancyMeasureDevice
(<state name="flowrate" value="high"), take quick action or you will
risk dangrous levels of exposure.</note>
```

10.5.2.5 <unknown>

The `<unknown>` element is an open extension that enables information architects to incorporate XML fragments.

Processing expectations

Processors should ignore this element unless otherwise instructed.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

This example features a specialized <unknown> element that includes other non-DITA content. If this specialization is imported to a DTD or schema, the DTD or schema will need to handle declaring the new elements or any namespaces.

By definition, the content of <unknown> can only be understood by DITA processors as unknown XML. This means that processors would generally ignore this content unless they are configured to recognize the <my-unknown> specialization.

```
<body>
  <my-unknown class="+ topic/unknown mything/my-unknown ">
    <thing value="4"/>
    <otherthing value="16"/>
  </my-unknown>
</body>
```

10.6 Domain elements

General purpose domains are not specific to any type of information, such as the hazard statement domain that provides elements for describing hazardous situations.

10.6.1 Alternative titles domain elements

The alternative title elements are used to describe alternative titles for resources for use in specific use cases.

10.6.1.1 <linktitle>

A link title is an alternative title for a resource to be used with references to the resource generated from relationships defined in map structures, such as relationship tables and parent-to-child links.

Usage information

Link titles are alternative titles for use in cases where a hyperlink or cross-reference to a resource is generated based on relationships described by a DITA map. This includes, but is not limited to, the following cases:

- Links from a topic to its child topics in the map hierarchy.
- Links from a topic to its parent topic in the map hierarchy.
- Links between sibling topics when the parent <topicref> element's @collection-type is sequence or family.

Processors may use this title for other custom linking scenarios.

Processing expectations

The <linktitle> element is a specialization of <titlealt> with the @title-role set to linking. Processing is dictated by the rules for <titlealt>.

Specialization hierarchy

The <linktitle> element is specialized from <titlealt>. It is defined in the alternativeTitle-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), `@class` (367), `@outputclass` (368), and the `@title-role` attribute set to `linking`.

Examples

This section contains examples of how the `<linktitle>` element can be used.

Figure 137: Link title within a map

The following code sample shows how a `<linktitle>` element can be used within a `<topicref>` to provide text for a related link to a non-DITA resource:

```
<topicref href="SQLJ-example.html" format="html" scope="local">
  <topicmeta>
    <linktitle>Accessing relational data with SQLJ</linktitle>
  </topicmeta>
</topicref>
```

Figure 138: Link title within a topic

The following code sample shows how a `<linktitle>` element can be used within a topic's prolog to provide text for generated links to this topic. Note that this `<linktitle>` may be overridden by any `<topicref>` element that references this topic via its `<topicmeta>`.

```
<topic id="topic">
  <title>Circuitry in the C-283 Drive Train</title>
  <prolog>
    <linktitle>Drive Train Circuitry</linktitle>
  </prolog>
```

10.6.1.2 <navtitle>

A navigation title is an alternative title for a resource; this alternate title is optimized for situations where the topic title is unsuitable for use in a table of contents or navigation pane.

Usage information

Navigation titles specify the title for use in tables of contents and other navigation aids. In some cases, when processing a `<topicref>` that has no `@href`, this should also be used as the title of the generated pseudo-topic, if applicable.

Processing expectations

The `<navtitle>` element is a specialization of `<titlealt>` with the `@title-role` set to `navigation`. Processing is dictated by the rules for `<titlealt>`.

Specialization hierarchy

The `<navtitle>` element is specialized from `<titlealt>`. It is defined in the alternativeTitle-domain module.

Attributes

Examples

This section contains examples of how the `<navtitle>` element can be used.

Figure 139: <navtitle> in a topic

The following code sample shows a `<navtitle>` element used in a topic. The `<navtitle>` element contains a shorter title that processors render in a TOC or navigation pane when the topic is published.

```
<task id="publishing-dita">
  <title>Publishing a DITA information set in PDF</title>
  <shortdesc>You can quickly publish your DITA information to PDF.
  </shortdesc>
  <prolog>
    <navtitle>Publishing in PDF</navtitle>
  </prolog>
  <!-- ... -->
</task>
```

Figure 140: <navtitle> in a map

The following code sample shows `<navtitle>` elements used in a DITA map. The navigation title in the map takes precedence over any navigation titles specified in the topic.

```
<map xml:lang="en">
  <title>Publishing a DITA information set</title>
  <topicref href="GUID-gibberish.dita">
    <topicmeta>
      <navtitle>Publishing to PDF</navtitle>
    </topicmeta>
  </topicref>
</map>
```

10.6.1.3 <searchtitle>

A search title is a title that is displayed by search tools.

Usage information

A search title is useful when the topic has a title that makes sense in the context of a single information set, but might be too general in a list of search results. For example, a topic title of "Markup example" might make sense as part of a guide to DITA, but when found among thousands of unrelated topics, a search title of "DITA markup example" is more useful.

Processing expectations

The `<searchtitle>` element is a specialization of `<titlealt>` with the `@title-role` set to `search`. Processing is dictated by the rules for `<titlealt>`.

Specialization hierarchy

The `<searchtitle>` element is specialized from `<titlealt>`. It is defined in the alternativeTitle-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), `@class` (367), `@outputclass` (368), and the `@title-role` attribute set to `navigation`.

Examples

This section contains examples of how the <searchtitle> element can be used.

Figure 141: Search title used in a topic

In the following code sample, the title "Programming Example" is useful in a set of information about XSLT basics; however, the same title is not helpful among a set of search results from the entire Internet. In the latter case, a title of "Example of basic programming in XSLT" will be more useful:

```
<topic id="programming-example">
  <title>Programming example</title>
  <prolog>
    <searchtitle>Example of basic programming in XSLT</searchtitle>
  </prolog>
  <body>
    <!-- ... -->
  </body>
</topic>
```

Figure 142: Search title in a map

When <searchtitle> is used in maps, the element provides a new search title for the topic when used in that specific context. For example, if the following code sample is from a map that includes information about programming in many languages, searches among that information set will be most useful when they return "Example of programming in XSLT":

```
<topicref href="programming-example.dita">
  <topicmeta>
    <navtitle>Programming example</navtitle>
    <searchtitle>Example of programming in XSLT</searchtitle>
  </topicmeta>
</topicref>
```

10.6.1.4 <subtitle>

A subtitle is an subordinate title for a resource to be used to augment the information about the resource in certain display contexts.

Usage information

Subtitles specify a subordinate title to be displayed alongside the title in certain display contexts.

Processing expectations

The <subtitle> element is a specialization of <titlealt> with the @title-role set to linking. Processing is dictated by the rules for <titlealt>.

Specialization hierarchy

The <subtitle> element is specialized from <titlealt>. It is defined in the alternativeTitle-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), @class (367), @outputclass (368), and the @title-role attribute set to subtitle.

Examples

This section contains examples of how the <subtitle> element can be used.

Figure 143: Subtitle within a map

The following code sample shows how a map may carry a subtitle for the publication.

```
<map>
  <title>Map title</title>
  <topicmeta>
    <subtitle>Sub title</subtitle>
  </topicmeta>
</map>
```

Figure 144: Subtitle within a topic

The following code sample shows how a topic may carry a subtitle.

```
<topic id="topic">
  <title>Getting started</title>
  <prolog>
    <subtitle>An introduction to the Acme Inc. processing system</subtitle>
  </prolog>
```

10.6.1.5 <titlehint>

A title hint provides information to map authors about the title of the referenced resource.

Usage information

Title hints are intended to provide the title of referenced resources to map authors when the title may not be immediately accessible. It is not rendered as part of the publication.

Processing expectations

The <titlehint> element is a specialization of <titlealt> with the @title-role set to hint. Processing is dictated by the rules for <titlealt>.

Specialization hierarchy

The <titlehint> element is specialized from <titlealt>. It is defined in the alternativeTitle-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), @class (367), @outputclass (368), and the @title-role attribute set to hint.

Examples

This section contains examples of how the <titlehint> element can be used.

Figure 145: Referencing remote DITA resources

The following code sample shows how <titlehint> can be used to show the title of a referenced topic to map authors in the context of a CMS with opaque URIs.

```
<topicref href="x-id://AOE82KJAW1B0">
  <topicmeta>
```

```
    <titlehint>Getting started</titlehint>
  </topicmeta>
</topicref>
```

10.6.2 Classification domain elements

The classification domain elements are used to identify the subject matter of content that is referenced in a map. The subjects are defined in a subject scheme map.

10.6.2.1 <subjectCell>

The `<subjectCell>` element contains subjects that are associated with topics in the first column of the row in the `<topicSubjectTable>`. The subjects themselves have no defined relationship across columns, other than the fact that they apply to the same content.

Specialization hierarchy

The `<subjectCell>` element is specialized from `<relcell>`. It is defined in the classification-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and Common map attributes (369). This element also uses `@scope`, `@format`, and `@type` from Link-relationship attributes (369).

Example

See 10.6.2.8 topicSubjectTable (324)

10.6.2.2 <subjectref>

The `<subjectref>` element identifies a subject with which to classify the content.

Specialization hierarchy

The `<subjectref>` element is specialized from `<topicref>`. It is defined in the classification-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with a narrowed definition of `@href`, given below), `@keyref` (388), and `@keys` (389). This element also uses `@collection-type`, `@linking`, and narrowed definitions of `@processing-role` and `@toc` (given below), from Common map attributes (369).

@href
: Specifies the referenced resource. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications. References to non-DITA content need to use the `@format` attribute to identify the kind of resource.

@processing-role
: Specifies the role that the resource plays in processing. By default, this is set to "resource-only". Otherwise, the definition matches the one in Common map attributes (369).

@toc
: Specifies whether the resource appears in the table of contents (TOC). If the value is not specified locally, but is specified on an ancestor, the value cascades from the closest containing element. By

default, the value for `@toc` is set to "no". See Common map attributes (369) for a complete definition of `@toc`.

Example

In the following example, the map is classified as covering the Linux subject, and `developing-web-applications.dita` is classified as covering the Web and development subjects. These subjects (and their keys) are defined externally in a subject scheme map; in order to reference the subject directly without the subject scheme map, the `@href` attribute would be used in place of `@keyref`.

```
<map>
  <title>Working with Linux</title>
  <topicsubject keyref="linux"/>
  <!-- ... -->
  <topicref href="developing-web-applications.dita">
    <topicsubject>
      <subjectref keyref="web"/>
      <subjectref keyref="development"/>
    </topicsubject>
    <!-- ... -->
  </topicref>
  <!-- ... -->
</map>
```

10.6.2.3 <topicapply>

The `<topicapply>` element identifies subjects that qualify the content for filtering or flagging but not retrieval. The `<topicapply>` element can identify a single subject. Additional subjects can be specified by nested `<subjectref>` elements.

Specialization hierarchy

The `<topicapply>` element is specialized from `<topicref>`. It is defined in the classification-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with a narrowed definition of `@href`, given below), `@keyref` (388), and `@keys` (389). This element also uses `@collection-type`, `@linking`, and narrowed definitions of `@processing-role` and `@toc` (given below), from Common map attributes (369).

@href
Specifies the referenced resource. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications. References to non-DITA content need to use the `@format` attribute to identify the kind of resource.

@processing-role
Specifies the role that the resource plays in processing. By default, this is set to "resource-only". Otherwise, the definition matches the one in Common map attributes (369).

@toc
Specifies whether the resource appears in the table of contents (TOC). If the value is not specified locally, but is specified on an ancestor, the value cascades from the closest containing element. By default, the value for `@toc` is set to "no". See Common map attributes (369) for a complete definition of `@toc`.

Example

The map content should be retrieved for Apache Tomcat and hidden as irrelevant for operating systems other than Red Hat or SUSE.

```
<map>
  <title>Installing Apache Tomcat on RedHat or SuSE Linux</title>
  <topicsubject href="../controlledValues/tomcatServer.dita"/>
  <topicapply>
    <subjectref href="../controlledValues/redhatLinux.dita"/>
    <subjectref href="../controlledValues/suseLinux.dita"/>
  </topicapply>
  <!-- ... -->
</map>
```

10.6.2.4 <topicCell>

The `<topicCell>` element contains topics that are associated with subjects in each following column of the row in the `<topicSubjectTable>`.

Specialization hierarchy

The `<topicCell>` element is specialized from `<relcell>`. It is defined in the classification-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and Common map attributes (369). This element also uses `@scope`, `@format`, and `@type` from Link-relationship attributes (369).

Example

See 10.6.2.8 topicSubjectTable (324)

10.6.2.5 <topicsubject>

The `<topicsubject>` element identifies the subjects that are covered by a topic or map.

Usage information

To identify a primary subject, refer to the subject with the `<topicsubject>` element itself. Secondary subjects can be specified by nested `<subjectref>` elements.

Specialization hierarchy

The `<topicsubject>` element is specialized from `<topicref>`. It is defined in the classification-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with a narrowed definition of `@href`, given below), `@keyref` (388), and `@keys` (389). This element also uses narrowed definitions of `@processing-role` and `@toc` (given below) from Common map attributes (369).

@href
Specifies the referenced resource. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications. References to non-DITA content need to use the `@format` attribute to identify the kind of resource.

@processing-role
Specifies the role that the resource plays in processing. By default, this is set to "resource-only". Otherwise, the definition matches the one in Common map attributes (369).

@toc
Specifies whether the resource appears in the table of contents (TOC). If the value is not specified locally, but is specified on an ancestor, the value cascades from the closest containing element. By default, the value for `@toc` is set to "no". See Common map attributes (369) for a complete definition of `@toc`.

Example

In the following example, the map is classified as covering the Linux subject, and `developing-web-applications.dita` is classified as covering the Web and development subjects. These subjects (and their keys) are defined externally in a subject scheme map; in order to reference the subject directly without the subject scheme map, the `@href` attribute would be used in place of `@keyref`.

```
<map>
  <title>Working with Linux</title>
  <topicsubject keyref="linux"/>
  <!-- ... -->
  <topicref href="developing-web-applications.dita">
    <topicsubject>
      <subjectref keyref="web"/>
      <subjectref keyref="development"/>
    </topicsubject>
    <!-- ... -->
  </topicref>
  <!-- ... -->
</map>
```

10.6.2.6 <topicSubjectHeader>

The `<topicSubjectHeader>` element represents a header row in the topic subject table.

Usage information

Use the `<topicSubjectHeader>` element to supply a header row for a topic subject table when you want to classify topics with subjects from different categories, a practice also known as facet classification. Each cell in the header row identifies the subject for a different category. As a best practice, the subjects in the same column within the classification rows must appear in the category in the subject scheme. For instance, if the cell within the header row specifies the Operating System category, the subjects in the column must be kinds of operating systems.

Specialization hierarchy

The `<topicSubjectHeader>` element is specialized from `<relrow>`. It is defined in the classification-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.6.2.8 topicSubjectTable (324).

10.6.2.7 <topicSubjectRow>

The `<topicSubjectRow>` element represents a single row of a topic subject table. It contains topic references in the first column and subject references in each following column.

Specialization hierarchy

The `<topicSubjectRow>` element is specialized from `<relrow>`. It is defined in the classification-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.6.2.8 topicSubjectTable (324).

10.6.2.8 <topicSubjectTable>

The `<topicSubjectTable>` element represents a specialized relationship table that associates topics with subjects. Search tools might use these classifications to retrieve content that is relative to a specific subject or combination of subjects.

Usage information

In a `<topicSubjectTable>`, the first column is reserved for references to content. Subsequent columns are reserved for subjects that classify the content; each column supplies the subjects for the category that is identified in the header. The table resembles a traditional relationship table in which the first column identifies the source and the other columns identify the targets, but the relationship reflects the subjects covered by the content rather than linking between documents.

In a `<reltable>`, topics in any given column establish relationships with topics in every other cell of the same row. In a `<topicSubjectTable>`, topics in the first column are related to all of the subjects in the row, but no relationship is implied between subjects in different columns of the same row. (Relationships between subjects are defined in the `<subjectRelTable>` in the subject scheme map.)

Specialization hierarchy

The `<topicSubjectTable>` element is specialized from `<reltable>`. It is defined in the classification-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and Common map attributes (369) (with a narrowed definition of `@toc`, given below). This element also uses `@type`, `@scope`, and `@format` from Link-relationship attributes (369).

@toc

Specifies whether the resource appears in the table of contents (TOC). If the value is not specified locally, but is specified on an ancestor, the value cascades from the closest containing element. By

default, the value for `@toc` is set to "no". See Common map attributes (369) for a complete definition of `@toc`.

Example

The topic subject table below associates topics with goals for retrieval and with operating systems for filtering. The subjects are defined in a separate subject scheme map.

Figure 146: Subject scheme map

```
<subjectScheme>
    <hasKind>
        <subjectdef href="goalType.dita" keys="goal">
            <subjectdef href="performanceGoal.dita" keys="performance"/>
            <subjectdef href="reliabilityGoal.dita" keys="reliability"/>
        </subjectdef>
        <subjectdef href="operatingSystem.dita" keys="os">
            <subjectdef href="linuxOS.dita" keys="linux"/>
            <subjectdef href="unixOS.dita" keys="unix"/>
            <subjectdef href="windowsOS.dita" keys="windows"/>
        </subjectdef>
    </hasKind>
</subjectScheme>
```

Figure 147: Topic subject table

The following `<topicSubjectTable>` classifies several topics according to subjects defined in the previous map. As with any `<topicSubjectTable>`, the first column is used to specify topics. In this specific example, the second column is used to specify a goal, based on the "goal" subject in the header. The third column is used to specify an operating system. Based on those definitions, the following classifications are made by this table:

- The topics `configure-cron-for-efficiency.dita` and `allocating-raw-storage.dita` are each classified by the goal of "performance"; they also are classified by the operating systems "linux" and "unix".
- The topics `analyze-web-logs.dita` and `detect-denial-of-service-attacks.dita` are each classified by the goal of "reliability"; they also are classified by the operating systems "linux", "unix", and "windows".
- No relationship is defined between subjects in the table, meaning that this table does not define any relationship between the goal of "performance" and the operating systems "linux" or "unix".

```
<map>
<!-- ... -->
<topicSubjectTable>
  <topicSubjectHeader>
    <topicCell type="task"/>
    <subjectCell>
      <topicsubject keyref="goal"/>
    </subjectCell>
    <subjectCell>
      <topicapply keyref="os"/>
    </subjectCell>
  </topicSubjectHeader>
  <topicSubjectRow>
    <topicCell>
      <topicref href="configure-cron-for-efficiency.dita"/>
      <topicref href="allocating-raw-storage.dita"/>
    </topicCell>
    <subjectCell>
      <topicsubject keyref="performance"/>
    </subjectCell>
    <subjectCell>
      <topicapply keyref="linux"/>
      <topicapply keyref="unix"/>
    </subjectCell>
```

```
  </topicSubjectRow>
  <topicSubjectRow>
    <topicCell>
      <topicref href="analyze-web-logs.dita"/>
      <topicref href="detect-denial-of-service-attacks.dita"/>
    </topicCell>
    <subjectCell>
      <topicsubject keyref="reliability"/>
    </subjectCell>
    <subjectCell>
      <topicapply keyref="linux"/>
      <topicapply keyref="unix"/>
      <topicapply keyref="windows"/>
    </subjectCell>
  </topicSubjectRow>
  <!-- ... -->
</topicSubjectTable>
</map>
```

A table view of this `<topicSubjectTable>` might look as follows. This is only one of many possible views; to aid in understanding the example, the content topics in the first column are displayed using only their file names, and related subjects are displayed using only their `@keyref` attribute value.

Task	Goal	Operating system

`configure-cron-for-efficiency.dita`
 	
`allocating-raw-storage.dita`
	performance	linux
		unix

`analyze-web-logs.dita`
 	
`detect-denial-of-service-attacks.dita`
	reliability	linux
		unix
		windows

10.6.3 DITAVAL-reference domain element

The DITAVAL reference domain is used to reference a DITAVAL file that contains conditions that apply only to a subset of a DITA map. It also can be used to replicate a subset of a DITA map for multiple audiences.

10.6.3.1 <ditavalref>

The `<ditavalref>` element references a DITAVAL document that specifies filter conditions that can be used to process a map or map branch. Other DITAVAL-reference domain elements can be used to imply multiple copies of the map branch that contains them and so apply a different set of conditions to each copy.

When a `<ditavalref>` element is included in a map, the conditions in the referenced DITAVAL document are used to filter the elements in the branch. The branch includes the parent element that contains the `<ditavalref>` element, any child elements, and all resources that are referenced by the parent element or its children. While there is no technical restriction that forces `<ditavalref>` to appear before peer topic references, placing them first is considered a best practice and all examples in the specification will use this convention.

In the simple case, a map can use `<ditavalref>` as follows:

```
<map>
  <topicref href="sampleBranch.dita" audience="admin">
```

```
    <topicmeta>
      <navtitle>Navigation title for branch</navtitle>
    </topicmeta>
    <ditavalref href="conditions.ditaval"/>
    <topicref href="insideBranch.dita" platform="win linux mac"/>
  </topicref>
  <!-- Other branches not affected by conditions.ditaval -->
</map>
```

The filtering conditions specified in the `conditions.ditaval` file apply to the following:

- The `<topicref>` element that references `sampleBranch.dita` and all child elements: `<topicmeta>`, `<navtitle>`, and `<topicref>` elements
- The `sampleBranch.dita` topic
- The `insideBranch.dita` topic

When more than one `<ditavalref>` element is specified in the same branch at the same level, the effective result is one copy of the branch for each `<ditavalref>` element. If the example above contains a reference to `otherConditions.ditaval` as a peer to the existing `<ditavalref>` element, the rendered version of this map would reflect two copies of "Sample branch", each reflecting the conditions that are specified in the corresponding DITAVAL document. One copy is created using the conditions in `conditions.ditaval`, while the other copy uses the conditions from `otherConditions.ditaval`. Map authors can use specific elements from the DITAVAL reference domain to indicate how resources are renamed, or processors can recover from naming collisions by using an alternate naming scheme. See Limitations (327) below for more information.

If DITAVAL conditions are specified at multiple levels within a single branch, "exclude" conditions that are specified at a higher level take precedence. In the following branch, assume alternate rules are specified for the condition `audience="novice"`, with the value set to "exclude" in `highLevel.ditaval` and "include" in `lowLevel.ditaval`. In that case, the "exclude" condition specified in `highLevel.ditaval` takes precedence and so applies to the entire branch. This is true regardless of how the "exclude" condition is specified within `highLevel.ditaval`. That is, there might be a specific rule for `audience="novice"`; alternatively, the `@audience` attribute might be set to "exclude" by default, with no specific condition specified for the value `audience="novice"`.

```
<topicref href="ancestor.dita">
  <ditavalref href="highLevel.ditaval"/>
  <topicref href="descendent.dita">
    <ditavalref href="lowLevel.ditaval"/>
    <!-- Other topicrefs -->
  </topicref>
</topicref>
```

If a `<ditavalref>` element is used that does not specify the `@href` attribute, the element is still processed but no additional filtering is applied. This can be used to create an unfiltered copy of a map branch alongside other filtered copies; other aspects of the `<ditavalref>` (such as any specified key scope or modified resource name) will still be applied to the branch.

Limitations

The following limitations apply when using the `<ditavalref>` element; these limitations cannot be enforced in a DTD or other XML grammar files.

When the use of the `<ditavalref>` element results in multiple copies of a branch, resource names within that branch can be controlled with sub-elements of the effective `<ditavalref>`. For situations where resource names are relevant, it is an error condition for multiple `<ditavalref>` elements to result

in conflicting resource names for different content. For example, the following map fragment would result in two distinct copies of the `c.dita` topic with the same file name:

```
<topicref href="c.dita">
  <ditavalref href="one.ditaval"/>
  <ditavalref href="two.ditaval"/>
</topicref>
```

Processors *MAY* recover by using an alternate naming scheme for the conflicting copies.

Specialization hierarchy

The `<ditavalref>` element is specialized from 10.4.1.2 topicref (269). It is defined in the DITAVALref domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (except for `@conkeyref`, which is removed for all elements in this domain) and the attributes defined below.

@href
: Provides a reference to a DITAVAL document. If the `@href` attribute is unspecified, this `<ditavalref>` will not result in any new filtering behavior, but other aspects of the element are still evaluated. See 10.8.3.6 The href attribute (387) for general information on the format and processing implications of the `@href` attribute.

@format
: Format of the target document, which *MUST* be a DITAVAL document. The default value for this element is "ditaval". See 10.8.3.5 The format attribute (386) for more information.

@processing-role
: The processing role defaults to "resource-only" for DITAVAL documents, which are only used for processing and do not contain content. There is no other valid value for this attribute on this element.

Example

See 7.5.6 Examples of branch filtering (127) for several examples of the `<ditavalref>` element.

10.6.3.2 <ditavalmeta>

The `<ditavalmeta>` element defines metadata that is associated with a DITAVAL document used for one branch of a map.

Use the `<ditavalmeta>` to specify the prefixes and suffixes that processors use to construct effective resource names and key scope names within the map branch. The `<ditavalmeta>` element also can contain other information, such as navigation title, that might be useful for map architects but is not rendered in the output.

Specialization hierarchy

The `<ditavalmeta>` element is specialized from 10.4.1.3 topicmeta (270). It is defined in the DITAVALref domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (except for `@conkeyref`, which is removed for all elements in this domain).

Example

See 7.5.6 Examples of branch filtering (127) for several examples of the <ditavalref> element.

10.6.3.3 <dvrResourcePrefix>

The <dvrResourcePrefix> element specifies the prefix to use when constructing the effective file names of the resources that are referenced from within the map branch that is implied by the ancestor <ditavalref> element.

For map branches that are implied by <ditavalref> elements, the value of the <dvrResourcePrefix> element contributes to the effective file names of resources that are referenced within the branch. The effective resource file name starts with the value of the <dvrResourcePrefix> element. If @copy-to is specified on a topic reference where <dvrResourcePrefix> or <dvrResourceSuffix> based renaming is in effect, the prefixes or suffixes are applied to the resource name inside the @copy-to attribute.

Some resources are not eligible for renaming, such as those marked with scope="external". Rules for which resources are eligible for renaming, and what names are allowed as valid resource names, are the same as those for the @copy-to attribute defined in Topicref-element attributes (369).

Specialization hierarchy

The <dvrResourcePrefix> element is specialized from 10.5.2.1 data (311). It is defined in the DITAVALref domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (except for @conkeyref, which is removed for all elements in this domain) and the attribute defined below.

@name
The name of the metadata item. For this element the default value is "dvrResourcePrefix".

Example

If the <dvrResourcePrefix> is specified in the following way:

```
<topicref href="branch-01.dita">
  <ditavalref href="condition-01.ditaval">
    <ditavalmeta>
      <dvrResourcePrefix>cond01-</dvrResourcePrefix>
    </ditavalmeta>
  </ditavalref>
  <topicref href="topics/subtopic-01.dita"/>
</topicref>
```

Then the effective file name of the resource subtopic-01.dita is cond01-subtopic-01.dita, as though the @copy-to attribute had been specified with that value on the <topicref> element that references subtopic-01.dita. Similarly, the effective file name of resource branch-01.dita is cond01-branch-01.dita.

10.6.3.4 <dvrResourceSuffix>

The <dvrResourceSuffix> element specifies the suffix to use when constructing the effective file names of resources that are referenced from within the map branch that is implied by the ancestor <ditavalref> element.

For map branches that are implied by <ditavalref> elements, the value of the <dvrResourceSuffix> element contributes to the effective file names of the resources that are referenced within the branch. The base part of the effective resource file name ends with the value of the <dvrResourceSuffix> element. The base part of the resource file name consists of the portion of the file name after any directory information, and before any period followed by the file extension. For example, in the original file name task/install.dita, the base portion of the file name is "install". If @copy-to is specified on a topic reference where <dvrResourcePrefix> or <dvrResourceSuffix> based renaming is in effect, the prefixes or suffixes are applied to the resource name inside the @copy-to attribute.

Some resources are not eligible for renaming, such as those marked with scope="external". Rules for which resources are eligible for renaming, and what names are allowed as valid resource names, are the same as those for the @copy-to attribute defined in Topicref-element attributes (369), with one exception. Where @copy-to and <dvrResourcePrefix> might include path information, path information is not valid in <dvrResourceSuffix>.

Specialization hierarchy

The <dvrResourceSuffix> element is specialized from 10.5.2.1 data (311). It is defined in the DITAVALref domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (except for @conkeyref, which is removed for all elements in this domain) and the attribute defined below.

@name
The name of the metadata item. For this element the default value is "dvrResourceSuffix".

Example

If the <dvrResourceSuffix> is specified in the following way:

```
<topicref href="branch-01.dita">
  <ditavalref href="condition-01.ditaval">
    <ditavalmeta>
      <dvrResourceSuffix>-cond01</dvrResourceSuffix>
    </ditavalmeta>
  </ditavalref>
  <topicref href="topics/subtopic-01.dita"/>
</topicref>
```

Then the effective file name of resource topics/subtopic-01.dita is topics/subtopic-01-cond01.dita, as though the @copy-to attribute had been specified with that value on the <topicref> to topics/subtopic-01.dita. Similarly, the effective file name of resource branch-01.dita is branch-01-cond01.dita.

10.6.3.5 <dvrKeyscopePrefix>

The <dvrKeyscopePrefix> element specifies the prefix to use when constructing the effective key scope names for the map branch that is implied by the ancestor <ditavalref> element.

For map branches that are implied by <ditavalref> elements, the value of the <dvrKeyscopePrefix> element contributes to the effective key scope names of the branch. The effective key scope names start with the value of the <dvrKeyscopePrefix> element. Note that if the branch as authored does not specify a @keyscope value, specifying <dvrKeyscopePrefix> (without also specifying <dvrKeyscopeSuffix>) results in the branch establishing a key scope whose name is the value of the <dvrKeyscopePrefix> element. The full key scope names also will reflect the value of a <dvrKeyscopeSuffix> element if one is specified, regardless of whether the branch as authored specifies a @keyscope value.

Specialization hierarchy

The <dvrKeyscopePrefix> element is specialized from 10.5.2.1 data (311). It is defined in the DITAVALref domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (except for @conkeyref, which is removed for all elements in this domain) and the attribute defined below.

@name
The name of the metadata item. For this element the default value is "dvrKeyscopePrefix".

Example

If the <dvrKeyscopePrefix> is specified in the following way:

```
<topicref keys="branch-01"
    href="branch-01.dita"
    keyscope="branch-01"
  >
  <ditavalref href="condition-01.ditaval">
    <ditavalmeta>
      <dvrKeyscopePrefix>cond01-</dvrKeyscopePrefix>
    </ditavalmeta>
  </ditavalref>
  <topicref href="topics/subtopic-01.dita"/>
</topicref>
```

Then the effective key scope name for the branch "branch-01" is "cond01-branch-01".

10.6.3.6 <dvrKeyscopeSuffix>

The <dvrKeyscopeSuffix> element specifies the suffix to use when constructing the effective key scope names for the map branch that is implied by the ancestor <ditavalref> element.

For map branches that are implied by <ditavalref> elements, the value of the <dvrKeyscopeSuffix> element contributes to the effective key scope names of the branch. The effective key scope names end with the value of the <dvrKeyscopeSuffix> element. Note that if the branch as authored does not specify a @keyscope value, specifying <dvrKeyscopeSuffix> (without also specifying <dvrKeyscopePrefix>) results in the branch establishing a key scope whose name is the value of the <dvrKeyscopeSuffix> element. The full key scope names also will reflect the value of a <dvrKeyscopePrefix> element if one is specified, regardless of whether the branch as authored specifies a @keyscope value.

Specialization hierarchy

The <dvrKeyscopeSuffix> element is specialized from 10.5.2.1 data (311). It is defined in the DITAVALref domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (except for @conkeyref, which is removed for all elements in this domain) and the attribute defined below.

@name
The name of the metadata item. For this element the default value is "dvrKeyscopeSuffix".

Example

If the <dvrKeyscopeSuffix> is specified in the following way:

```
<topicref keys="branch-01"
    href="branch-01.dita"
    keyscope="branch-01"
  >
  <ditavalref href="condition-01.ditaval">
    <ditavalmeta>
      <dvrKeyscopeSuffix>-cond01</dvrKeyscopeSuffix>
    </ditavalmeta>
  </ditavalref>
  <topicref href="topics/subtopic-01.dita"/>
</topicref>
```

Then the effective key scope name for the branch "branch-01" is "branch-01-cond01".

10.6.4 Emphasis domain elements

The emphasis elements are used to emphasize text that is important or serious.

10.6.4.1

Emphasized text indicates stress or otherwise highlights content.

Usage information

Use this only when something more semantically appropriate is not available. For example, when indicating a different mood or voice, <i> may be more relevant.

Rendering expectations

The element is typically rendered using an italic font.

Specialization hierarchy

The element is specialized from <ph>. It is defined in the emphasis-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

How can be used in context

The following code sample shows how can be used to emphasize a phrase in a paragraph.

```
<p>A good plan once adopted and put into execution <em>should not be
abandoned</em> unless it becomes clear that it can not succeed.</p>
```

10.6.4.2

Strong text is considered to be important or serious, or has some form of urgency.

Usage information

Use this only when a more semantically appropriate alternative is not available. For example, for a specific warning, consider using something from the hazard statement domain, such as <hazardstatement>.

Rendering expectations

The element is typically rendered using a bold.

Specialization hierarchy

The element is specialized from <ph>. It is defined in the emphasis-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

How can be used in context

The following code sample shows how can be used to highlight an important detail.

```
<p>Your doctor prescribed this medicine to treat an infection.
It is important that you <strong>take all of the medicine</strong>
as described.</p>
```

10.6.5 Hazard-statement domain elements

The hazard statement domain contains the <hazardstatement> element, which can be used to provide information about product safety hazards. The domain can be included in any topic type or map. Hazard statements are used to inform readers about potential hazards, consequences, and avoidance strategies.

10.6.5.1 <consequence>

The <consequence> element specifies the consequence of failing to avoid a hazard, for example, "Contact may cause a burn."

Specialization hierarchy

The <consequence> element is specialized from . It is defined in the hazard-statement domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

In the following code sample, a hazard statement is assigned a "CAUTION" signal word. The hazard image is explicitly associated with the type of hazard.

```
<hazardstatement type="caution">
  <messagepanel>
    <typeofhazard>
      <hazardsymbol keyref="hazard-hotsurface"/>
      HOT SURFACES
    </typeofhazard>
    <consequence>Contact may cause a burn.</consequence>
    <howtoavoid>Wear gloves before servicing internal parts.</howtoavoid>
  </messagepanel>
</hazardstatement>
```

10.6.5.2 <hazardstatement>

The <hazardstatement> element contains hazard warning information. It is based on the regulations of ANSI Z535 and ISO 3864. It enables the author to select the signal word, describe the hazard and its consequences, explain how to avoid it, and add one or more safety images.

Specialization hierarchy

The <hazardstatement> element is specialized from <note>. It is defined in the hazard-statement domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and spectitle (specialization attributes) (376), and the attributes defined below.

@type
Indicates the level of hazard. The values correspond to the signal words defined by the ANSI Z535.6 standard:

caution
Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.

danger
Indicates a hazardous situation that, if not avoided, will result in death or serious injury. This signal word is to be limited to the most extreme situations.

notice
Indicates information considered important but not hazard-related, for example, messages relating to property damage.

warning
Indicates a hazardous situation that, if not avoided, could result in death or serious injury.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

Example

The following code sample defines a hazard statement with a signal word of "DANGER."

```
<hazardstatement type="danger">
  <messagepanel>
    <typeofhazard>
    <hazardsymbol keyref="hazard-rotatingblade"/>
    Rotating blade</typeofhazard>
```

```
    <consequence>Moving parts can crush and cut.</consequence>
    <howtoavoid>Follow lockout procedure before servicing.</howtoavoid>
  </messagepanel>
</hazardstatement>
```

10.6.5.3 <hazardsymbol>

The `<hazardsymbol>` element specifies a graphic. The graphic might represent a hazard, a hazardous situation, a result of not avoiding a hazard, or any combination of these messages.

Rendering expectations

Processors *SHOULD* scale the object when values are provided for the `@height` and `@width` attributes. The following expectations apply:

- If a height value is specified and no width value is specified, processors *SHOULD* scale the width by the same factor as the height.
- If a width value is specified and no height value is specified, processors *SHOULD* scale the height by the same factor as the width.
- If both a height value and width value are specified, implementations *MAY* ignore one of the two values when they are unable to scale to each direction using different factors.

Usage information

When a `<hazardsymbol>` element is directly contained by `<messagepanel>`, the `<hazardsymbol>` is assumed to be associated with the `<typeofhazard>` element.

Specialization hierarchy

The `<hazardsymbol>` element is specialized from `<image>`. It is defined in the hazard-statement domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), `@keyref` (388), and the attributes defined below.

@format
Identifies the format of the resource that is referenced. See 10.8.3.5 The format attribute (386) for details on supported values.

@href
Specifies the image. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications.

@scope
Specifies the relationship between the current document and the target resource. Allowable values are "local", "peer", "external", and -dita-use-conref-target (385). See 10.8.3.11 The scope attribute (391) for more information on values.

@height
Specifies the vertical dimension for the resulting display. The value of this attribute is a real number (expressed in decimal notation) optionally followed by a unit of measure from the set of pc, pt, px, in, cm, mm, em (picas, points, pixels, inches, centimeters, millimeters, and ems respectively). The default unit is px (pixels). Possible values include: "5", "5in", and "10.5cm".

@width
Specifies the horizontal dimension for the resulting display. The value of this attribute is a real number (expressed in decimal notation) optionally followed by a unit of measure from the set of pc, pt, px, in, cm, mm, em (picas, points, pixels, inches, centimeters, millimeters, and ems respectively). The default unit is px (pixels). Possible values include: "5", "5in", and "10.5cm".

@align
Specifies the horizontal alignment of an image when placement is specified as "break". Common values include "left", "right", and "center".

@scale
Specifies a percentage as an unsigned integer by which to scale the image in the absence of any specified image height or width; a value of 100 implies that the image should be presented at its intrinsic size. If a value has been specified for the `@height` or `@width` attribute (or both), the `@scale` attribute is ignored.

It is an error if the value of this attribute is not an unsigned integer. In this case, the implementation might give an error message and might recover by ignoring this attribute.

@scalefit
Specifies whether an image is scaled up or down to fit within available space. Allowable values are "yes", "no", and -dita-use-conref-target (385). For a given image, if any one of `@height`, `@width`, or `@scale` is specified, those attributes determine the graphic size and any setting of `@scalefit` is ignored. If none of those attributes are specified and `scalefit="yes"`, then the image is scaled by the same factor in both dimensions, so that the graphic will just fit within the available height or width (whichever is more constraining).

The available width would be the prevailing column (or table cell) width—that is, the width a paragraph of text would have if the graphic were a paragraph instead. The available height is implementation dependent, but if feasible, it is suggested to be the page (or table cell) height or some other reasonable value.

@placement
Specifies whether an image is displayed inline or on a separate line. The default value is inline. Allowable values are: "inline", "break", or and -dita-use-conref-target (385).

Example

The following code sample defines a hazard statement with a signal word of "DANGER." The `<hazardsymbol>` element is explicitly associated with the `<typeofhazard>` element.

```
<hazardstatement type="danger">
  <messagepanel>
    <typeofhazard>
    <hazardsymbol keyref="hazard-rotatingblade"/>
    Rotating blade</typeofhazard>
    <consequence>Moving parts can crush and cut.</consequence>
    <howtoavoid>Follow lockout procedure before servicing.</howtoavoid>
  </messagepanel>
</hazardstatement>
```

In the following code sample, the `<hazardsymbol>` is contained by the `<messagepanel>` element. By default, this markup also associates the `<hazardsymbol>` element with the `<typeofhazard>` element. This markup can be useful when migrating hazard statements from earlier versions to DITA 2.0.

```
<hazardstatement type="danger">
  <messagepanel>
    <typeofhazard>Rotating blade</typeofhazard>
    <consequence>Moving parts can crush and cut.</consequence>
    <howtoavoid>Follow lockout procedure before servicing.</howtoavoid>
    <hazardsymbol keyref="hazard-rotatingblade"/>
```

```
  </messagepanel>
</hazardstatement>
```

10.6.5.4 <howtoavoid>

The <howtoavoid> element contains information about how a user can avoid a hazard, for example, "Do not use solvents to clean the drum surface."

Specialization hierarchy

The <howtoavoid> element is specialized from . It is defined in the hazard-statement domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample illustrates a hazard statement with the signal word of "NOTICE", which is reserved for situations that only involve possible property damage or other undesirable states. The <howtoavoid> element provides instructions about how to avoid the undesirable state.

```
<hazardstatement type="notice">
  <messagepanel>
    <typeofhazard>
      <hazardsymbol keyref="hazard-agressivesolvent"/>
      Machinery Damage</typeofhazard>
    <howtoavoid>
      <hazardsymbol keyref="hazard-readmanual"/>
      <ul>
        <li>Do NOT use solvents to clean the drum surface</sli>
     <li>Read manual for proper drum cleaning procedure</sli>
      </ul>
    </howtoavoid>
  </messagepanel>
</hazardstatement>
```

10.6.5.5 <messagepanel>

The <messagepanel> element contains the textual information that is displayed on the hazard statement. This information identifies the hazard, specifies how to avoid the hazard, and states the probable consequences of failing to avoid the hazard.

Specialization hierarchy

The <messagepanel> element is specialized from . It is defined in the hazard-statement domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), compact (371), and spectitle (specialization attributes) (376).

Example

The following code sample generates an ANSI Z535.6 grouped safety message that specifies information about multiple hazards.

```
<hazardstatement type="warning">
  <messagepanel>
    <typeofhazard>
      <hazardsymbol keyref="hazard-electricshock"/>
      ELECTRIC SHOCK HAZARD</typeofhazard>
      <consequence>The equipment must be grounded. Improper grounding, setup, or usage of
                   the system can cause electric shock
      </consequence>
      <howtoavoid>
        <hazardsymbol keyref="hazard-groundpowersource"/>
        <ul>
          <li>Turn off and disconnect power at main switch before disconnecting any
              cables or before servicing or installing any equipment.</li>
          <li>Connect only to grounded power sources.</li>
          <li>All electric wiring must be done by a qualified electrician and comply
              with all local codes and regulations.</li>
        </ul>
      </howtoavoid>
    </messagepanel>
    ...
    <messagepanel>
      <typeofhazard>
        <hazardsymbol keyref="hazard-hotsurface"/>
        BURN HAZARD</typeofhazard>
        <consequence>Electric sufaces and fluid can become very hot during
                     operation.</consequence>
        <howtoavoid>
          To avoid burns:
          <ul>
            <li>Do not touch hot fluid or equipment.</li>
          </ul>
        </howtoavoid>
  </messagepanel>
</hazardstatement>
```

10.6.5.6 <typeofhazard>

The `<typeofhazard>` element contains a description of the type of hazard, for example, "Hot surfaces inside."

Specialization hierarchy

The `<typeofhazard>` element is specialized from `<div>`. It is defined in the hazard-statement domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample generate a "CAUTION" hazard statement, which warns users against lifting heavy objects without referring to the product safety manual.

```
<hazardstatement type="caution">
  <messagepanel>
    <typeofhazard>
      <hazardsymbol keyref="hazard-heavyobject"/>
      Lifting hazard
    </typeofhazard>
    <consequence>May result in injury.</consequence>
```

```
    <howtoavoid>See safety manual for lifting instructions.</howtoavoid>
  </messagepanel>
</hazardstatement>
```

10.6.6 Highlighting domain elements

The highlighting elements are used to highlight text with styles such as bold, italic, and monospaced. These elements are intended solely for use by authors when no semantically appropriate element is available and a formatting effect is required.

10.6.6.1 <b>

Bold text is used to draw a reader's attention to a phrase without otherwise adding meaning to the content.

Specialization hierarchy

The <b> element is specialized from <ph>. It is defined in the highlighting-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows bold highlighting used to draw a reader's attention to a phrase:

```
<p>Use the bold tag <b>for visual emphasis only</b>; do not use it if another phrase-level
element better signifies the reason for the emphasis.</p>
```

10.6.6.2 <i>

Italic text is used to emphasize the key points in printed text, or when quoting a speaker, to show which words the speaker stressed.

Specialization hierarchy

The <i> element is specialized from <ph>. It is defined in the highlighting-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows italic highlighting used to indicate a different voice, such as the use of a foreign word:

```
<note type="tip">Take care to measure the right amount when mixing ingredients.
A <i>laissez-faire</i> attitude to baking is a recipe for disaster.</note>
```

10.6.6.3 <line-through>

The <line-through> element indicates text that is rendered with a line struck through the content. This element is designed to enable authors to indicate a deletion or revision for rhetorical purpose; it is not intended to be used for indicating revisions.

Specialization hierarchy

The <line-through> element is specialized from <ph>. It is defined in the highlighting-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows line-through highlighting used to indicate a rhetorical revision:

```
<p>Line-through: DITA technology can be
      <line-through>maddening</line-through>a challenge to implement.</p>
```

10.6.6.4 <overline>

The <overline> element indicates content that is rendered with a line above it.

Specialization hierarchy

The <overline> element is specialized from <ph>. It is defined in the highlighting-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows overline highlighting used for mathematical notation:

```
<p>Overline: <overline><i>x</i></overline> is the
average value of<i>x<sub>i</sub></i></p>
```

10.6.6.5 <sub>

A subscript is text that is printed below the line. It is frequently used in chemical and mathematical formulas.

Specialization hierarchy

The <sub> element is specialized from <ph>. It is defined in the highlighting-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows how the <sub> element is used in a chemical formula:

```
<note>When cleaning, be sure to dilute the baking soda (NaHCO<sub>3</sub>) with water
(H<sub>2</sub>O) before mixing in the vinegar (CH<sub>3</sub>COOH).</note>
```

10.6.6.6 <sup>

A superscript is text that is printed above the line. It is frequently used in chemical and mathematical formulas.

Specialization hierarchy

The <sup> element is specialized from <ph>. It is defined in the highlighting-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows a <sup> element used to ensure proper formatting of the exponent in the number ten to the power of ten:

```
<p>The power produced by the electrohydraulic dam was 10<sup>5</sup> more than
the older electric plant.</p>
```

10.6.6.7 <tt>

The <tt> (teletype) indicates text that typically is rendered in monospaced highlighting.

Specialization hierarchy

The <tt> element is specialized from <ph>. It is defined in the highlighting-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows how the <tt> element can be used to apply monospaced highlighting:

```
<p>Make sure that the screen displays <tt>File successfully created</tt> before
proceeding to the next stage of the task.</p>
```

While the example demonstrates a potential use of <tt>, use <systemoutput> if you have access to the elements in the software domain.

10.6.6.8 <u>

An underline, also called an underscore, is a line immediately below a portion of text.

Specialization hierarchy

The <u> element is specialized from <ph>. It is defined in the highlighting-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

The following code sample shows underlining used to provide emphasis in a marketing blurb, without giving any extra meaning to the underlined phrase:

```
<p>Using our patented <u>SuperFast BitSpeed Technology</u>, our product
will answer all of your questions only a few nanoseconds after you ask!</p>
```

10.6.7 Mapgroup domain elements

The mapgroup domain elements define, group, or reference content. Many of the mapgroup elements are convenience elements; they simply provide shortcuts for an author to use existing markup

For example, the <topichead> element allows a map to specify a heading without a reference to a topic. While a <topicref> element might accomplish the same thing by creating a title and leaving off the @href attribute, the <topichead> element makes the intent clearer and prevents the accidental inclusion of an @href attribute.

10.6.7.1 <anchorref>

An anchor reference integrates a map branch into a point (an anchor) in the same or different DITA map. The contents of the map branch are rendered at the location of both the <anchorref> and <anchor> elements.

Usage information

The functionality of the <anchorref> element is similar to that provided by the @anchorref attribute of the <map> element. However, instead of attaching an entire map to an anchor point, this element enables the map author to attach only the contents of a single map branch. This enables map architects to reuse a branch of content without reusing the entire map.

To prevent the content of the map branch from being rendered at the location of the <anchorref> element, set toc="no" on the <anchorref> element, and then set toc="yes" on each of its children so that they will not inherit the toc="no" setting.

Rendering expectations

When possible, the content of the <anchorref> element is rendered when the map with <anchor> element is displayed in an authoring tool.

Processing expectations

If the rendering platform does not support runtime integration of the content at the anchor point, processors treat the <anchorref> element similar to a "conref push" instruction by pushing the content to the spot of the <anchor>. Note that many <anchorref> elements might push content to the same point; the order in which items are pushed is left undefined, although the order within a single <anchorref> element is preserved.

Metadata cascading takes place in the original authored context, because the branch of content defined with the <anchorref> remains independent from the referenced map. The <anchorref> content does not take on the cascading metadata at the <anchor> location. For example, if the map containing the

<anchorref> element sets a local copyright, that copyright cascades to the <anchorref> element and its children; it is retained after the content is rendered at the location of the <anchor> element.

Specialization hierarchy

The <anchoref> element is specialized from <topicref>. It is defined in the mapgroup module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with narrowed definitions of @href, @type, and @format, all given below), Common map attributes (369), Topicref-element attributes (369), @keys (389), and @keyref (388).

@href
Specifies an <anchor> element in this or another DITA map. When rendered, the contents of the current element are copied to the location of the <anchor> element. See 10.8.3.6 The href attribute (387) for the syntax to use when referencing a map element.

@type
Specifies the type of the referenced resource. By default, this is set to "anchor".

@format
Specifies the format of the referenced resource. By default, this is set to "ditamap".

Example

The section contains an example of how the <anchorref> element works.

Figure 148: Map that contains <anchor> and <anchorref> elements

The following code sample shows a DITA map that contains both <anchor> and <anchorref> elements:

```
<topicref href="carPrep.dita">
    <topicref href="beforePrep.dita"/>
    <anchor id="prepDetail"/>
    <topicref href="afterPrep.dita"/>
</topicref>
<!-- ... -->
<topicref href="astroTasks.dita">
    <topicref href="astroOverview.dita"/>
    <anchorref href="#prepDetail">
        <topicref href="astroChecklist.dita"/>
        <topicref href="otherPreparation.dita"/>
    </anchorref>
    <topicref href="astroConclusion.dita"/>
</topicref>
```

Figure 149: Effective result of evaluating the <anchorref> element

After the processor evaluates the contents of the <anchorref> element, the effective result is as follows; the content of the anchor reference is rendered both at its original location and at the location of the anchor:

```
<topicref href="carPrep.dita">
    <topicref href="beforePrep.dita"/>
    <anchor id="prepDetail"/>
    <topicref href="astroChecklist.dita"/>
    <topicref href="otherPreparation.dita"/>
    <topicref href="afterPrep.dita"/>
</topicref>
<!-- ... -->
<topicref href="astroTasks.dita">
```

```
    <topicref href="astroOverview.dita"/>
    <topicref href="astroChecklist.dita"/>
    <topicref href="otherPreparation.dita"/>
    <topicref href="astroConclusion.dita"/>
</topicref>
```

10.6.7.2 <keydef>

A key definition provides a simple way to define a key without making the definition itself a part of rendered content.

Usage information

The `<keydef>` element is a convenience element. It is equivalent to a `<topicref>` element with `@processing-role` set to "resource-only".

Attributes defaulted on the `<keydef>` element ensure that key definitions do not appear in tables of contents, do not add extra links, and are not rendered as topics.

Specialization hierarchy

The `<keydef>` element is specialized from `<topicref>`. It is defined in the mapgroup-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with a narrowed definition of `@href`, given below), Common map attributes (369) (with a narrowed definition of `@processing-role`, given below), Topicref-element attributes (369), `@keyref` (388), and the attributes defined below.

@keys (REQUIRED)
: Specifies the required key. Otherwise, the attribute is the same as that described in 10.8.3.8 The keys attribute (389).

@href
: Specifies the referenced resource. In the case of a key definition for variable text, this attribute might be omitted. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications. References to non-DITA content need to use the `@format` attribute to identify the kind of resource.

@processing-role
: Specifies the role that the resource plays in processing. By default, this is set to "resource-only". Otherwise, the definition matches the one in Common map attributes (369).

Example

The following code sample shows several different types of key definitions:

```
<map>
  <title>Possible keys for use in the DITA specification</title>
  <!-- Key definition #1-->
  <keydef keys="dita-tc" scope="external" format="html"
          href="https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dita">
    <topicmeta>
      <linktext>DITA Technical Committee</linktext>
    </topicmeta>
  </keydef>
  <!-- Key definition #2-->
  <keydef keys="addressing" href="dita-addressing.dita"/>
  <!-- Key definition #3-->
  <keydef keys="dita-version">
```

```
    <topicmeta>
      <linktext>2.0</linktext>
    </topicmeta>
  </keydef>
</map>
```

1. The first <keydef> element defines a key that links to a Web page. It contains link text; it also specifies the necessary @scope and @format attributes, so that authors do not need to include them when they reference this key.
2. The second <keydef> element defines a key for a local DITA topic about addressing in DITA; that topic is available to resolve link text.
3. The third <keydef> element defines a text-only key that specifies the current DITA version number.

10.6.7.3 <mapref>

A map reference is a mechanism for referencing a DITA map from a DITA map.

Usage information

The <mapref> element is a convenience element. It is equivalent to a <topicref> element with the @format attribute set to "ditamap".

Processing expectations

The hierarchy of the referenced map is merged into the container map at the position of the reference, and the relationship tables of the child map are added to the parent map.

Comment by Kristen J Eberlein on 06 August 2018

Should we make normative statements about the above points?

Discussed at DITA TC meeting on 02 July 2019.

Consensus: Yes, these need to be normative statements, but this is not the correct location. We need architectural topics that address processing of map hierarchy and relationship tables. This content should be located in the same chapter as the material about subjectScheme maps; it probably should absorb content currently in Processing > Navigation > TOC.

We should plan a review that covers both architectural and element-reference topics about maps.

Comment by Kristen J Eberlein on 05 September 2020

The terms "container map" and "parent map" here are unclear. How are they different?

Specialization hierarchy

The <mapref> element is specialized from <topicref>. It is defined in the map-group module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with narrowed definitions of @href and @format, given below), Common map attributes (369), Topicref-element attributes (369), @keyref (388), and @keys (389).

@format
Specifies the format of the resource. By default, it is set to "ditamap". Otherwise, the attribute is the same as described in Link-relationship attributes (369).

@href
Specifies the referenced resource. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications. References to non-DITA content need to use the `@format` attribute to identify the kind of resource.

Example

The example shows how `<mapref>` elements are used to reference a submap.

Figure 150: DITA map for all element-reference topics (base-elements.ditamap)

The `base-elements.ditamap` document references the `map-group-elements.ditamap)`:

```
<map>
 <title>Base elements</title>
 ...
 <topicref href="containers/domain-elements.dita" >
  ...
  <mapref href="map-group-elements.ditamap"/>
  ...
 </topicref>
 ...
</map>
```

Figure 151: DITA map for map-group elements (map-group-elements.ditamap)

The `map-group-elements.ditamap` document contains references to the element-reference topics for the map group domain. It is constructed as a map in order to enable easy editing of the child topics.

```
<map>
 <title>Map group elements</title>
 <topicref keyref="mapgroup-d" >
  <topicref keyref="anchorref" />
  <topicref keyref="keydef" />
  <topicref keyref="mapref" />
  <topicref keyref="topicgroup" />
  <topicref keyref="topichead" />
 </topicref>
</map>
```

Figure 152: Result after processing base-elements.ditamap

After processing, the `base-elements.ditamap` contains the topic references that originally were located in the submap:

```
<map>
 <title>Base elements</title>
 ...
 <topicref href="containers/domain-elements.dita" >
  ...
  <topicref keyref="mapgroup-d" >
    <topicref keyref="anchorref" />
    <topicref keyref="keydef" />
    <topicref keyref="mapref" />
    <topicref keyref="topicgroup" />
    <topicref keyref="topichead" />
  </topicref>
  ...
 </topicref>
 ...
</map>
```

10.6.7.4 <mapresources>

The <mapresources> element serves as an intuitive location for map authors to place objects with @processing-role set to "resource-only", for example, key definitions and subject scheme maps

Specialization hierarchy

The <mapresources> element is specialized from <topicref>. It is defined in the map group module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Link-relationship attributes (369) (with a narrowed definition of @href, given below), Common map attributes (369) (with the exclusion of @chunk and @collection-type, and a narrowed definition of @processing-role, given below), @keys (389), @keyref (388), and the attributes defined below.

@href
Specifies the referenced resource. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications. References to non-DITA content need to use the @format attribute to identify the kind of resource.

@processing-role
Specifies the role that the resource plays in processing. By default, this is set to "resource-only". Otherwise, the definition matches the one in Common map attributes (369).

Examples

This section provides examples of how the <mapresources> element can be used.

Figure 153: Specifying resource-only objects in an intuitive location in a book map

The following code sample illustrate how the <mapresources> element can group references to key definitions, subject schemes, and other resources in a bookmap:

```
<bookmap>
  <booktitle>
    <mainbooktitle>Test bookmap</mainbooktitle>
  </booktitle>
  <mapresources>
    <mapref href="key-definitions.ditamap"/>
    <mapref href="subject-scheme.ditamap" type="subjectscheme"/>
    <topicref href="cover-page.dita outputclass="cover-page"/>
  </mapresources>
  ...
</bookmap>
```

Note that this example illustrates that <mapresources> can be used to make topics available for resource-only processing. In this scenario, the company uses a processor that uses content contained in the cover-page.dita file to generate a PDF cover page.

Figure 154: Specifying resource-only objects in a map

The following code sample shows a map that contains information for a specific model of a controller. This map is referenced in an omnibus publication that contains information for an entire family of controllers.

```
<map keyscope="model-XNP09">
  <title>Model XNP09</title>
  <mapresources>
    <keydef keys="model-illustration" href="model-XNP09.png" format="png"/>
    <keydef keys="remove-cover" href="remove-cover-XNP09.png" format="png"/>
  </mapresources>
```

```
  <topicref href="model-overview.dita/>
  <topicref href="installing.dita"/>
  <topicref href="uninstalling.dita/>
  ...
</map>
```

10.6.7.5 <topicgroup>

A topic group is a set of topic references that share common attributes and linking relationships.

Usage information

The <topicgroup> element does not affect the navigation hierarchy of the map.

Most <titlealt> elements within the <topicmeta> element inside of a <topicgroup> have no effect on rendered publications, but can be used to hold descriptive information about the grouped <topicref> elements.

Rendering expectations

When a map that contains a <topicgroup> element with a navigation title is used to generate publication output, processors *MUST* ignore the navigation title and *MAY* issue an error message.

Specialization hierarchy

The <topicgroup> element is specialized from <topicref>. It is defined in the map-group domain.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and Common map attributes (369).

The @scope, @format, and @type attributes from Link-relationship attributes (369) are also available.

Example

In the following code sample, the <topicgroup> element specifies common attributes (@audience and @linking) that are inherited by the topic references. The navigation hierarchy is not affected.

```
<topicgroup audience="novice" linking="none">
  <topicmeta>
    <navtitle>Topics used only in "Getting started" material.</navtitle>
  </topicmeta>
  <topicref href="getting-started.dita"/>
  <topicref href="basic-concepts.dita"/>
  <topicref href="cheat-sheet-reference.dita"/>
</topicgroup>
```

10.6.7.6 <topichead>

A topic head is a title-only entry in a DITA map.

Rendering expectations

The content of the <titlealt> element with a @title-role of <navigation>, such as <navtitle>, appears as a heading when the map is rendered as a table of content. In print contexts, it also appears as a heading in the rendered content.

Processing expectations

Processors *SHOULD* generate a warning if a navigation title is not specified on a <topichead> element.

Specialization hierarchy

The <topichead> element is specialized from <topicref>. It is defined in the map-group module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366), Common map attributes (369), and @copy-to from Topicref-element attributes (369). This element also uses the @scope, @format, and @type attributes from Link-relationship attributes (369).

Example

In the following example, the <topichead> elements provide titles ("Computers" and "Books") for two groups of topics.

```
<map>
  <topichead>
    <topicmeta>
      <navtitle>Computers</navtitle>
    </topicmeta>
    <topicref href="eniac.dita"/>
    <topicref href="system360.dita"/>
    <topicref href="pdp8.dita"/>
  </topichead>
  <topichead>
    <topicmeta>
      <navtitle>Books</navtitle>
    </topicmeta>
    <topicref href="hardback.dita"/>
    <topicref href="paperback.dita"/>
  </topichead>
</map>
```

10.6.8 Utilities domain elements

The utilities domain elements represent common features of a language that might not necessarily be semantic, such as image maps.

10.6.8.1 <area>

The <area> element describes a linkable area within an <imagemap>. It allows the author to specify a shape within the image, the coordinates of that shape, and a link target for the area.

Specialization hierarchy

The <area> element is specialized from . It is defined in the utilities-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366).

Example

See 10.6.8.3 imagemap (350).

10.6.8.2 <coords>

The <coords> element specifies the coordinates of a linkable region in an <imagemap>.

Usage information

This element contains text data representing coordinates for a region in an image map. Pixels are the recommended units for describing coordinates. The syntax of the coordinate data depends on the shape described by the coordinates, and is based on the image map definition in HTML. It uses the following data for the appropriate shapes:

Shape	Data format
rect	left-x, top-y, right-x, bottom-y
circle	center-x, center-y, radius
poly	x1, y1, x2, y2, …, xN, yN. To close the polygon, ensure that the first x and y coordinate pair and the last are the same.

Specialization hierarchy

The <coords> element is specialized from <ph>. It is defined in the utilities-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (with a narrowed definition of @translate, given below), and @keyref (388).

@translate

Specifies whether the content of the element is translatable. The default value is "no". Setting @translate to "yes" overrides the default value. The DITA specification contains a list of each OASIS DITA element and its common processing default for the translate value; because this element uses an actual default, it is always treated as translate="no" unless overridden. Available values are:

no

The content of this element is not translatable.

yes

The content of this element is translatable.

-dita-use-conref-target

See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

Example

See 10.6.8.3 imagemap (350).

10.6.8.3 <imagemap>

The <imagemap> element supports the basic functionality of the HTML "client-side" image map markup. <imagemap> allows you to designate a linkable area or region over an image, allowing a link in that region to display another topic.

Usage information

An HTML client-side image map binds an image to the navigation structure (the "map") by means of an ID association from the map to the image. In contrast, the DITA version of <imagemap> markup simply

includes the target image as the first required element in the markup, followed by a sequence of <area> elements that represent the links associated with the contained image.

The <xref> content within <area> contains the intended alternative text or hover text for that image map area. Normal text retrieval for <xref> elements can be used to populate that alternative text.

Rendering expectations

An <imagemap> structure can be rendered as a standard HTML image map or as an alternative form of navigation (such as table-based image maps). When rendered in PDF, the minimal form would be to represent the image; advanced PDF output processors could provide equivalent region-oriented hyperlinks.

Specialization hierarchy

The <imagemap> element is specialized from . It is defined in the utilities-domain module.

Attributes

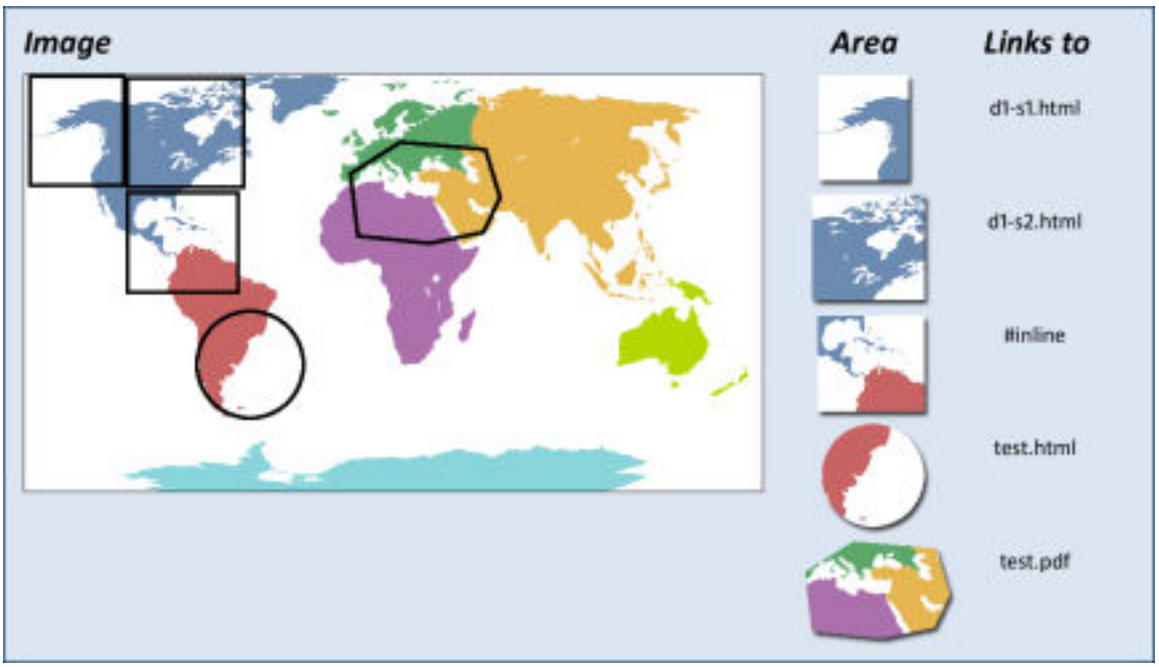
The following attributes are available on this element: 10.8.1 Universal attribute group (366), Display attributes (369), and spectitle (specialization attributes) (376).

Example

The following code sample shows a simple <imagemap> with five areas, using three different shapes:

```
<imagemap>
 <image href="imagemapworld.jpg">
   <alt>Map of the world showing 5 areas</alt>
 </image>
 <area><shape>rect</shape><coords>2,0,53,59</coords>
  <xref href="d1-s1.dita">Section 1 alternative text</xref>
 </area>
 <area><shape>rect</shape><coords>54,1,117,60</coords>
  <xref href="d1-s2.dita"><!-- Pull title from d1-s2.dita --></xref>
 </area>
 <area><shape>rect</shape><coords>54,62,114,116</coords>
  <xref href="#inline" type="topic">Alternative text for this rectangle</xref>
 </area>
 <area><shape>circle</shape><coords>120,154,29</coords>
  <xref format="html" href="test.html">Link to a test html file</xref>
 </area>
 <area><shape>poly</shape>
  <coords>246,39,200,35,173,52,177,86,215,90,245,84,254,65</coords>
  <xref format="pdf" href="test.pdf">Link to a test PDF file</xref>
 </area>
</imagemap>
```

The areas defined correspond to this graphic image with the areas visible:



Comment by Robert
I think we should update this to use the HTML5 model, rather than HTML4.1: https://www.w3.org/TR/2011/WD-html5-20110525/the-map-element.html#the-area-element

The "latest" version is much newer, but is also a living standard, and creating a dependency would seem to turn this one element from DITA into its own living standard: https://html.spec.whatwg.org/multipage/image-maps.html#image-map-processing-model:the-area-element-10

The values for use in the `<shape>` and `<coords>` elements follow the guidelines defined for image maps in HTML 4.1, Client-side image maps: the MAP and AREA elements

10.6.8.4 <shape>

The `<shape>` element defines the shape of a linkable area in an `<imagemap>`.

Usage information

The `<shape>` element supports these values:

rect
Define a rectangular region. If you leave the `<shape>` element blank, a rectangular shape is assumed.

circle
Define a circular region.

poly
Define a polygonal region.

default
Indicates the entire diagram.

Specialization hierarchy

The `<shape>` element is specialized from `<keyword>`. It is defined in the utilities-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (with a narrowed definition of `@translate`, given below), and `@keyref` (388).

@translate
Specifies whether the content of the element is translatable. The default value is "no". Setting `@translate` to "yes" overrides the default value. The DITA specification contains a list of each OASIS DITA element and its common processing default for the translate value; because this element uses an actual default, it is always treated as `translate="no"` unless overridden. Available values are:

no
The content of this element is not translatable.

yes
The content of this element is translatable.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

Example

See 10.6.8.3 imagemap (350).

10.6.8.5 <sort-as>

For elements that are sorted, the `<sort-as>` element provides text that is combined with the base sort phrase to construct the effective sort phrase.

Usage information

Sort text can be specified in the content of the `<sort-as>` element or in the `@value` attribute on the `<sort-as>` element. The `<sort-as>` element also is useful for elements where the base sort phrase is inadequate or non-existent, for example, a glossary or index entry for a Japanese Kanji phrase.

If a `<keyword>` element is used within `<sort-as>`, the `@keyref` attribute can be used to set the sort phrase. If a `<keyword>` uses `@keyref` and would otherwise also act as a navigation link, the link aspect of the `@keyref` attribute is ignored.

Some elements in the base DITA vocabulary are natural candidates for sorting, including topics, definition list entries, index entries, and rows in tables and simple tables. Authors are likely to include `<sort-as>` elements in the following locations:

- For topics, the `<sort-as>` element can be included directly in `<title>` or `<titlealt>` when the different forms of title need different effective sort phrases. If the effective sort phrase is common to all the titles for a topic, the `<sort-as>` element can be included as a direct child of the `<prolog>` element in the topic.
- For glossary entry topics, the `<sort-as>` element can be included directly in `<glossterm>` or as a direct child of the `<prolog>` element.
- For topic references, the `<sort-as>` element can be included directly in the `<titlealt>` element with a `@title-role` of `navigation`, such as `<navtitle>`, within `<topicmeta>` or as a child of `<topicmeta>`.
- For definition list items, the `<sort-as>` element can be included in the `<dt>` element.
- For index entries, the `<sort-as>` can be included as a child of `<indexterm>`. In a multilevel `<indexterm>` element, the `<sort-as>` element only affects the level in which it occurs.

Processing expectations

As a specialization of <data>, the <sort-as> element is allowed in any context where <data> is allowed. However, the presence of <sort-as> within an element does not, by itself, indicate that the containing element should be sorted. Processors can choose to sort any DITA elements for any reason. Likewise, processors are not required to sort any elements. See 7.7 Sorting (139) for more information on sorting.

Processors *SHOULD* expect to encounter <sort-as> elements in the above locations. Processors that sort *SHOULD* use the following precedence rules:

- A <sort-as> element that is specified in a title takes precedence over a <sort-as> element that is specified as a child of the topic prolog.
- Except for instances in the topic prolog, processors only apply <sort-as> elements that are either a direct child of the element to be sorted or a direct child of the title- or label-defining element of the element to be sorted.
- When an element contains multiple, direct-child, <sort-as> elements, the first direct-child <sort-as> element in document order takes precedence.
- It is an error if there is more than one <sort-as> child for a given <indexterm>. An implementation encountering more than one <sort-as> in this case might give an error message.
- Sort phrases are determined after filtering and content reference resolution occur.

When a <sort-as> element is specified, processors that sort the containing element *MUST* construct the effective sort phrase by prepending the content of the <sort-as> element to the base sort phrase. This ensures that two items with the same <sort-as> element but different base sort phrases will sort in the appropriate order.

For example, if a processor uses the content of the <title> element as the base sort phrase, and the title of a topic is "24 Hour Support Hotline" and the value of the <sort-as> element is "twenty-four hour", then the effective sort phrase would be "twenty-four hour24 Hour Support Hotline".

Specialization hierarchy

The <sort-as> element is specialized from <data>. It is defined in the utilities-domain module.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) and the attributes defined below.

@name
Specifies the metadata item that the element represents. The default value is "sort-as". Specializations of <sort-as> can set the default value of the @name attribute to reflect the tag name of the specialized element.

@value
Specifies the value of the metadata item. When the <sort-as> element has content and the @value attribute is specified, the @value attribute takes precedence. If the @value attribute is not specified and the <sort-as> element does not contain content, then the <sort-as> element has no effect.

Example

The following examples illustrate how a glossary entry for the Chinese ideographic character for "big" might specify an effective sort phrase of "dada" (the Pin-Yin transliteration for Mandarin):

Figure 155: The <sort-as> element located within <glossterm>

```
<glossentry id="gloss-dada">
  <glossterm><sort-as value="dada"/>&#x5927;&#x5927;</glossterm>
    <glossdef>Literally "big big".</glossdef>
</glossentry>
```

Figure 156: The <sort-as> element within <prolog>

```
<glossentry id="gloss-dada">
  <glossterm>&#x5927;&#x5927;</glossterm>
  <prolog>
    <sort-as>dada</sort-as>
  </prolog>
    <glossdef>Literally "big big".</glossdef>
</glossentry>
```

Related concepts

Sorting (139)

Processors can be configured to sort elements. Typical processing includes sorting glossary entries, index entries, lists of parameters or reference entries in custom navigation structures, and tables based on the contents of cells in specific columns or rows.

10.7 Other elements

10.7.1 Legacy conversion elements

Conversion elements exist primarily to aid in the conversion of content to DITA.

10.7.1.1 <required-cleanup>

Required cleanup sections are placeholders for migrated elements that cannot be appropriately tagged without manual intervention, or for content that must be cleaned up before publishing.

Usage information

As the element name implies, the intent for authors is to clean up the contained material and eventually remove the `<required-cleanup>` element.

Rendering expectations

Comment by Robert on 20200831
The following statement was written as if RFC-level "must" was used, but was not marked with RFC. I've added the RFC notation; if this is not meant to be "MUST" then we need to find alternate wording.

We should probably also use a clearer definition of "Processors" in this context, as an editor is a processor that should not be forced to strip the content from displaying.

Update 2020-10-19: moving the "must strip" language into a rendering section. This is about not *rendering* the element.

Processors *MUST* strip this element from output by default. The content of `<required-cleanup>` is not considered to be publishable data.

Processing expectations

Processor options might be provided to allow a draft view of migrated content in context.

Attributes

The following attributes are available on this element: 10.8.1 Universal attribute group (366) (with a modified definition of `@translate`, given below), and the attributes defined below.

@remap
: Specifies information about the origins of the content of the `<required-cleanup>` element. This provides authors with context for determining how migrated content was originally encoded.

@translate
: Specifies whether the content of the element should be translated or not. The default value for this element is "no"; setting to "yes" will override the default. The -dita-use-conref-target (385) value is also available. The DITA architectural specification contains a list of each OASIS DITA element and its common processing default for the translate value; because this element uses an actual default, it will always be treated as `translate="no"` unless overridden as described.

Example

In the following example, an HTML document that used the `<center>` element was migrated to DITA. Because DITA has no clear equivalent element, the content is stored in `<required-cleanup>` until it can be marked up appropriately.

```
<section>
  <title>Using the display</title>
  <required-cleanup remap="center">If you cannot read
your display, see "Adjusting the language setting"
before you continue.</required-cleanup>
</section>
```

10.7.2 DITAVAL elements

A conditional processing profile (DITAVAL file) is used to identify which values are to be used for conditional processing during a particular output, build, or some other purpose. The profile should have an extension of `.ditaval`.

The DITAVAL format has several elements: `<val>`, the root element, can contain a `<style-conflict>` element followed by `<prop>` or `<revprop>` elements; the `<prop>` and `<revprop>` elements can contain `<startflag>` and `<endflag>` elements; and the `<startflag>` and `<endflag>` elements can contain `<alt-text>` elements.

Notes on DITAVAL messages

Conditional processing code should provide a report of any attribute values encountered in content that do not have an action associated with them.

Note on DITAVAL flagging of images

If an image in DITA content becomes flagged using a background color, the color should be represented as a thick border. If a foreground color is expressed, it should be represented as a thin border.

Related concepts

Filtering and flagging attributes (32)
Conditional-processing attributes are available on most elements.

Conditional processing (profiling) (117)
Conditional processing, also known as profiling, is the filtering or flagging of information based on processing-time criteria.

Conditional processing values and groups (117)
Conditional processing attributes classify elements with metadata. The metadata is specified using space-delimited string values or grouped values.

10.7.2.1 <alt-text>

The `<alt-text>` element in a DITAVAL document provides alternate text for an image used for flagging.

Rendering expectations

If alternate text is specified but the containing element does not reference an image, applications can render the text itself as a way to flag content.

When no alternate text is specified for a revision flag, the default alternate text for `<revprop>` start of change is typically a localized translation of "Start of change", and the default alternate text for `<revprop>` end of change is typically a localized translation of "End of change".

Attributes

This element does not define any attributes.

Example

See 10.7.2.7 val (363).

10.7.2.2 <endflag>

The `<endflag>` element in a DITAVAL document provides information that identifies the end of flagged content.

Usage information

The `<endflag>` element defines a flag to be used at the end of content identified by "flag" conditions in a DITAVAL document:

- If an image is specified, the specified image is a flag to identify the end of the content, with the `<alt-text>` contents as alternative text for that image.
- If `<alt-text>` is specified but `<endflag>` does not reference an image, that text can be used to flag the content instead of an image.
- If no image and no `<alt-text>` are specified, then this element has no defined purpose.

Attributes

The following attribute is available on this element:

@imageref
Provides a URI reference to the image file, using the same syntax as the `@href` attribute. See 10.8.3.6 The href attribute (387) for information on supported values and processing implications.

Example

See 10.7.2.7 val (363).

10.7.2.3 <prop>

The `<prop>` element in a DITAVAL document identifies an attribute, and usually values in the attribute, to take an action on. The identified attribute is a conditional-processing attribute (either `@props` or a specialization of `@props`, such as `@audience`, `@deliveryTarget`, `@platform`, `@product`, or `@otherprops`).

Usage information

A `<prop>` element performs the following functions:

- A `<prop>` element with no `@att` attribute specified sets a default action for every `<prop>` element.
- A `<prop>` element with an `@att` attribute but no `@val` attribute sets a default action for that specific attribute or attribute group.
- A `<prop>` element with an `@att` attribute and a `@val` attribute sets an action for that value within that attribute or attribute group.

Rendering expectations

For the `@color` and `@backcolor` attributes on `<rev>` and `<revprop>`, processors *SHOULD* support the following:

- The color names listed under the heading "<color>" in http://www.w3.org/TR/2006/REC-xsl11-20061205/#datatype
- The associated 6 digit hex code form (#rrggbb, case insensitive).

For the `@style` attribute on `<rev>` and `<revprop>`, the following tokens *SHOULD* be processed by all DITAVAL processors:

- underline
- double-underline
- italics
- overline
- bold

In addition, processors *MAY* support other proprietary tokens for the `@style` attribute. Such tokens *SHOULD* have a processor-specific prefix to identify them as proprietary. If a processor encounters an unsupported style token, it *MAY* issue a warning, and *MAY* render content flagged with such a style token using some default formatting.

Processing expectations

Comment by rodaande
This seems a weird way to write up error conditions, feels like each list item should explicitly state "It is an error to XYZ"?

Processors *MAY* provide an error or warning message for the following error conditions in a DITAVAL document:

- More than one `<prop>` element with no `@att` attribute
- More than one `<prop>` element with the same `@att` attribute and no value
- More than one `<prop>` element with the same `@att` attribute and same `@value`

Attributes

The following attributes are available on this element:

@att
Specifies the attribute to be acted upon. If using a literal attribute name, it is `@props` or a specialization of `@props` (such as `@audience`, `@deliveryTarget`, `@platform`, `@product`, or `@otherprops`). Otherwise, the value is the name of a group within one of those attributes, with the group name specified using the generalized attribute syntax. If the `@att` attribute is absent, then the `<prop>` element declares a default behavior for any conditional processing attribute.

@val
Specifies the attribute value to be acted upon. If the `@val` attribute is absent, then the `<prop>` element declares a default behavior for any value in the specified attribute.

@action (REQUIRED)
Specifies the action to be taken. Allowable values are:

include
Include the content in output. This is the default behavior unless otherwise set.

exclude
Exclude the content from output (if all values in the particular attribute are excluded).

passthrough
Include the content in output, and preserve the attribute value as part of the output stream for further processing by a runtime engine. For example, this could be used to enable runtime filtering based on individual user settings. The value should be preserved in whatever syntax is required by the target runtime. Values that are not explicitly passed through should be removed from the output stream, even though the content is still included.

flag
Include and flag the content on output (if the content has not been excluded).

@outputclass
If the `@action` attribute is set to "flag", treat the flagged element as if the full `@outputclass` value in the DITAVAL was specified on that element's `@outputclass` attribute. If two or more DITAVAL properties apply `@outputclass` flags to the same element, treat the flagged element as if each value was specified on that element's `@outputclass` attribute; in that case, the order of those DITAVAL-based tokens is undefined. If the flagged element already specifies `@outputclass`, treat the flagged element as if all DITAVAL-based `@outputclass` values come first in the attribute.

@color
If the `@action` attribute is set to "flag", specifies the color to use to flag text. Colors can be entered by name or code. If the `@action` attribute is not set to "flag", this attribute is ignored.

@backcolor
If the `@action` attribute is set to "flag", specifies the color to use as background for flagged text. Colors can be entered by name or code. If the `@action` attribute is not set to "flag", this attribute is ignored.

@style

If the `@action` attribute is set to "flag", specifies the text styles to use for flagged text. This attribute can contain multiple space-delimited tokens. If the `@action` attribute is not set to "flag", this attribute is ignored.

Example

See the example in the <val> (363) description.

10.7.2.4 <revprop>

The `<revprop>` element in a DITAVAL document identifies a value in the `@rev` attribute that should be flagged in some manner. Unlike the conditional processing attributes, which can be used for both filtering and flagging, the `@rev` attribute only can be used for flagging.

Usage information

The `@rev` attribute identifies when a particular section of a document was added or modified. The attribute is not considered a filtering attribute because this is not sufficient to be used for full version control, such as single-sourcing multiple product variants based on version level – it only represents one aspect of the revision level.

Rendering expectations

When no alternate text is specified for a revision flag, the default alternate text for `<revprop>` start of change is typically a localized translation of "Start of change", and the default alternate text for `<revprop>` end of change is typically a localized translation of "End of change".

For the `@color` and `@backcolor` attributes on `<rev>` and `<revprop>`, processors *SHOULD* support the following:

- The color names listed under the heading "<color>" in http://www.w3.org/TR/2006/REC-xsl11-20061205/#datatype
- The associated 6 digit hex code form (#rrggbb, case insensitive).

For the `@style` attribute on `<rev>` and `<revprop>`, the following tokens *SHOULD* be processed by all DITAVAL processors:

- underline
- double-underline
- italics
- overline
- bold

In addition, processors *MAY* support other proprietary tokens for the `@style` attribute. Such tokens *SHOULD* have a processor-specific prefix to identify them as proprietary. If a processor encounters an unsupported style token, it *MAY* issue a warning, and *MAY* render content flagged with such a style token using some default formatting.

Processing expectations

It is an error to include more than one `<revprop>` element with the same `@val` attribute setting. Recovery from this error is implementation dependent; in such cases processors *MAY* provide an error or warning message.

Attributes

The following attributes are available on this element:

@val
Specifies the revision value to be acted upon. If the `@val` attribute is absent, then the `<revprop>` element declares a default behavior for any value in the `@rev` attribute.

@action (REQUIRED)
Specifies the action to be taken. Allowable values are:

include
Include the content in output without flags. This is the default behavior unless otherwise set.

passthrough
Include the content in output, and preserve the attribute value as part of the output stream for further processing by a runtime engine. For example, this could be used to enable runtime highlighting based on individual user settings. The value should be preserved in whatever syntax is required by the target runtime. Values that are not explicitly passed through should be removed from the output stream, even though the content is still included.

flag
Include and flag the content when rendered (if the content has not been excluded).

@changebar
When flag has been set, specify a changebar color, style, or character, according to the changebar support of the target output format. If the `@action` attribute is not set to "flag", this attribute is ignored.

@outputclass
If the `@action` attribute is set to "flag", treat the flagged element as if the full `@outputclass` value in the DITAVAL was specified on that element's `@outputclass` attribute. If two or more DITAVAL properties apply `@outputclass` flags to the same element, treat the flagged element as if each value was specified on that element's `@outputclass` attribute; in that case, the order of those DITAVAL-based tokens is undefined. If the flagged element already specifies `@outputclass`, treat the flagged element as if all DITAVAL-based `@outputclass` values come first in the attribute.

@color
If the `@action` attribute is set to "flag", specifies the color to use to flag text. Colors can be entered by name or code. If the `@action` attribute is not set to "flag", this attribute is ignored.

@backcolor
If the `@action` attribute is set to "flag", specifies the color to use as background for flagged text. Colors can be entered by name or code. If the `@action` attribute is not set to "flag", this attribute is ignored.

@style
If the `@action` attribute is set to "flag", specifies the text styles to use for flagged text. This attribute can contain multiple space-delimited tokens. If the `@action` attribute is not set to "flag", this attribute is ignored.

Example

See 10.7.2.7 val (363).

Related concepts

Flagging (120)

Flagging is the application of text, images, or styling during rendering. This can highlight the fact that content applies to a specific audience or operating system, for example; it also can draw a reader's attention to content that has been marked as being revised.

10.7.2.5 <startflag>

The `<startflag>` element in a DITAVAL document provides information that identifies the beginning of flagged content.

Usage information

The `<start>` element defines a flag to be used at the beginning of content identified by "flag" conditions in a DITAVAL document:

- If an image is specified, the specified image is a flag to identify the beginning of the content, with the `<alt-text>` contents as alternative text for that image.
- If `<alt-text>` is specified but `<startflag>` does not reference an image, that text can be used to flag the content instead of an image.
- If no image and no `<alt-text>` are specified, then this element has no defined purpose.

Attributes

The following attribute is available on this element:

@imageref
: Provides a URI reference to the image file, using the same syntax as the `@href` attribute. See 10.8.3.6 The href attribute (387) for information on supported values and processing implications.

Example

See 10.7.2.7 val (363).

10.7.2.6 <style-conflict>

The `<style-conflict>` element in a DITAVAL document declares the behavior to be used when one or more flagging methods collide on a single content element.

Usage information

In case of conflicts between flagging methods at different levels (for example, a section is flagged green and a paragraph within the section is flagged red), the most deeply nested flagging method applies.

Rendering expectations

In case of conflicts between flagging methods on the same element (for example, a single element is being flagged with both green and red color), it is recommended that the conflicts be resolved as follows:

Flagging method	Conflict behavior
<startflag>	Add all flags that apply.
<endflag>	Add all flags that apply.
color	Follow the `<style-conflict>` element's `@foreground-conflict-color` setting, or use an output-appropriate default color if no conflict color is set.

Flagging method	Conflict behavior
backcolor	Follow the `<style-conflict>` element's `@background-conflict-color` setting, or use an output-appropriate default color if no conflict color is set.
style	Add all font styles that apply. If two different kinds of underline are used, default to the heaviest (double underline) and use the `@foreground-conflict-color`.
changebar	Add all change bars that apply.

Attributes

The following attributes are available on this element:

@foreground-conflict-color
Specifies the color to be used when more than one flagging color applies to a single content element.

@background-conflict-color
Specifies the color to be used when more than one flagging background color applies to a single content element.

Example

See 10.7.2.7 val (363).

10.7.2.7 <val>

The `<val>` element is the root element of a DITAVAL document.

Usage information

For information about processing DITAVAL files, including how to filter or flag elements with multiple property attributes or multiple properties within a single attribute, see 7.4 Conditional processing (profiling) (117).

Attributes

This element does not define any attributes.

Example

Figure 157: Sample DITAVAL file

```
<val>
   <style-conflict background-conflict-color="red"/>
   <prop action="include" att="audience" val="everybody"/>
   <prop action="flag" att="product" val="YourProd" backcolor="purple"/>
   <prop action="flag" att="product" backcolor="blue"
         color="yellow" style="underline" val="MyProd">
      <startflag imageref="startflag.jpg">
        <alt-text>This is the start of my product info</alt-text>
      </startflag>
      <endflag imageref="endflag.jpg">
        <alt-text>This is the end of my product info</alt-text>
      </endflag>
   </prop>
   <revprop action="flag" val="1.2"/>
</val>
```

This sample DITAVAL file performs the following actions:

- Elements with `audience="everybody"` are included without change.
- Elements with `product="YourProd"` get a background color of purple.
- Elements with `product="MyProd"` get the following actions:
  - The image `startflag.jpg` is placed at the start of the element.
  - The image `endflag.jpg` is placed at the end of the element.
  - The element gets a background color of blue.
  - The text in the element appears in yellow; the text is underlined.
- Elements marked with are flagged with the default revision flags, which are implementation dependent.
- When there are conflicts, for example, if an element is marked with `product="MyProd YourProd"`, it will be flagged with a background color of red.

Figure 158: DITAVAL file that overrides the default "include" action

```
<val>
   <prop action="exclude"/>
   <prop action="include" att="audience" val="everybody"/>
   <prop action="include" att="audience" val="novice"/>
   <prop action="include" att="product" val="productA"/>
   <prop action="include" att="product" val="productB"/>
</val>
```

This simple DITAVAL file performs the following actions:

- The first `<prop>` element does not specify an attribute, which sets a default action of "exclude" for every prop value. This means that, by default, any property value not otherwise defined in this file evaluates to "exclude". Note that this same behavior can be limited to a single attribute; the following `<prop>` element sets a default action of "exclude" for all properties specified on the `@platform` attribute: `<prop action="exclude" att="platform"/>`
- The second and third `<prop>` elements set an action of "include" for two values on the `@audience` attribute. All other values on the `@audience` attribute still evaluate to "exclude".
- The fourth and fifth `<prop>` elements set an action of "include" for two values on the `@product` attribute. All other values on the `@product` attribute still evaluate to "exclude".

Figure 159: DITAVAL with conditions for groups

```
<val>
   <prop action="exclude" att="product" val="appserver"/>
   <prop action="include" att="product" val="mySERVER"/>
   <prop action="include" att="database" val="dbFIRST"/>
   <prop action="include" att="database" val="dbSECOND"/>
   <prop action="exclude" att="database" val="newDB"/>
</val>
```

Assume that "database" and "appServer" are used as group names within the `@product` attribute. In that case, the sample DITAVAL above performs the following actions:

- The first `<prop>` element excludes the value "appServer" when used within the `@product` attribute. It also sets a default of "exclude" for values within any appServer() group inside of the `@product` attribute.
- The second `<prop>` element sets "mySERVER" to include; this applies whether "mySERVER" appears alone in the `@product` attribute (`product="mySERVER"`) or inside of any group (`product="appServer(mySERVER)"` or `product="otherGroup(mySERVER)"`).
- The third and fourth `<prop>` elements set the database values "dbFIRST" and "dbSECOND" to include. If those values appear inside of a "database" group, they are explicitly set to "include". If they appear elsewhere in a conditional attribute (such as `product="dbFIRST"` or `platform="dbSECOND"`), this rule does not apply.

- The final `<prop>` element sets the database value "newDB" to exclude. If that value appears inside of a database group, it is explicitly set to "exclude". If it appears in any other group or attribute, this rule does not apply.

Remember that with groups, if all values inside of a single group evaluate to "exclude", that is equivalent to an entire attribute evaluating to "exclude", which results in the removal of the content. Using the above sample DITAVAL:

- `<p product="appServer">` is filtered out, because this value is excluded.
- `<p product="appServer(A B)">` is filtered out, because there is no explicit rule for A or B, and values in the "appServer" group inside of `@product` default to exclude.
- `<p product="appServer(A B mySERVER)">` is included, because `product="mySERVER"` evaluates to "include", which means the entire group evaluates to "include".
- `<p product="newDB">` is included, because no rule in the DITAVAL applies, so the "newDB" token defaults to "include".
- `<p product="database(newDB)">` is filtered out, because the token "newDB" is excluded when found in the database group.
- `<p product="database(dbFIRST dbSECOND newDB)">` is included, because both "dbFIRST" and "dbSECOND" are included, so the group evaluates to include.
- `<p product="database(newDB) appserver(mySERVER)">` is filtered out, because the token "newDB" is excluded when found in the database group. The entire "database" group on this paragraph evaluates to "exclude", so the element is excluded, regardless of how the "appServer" group evaluates.

Note If two groups with the same name exist on different attributes, each group will evaluate the same way. For example, rules for the database group in this sample would evaluate the same whether the group is used within `@product` or `@platform`. See 7.4 Conditional processing (profiling) (117) for suggestions on how to handle similar groups on different attributes.

Related concepts

Filtering (118)
At processing time, a conditional processing profile can be used to specify profiling attribute values that determine whether an element with those values is included or excluded.

Flagging (120)
Flagging is the application of text, images, or styling during rendering. This can highlight the fact that content applies to a specific audience or operating system, for example; it also can draw a reader's attention to content that has been marked as being revised.

Examples of conditional processing (121)

This section provides examples that illustrate the ways that conditional processing attributes can be set and used.

10.8 Attributes

This section collects commonly used attributes, with common definitions. If an element uses a different definition, or narrows the scope of, an otherwise common attribute, it will be called out in the topic that defines the element.

10.8.1 Universal attribute group

The universal attribute group defines a set of common attributes that are available on almost every DITA element. The universal attribute group includes all attributes from the ID, metadata, and localization attribute groups, plus the `@class` and `@outputclass` attributes.

Common attribute groups

The following groups are referenced in this specification, and are also used in grammar files when defining attributes for elements.

Universal attributes
: Includes `@class` and `@outputclass`, along with every attribute in the ID, Localization, and Metadata attribute groups.

ID attributes
: Includes attributes that enable the naming and referencing of elements in topics and maps: `@conaction`, `@conkeyref`, `@conref`, `@conrefend`, and `@id`.

Localization attributes
: Includes attributes that are related to translation and localization: `@dir`, `@translate`, and `@xml:lang`.

Metadata attributes
: Includes common metadata attributes, two of which are available for specialization: `@base`, `@importance`, `@props`, `@rev`, and `@status`.
The base DITA vocabulary from OASIS includes several predefined specializations of `@props`. These attributes are defined as independent attribute extension domains; they are integrated by default into all OASIS-provided document-type shells, and can be easily removed from custom document-type shells: `@audience`, `@deliveryTarget`, `@platform`, `@product`, and `@otherprops`.

Universal attribute definitions

The universal attributes for OASIS DITA elements are defined below. Specialized attributes, which are part of the OASIS distribution but are only available when explicitly included in a shell, are noted in the list.

@audience *(specialized attribute)*
: Indicates the intended audience for the element. If no value is specified, but the attribute is specified on an ancestor within a map or within the related-links section, the value will cascade from the closest ancestor.

@base
: A generic attribute that has no specific purpose. It is intended to act as a base for specialized attributes that have a simple value syntax like the conditional processing attributes (one or more alphanumeric values separated by whitespace), but is not itself a filtering or flagging attribute.

The `@base` attribute takes a space-delimited set of values. However, when acting as a container for generalized attributes, the attribute values will be more complex; see 8.4.4 Attribute generalization (154) for more details.

@class *(not for use by authors)*
This attribute is not for use by authors. If an editor displays `@class` *attribute values, do not edit them.* The `@class` attribute supports specialization. Its predefined values allow DITA tools to work correctly with ranges of related content. In a generalized DITA document the `@class` attribute value in the generalized instance might differ from the default value for the `@class` attribute for the element as given in the DTD or schema. See 8.3.6 class attribute rules and syntax (147) for more information. This attribute is specified on every element except for the `<dita>` container element. It is always specified with a default value, which varies for each element.

@conaction
This attribute enables users to push content into a new location. Allowable values are "mark", "pushafter", "pushbefore", "pushreplace", and "-dita-use-conref-target". See 10.8.3.1 The conaction attribute (377) for examples and details about the syntax.

@conkeyref
Allows the conref feature to operate using a key instead of a URI. See 10.8.3.3 The conkeyref attribute (384) for more details about the syntax and behaviors.

@conref
This attribute is used to reference an ID on content that can be reused. See 10.8.3.4 The conref attribute (384) for examples and details about the syntax.

@conrefend
The `@conrefend` attribute is used when reusing a range of elements through `@conref`. The syntax is the same as for the `@conref` attribute; see 10.8.3.2 The conrefend attribute (380) for examples.

@deliveryTarget *(specialized attribute)*
The intended delivery target of the content, for example, "html", "pdf", or "epub". If no value is specified, but the attribute is specified on an ancestor within a map or within the related-links section, the value will cascade from the closest ancestor.

@dir
Specifies the directionality of text: left-to-right ("ltr", the processing default) or right-to-left ("rtl"). The value "lro" indicates an override of normal bidirectional text presentation, forcing the element into left-to-right mode; "rlo" overrides normal rules to force right-to-left presentation. Allowable values are "ltr", "rtl", "lro", "rlo", and "-dita-use-conref-target". See 7.6.2 The dir attribute (138) for more information.

@id
An anchor point. This ID is the target for references by `@href` and `@conref` attributes and for external applications that refer to DITA content. This attribute is defined with the XML data type NMTOKEN, except where noted for specific elements within the language reference. See 6.1 ID attribute (73) for more details.

@importance
A range of values that describe an importance or priority attributed to an element. For example, in steps of a task, the attribute indicates whether a step is optional or required. This attribute is not used for DITAVAL-based filtering or flagging; applications might use the importance value to highlight elements. Allowable values are: "obsolete", "deprecated", "optional", "default", "low", "normal", "high", "recommended", "required", "urgent", and "-dita-use-conref-target".

@otherprops *(specialized attribute)*
This attribute can be used for any other properties that might be needed to describe an audience, or to provide selection criteria for the element. Alternatively, the `@props` attribute can be specialized to provide a new metadata attribute instead of using the general `@otherprops` attribute. If no value is

specified, but the attribute is specified on an ancestor within a map or within the related-links section, the value will cascade from the closest ancestor.

@outputclass
Names a role that the element is playing. The role must be consistent with the basic semantic and expectations for the element. In particular, the `@outputclass` attribute can be used for styling during output processing; HTML output will typically preserve `@outputclass` for CSS processing.

@platform *(specialized attribute)*
Indicates operating system and hardware. If no value is specified, but the attribute is specified on an ancestor within a map or within the related-links section, the value will cascade from the closest ancestor.

@product *(specialized attribute)*
Contains the name of the product to which the element applies. If no value is specified, but the attribute is specified on an ancestor within a map or within the related-links section, the value will cascade from the closest ancestor.

@props
Root attribute from which new metadata attributes can be specialized. This is a property attribute which supports conditional processing for filtering or flagging. If no value is specified, but the attribute is specified on an ancestor within a map or within the related-links section, the value will cascade from the closest ancestor.
The `@props` attribute takes a space-delimited set of values. However, when acting as a container for generalized attributes, the attribute values will be more complex; see 8.4.4 Attribute generalization (154) for more details.

@rev
Indicates a revision level of an element that identifies when the element was added or modified. It can be used to flag outputs when it matches a run-time parameter; it cannot be used for filtering. It is not sufficient to be used for version control. If no value is specified, but the attribute is specified on an ancestor within a map or within the related-links section, the value will cascade from the closest ancestor.

@status
The modification status of the current element. Allowable values are: "new", "changed", "deleted", "unchanged", and "-dita-use-conref-target".

@translate
Indicates whether the content of the element should be translated or not. Allowable values are "yes", "no", and "-dita-use-conref-target". See C.6 Element-by-element recommendations for translators (402) for suggested processing defaults for each element.

@xml:lang
Specifies the language of the element content. The `@xml:lang` attribute and its values are described in the XML Recommendation at http://www.w3.org/TR/REC-xml/#sec-lang-tag. Allowable values are language tokens or the null string.

Related concepts

Specialization (144)
The specialization feature of DITA allows for the creation of new element types and attributes that are explicitly and formally derived from existing types. This facilitates interchange of conforming DITA content and ensures a minimum level of common processing for all DITA content. It also allows specialization-aware processors to add specialization-specific processing to existing base processing.

DTD: Coding requirements for attribute domain modules (181)
The vocabulary modules that define attribute domains have additional coding requirements. The module must include a parameter entity for the new attribute, which can be referenced in document-type shells,

as well as a text entity that specifies the contribution to the `@specializations` attribute for the attribute domain.

10.8.2 Common attributes

Many attributes and attribute groups are not used universally, but are still used across many DITA elements.

Common attribute groups

The following groups are referenced in this specification, and are also used in grammar files when defining attributes for elements.

Architectural attributes
: Includes a set of attributes defined for document level elements such as `<topic>` and `<map>`: `@DITAArchVersion`, `@specializations`, and `@xmlns:ditaarch`.

Common map attributes
: Includes several attributes that are used on a variety of map elements: `@cascade`, `@chunk`, `@collection-type`, `@keyscope`, `@linking`, `@processing-role`, `@search`, and `@toc`.

Complex table attributes
: Includes several attributes that are defined on `<table>` elements (but not on `<simpletable>` elements). Most of these attributes are part of the OASIS Exchange model; table elements generally use only a subset of the attributes defined in this group: `@align`, `@char`, `@charoff`, `@colsep`, `@rowheader`, `@rowsep`, and `@valign`

Data-element attributes
: Includes attributes defined on `<data>` and its many specializations: `@datatype`, `@name`, and `@value`

Date attributes
: Includes attributes that take date values, and are defined on metadata elements that work with date information: `@expiry` and `@golive`

Display attributes
: Includes attributes whose values can be used for affecting the display of many elements: `@expanse`, `@frame`, and `@scale`.

Inclusion attributes
: Includes attributes defined on `<include>` and its specializations: `@encoding` and `@parse`.

Link-relationship attributes
: Includes attributes whose values can be used for representing navigational relationships: `@format`, `@href`, `@type`, and `@scope`.

Simpletable attributes
: Includes attributes that are defined on the `<simpletable>` elements (but not on the OASIS exchange (`<table>` element): `@keycol` and `@relcolwidth`.

Specialization attributes
: Includes attributes designed to be used by specializations, but not intended for direct use by authors: `@specentry` and `@spectitle`.

Topicref-element attributes
: Includes attributes defined on `<topicref>` and on most specializations of the `<topicref>` element: `@copy-to`.

Common attribute definitions

Common attributes, including those in the groups listed above, are defined as follows.

@align (complex table attributes)
Describes the alignment of text in a table column. Allowable values are:

left
Indicates left alignment of the text.

right
Indicates right alignment of the text.

center
Indicates center alignment of the text.

justify
Justifies the contents to both the left and the right.

char
Use the character specified on the `@char` attribute for alignment.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

The `@align` attribute is available on the following table elements: `<tgroup>`, `<colspec>`, and `<entry>`.

@anchorref
Identifies a location within another map file where this map will be anchored at runtime. Resolution of the map is deferred until the final step in the delivery of any rendered content. For example, `anchorref="map1.ditamap/a1"` causes this map to be pulled into the location of the anchor point "a1" inside `map1.ditamap` when `map1.ditamap` is rendered for delivery.

@cascade (common map attributes)

Controls how metadata attributes cascade within a map. There are two defined values that should be supported: "merge" and "nomerge".

If no value is set, and no value cascades from an ancestor element, processors *SHOULD* assume a default of "merge".

See 5.3.1 Cascading of metadata attributes in a DITA map (49) for more information about how this attribute interacts with metadata attributes.

@char (complex table attributes)
Specifies the character for aligning the table entry data.

Default source for `<entry>` elements starting in this column. If character alignment is specified, the value is the single alignment character source for any implied `@char` values for entry immediately in this column. A value of "" (the null string) means there is no aligning character.

For example, if `align="char"` and `char="r"` are specified, then text in the entry should align with the first occurrence of the letter "r" within the entry.

The `@char` attribute is available on the following table elements: `<colspec>` and `<entry>`.

@charoff (complex table attributes)
Specifies the horizontal offset of alignment character when `align="char"`.

Default source for <entry> elements starting in this column. For character alignment on an entry in the column, horizontal character offset is the percent of the current column width to the left of the (left edge of the) alignment character.

This value should be number, greater than 0 and less than or equal to 100.

For example, if align="char", char="r", and charoff="50" are all specified, then text in the entry should align 50% of the distance to the left of the first occurrence of the character "r" within the entry.

The @charoff attribute is available on the following table elements: <colspec> and <entry>.

@chunk (common map attributes)

When a set of topics is transformed using a map, the @chunk attribute allows documents that contain multiple topics to be broken into smaller files and multiple individual topics to be combined into larger combined documents.

For a detailed description of the @chunk attribute and its usage, see 5.4 Chunking (57).

@collection-type (common map attributes)

Collection types describe how links relate to each other. The processing default is "unordered", although no default is specified in the DTD or Schema. Allowable values are:

unordered
Indicates that the order of the child topics is not significant.

sequence
Indicates that the order of the child topics is significant; output processors will typically link between them in order.

choice
Indicates that one of the children should be selected.

family
Represents a tight grouping in which each of the referenced topics not only relates to the current topic but also relate to each other.

@colsep (complex table attributes)

Column separator. A value of 0 indicates no separators; 1 indicates separators.

The @colsep attribute is available on the following table elements: <table>, <tgroup>, <colspec>, and <entry>.

@compact

Indicates close vertical spacing between list items. Expanded spacing is the processing default. The output result of compact spacing depends on the processor or browser. Allowable values are:

yes
Indicates compact spacing.

no
Indicates expanded spacing.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

@copy-to (topicref-element attributes)

Use the @copy-to attribute on the <topicref> element to provide a different resource name for a particular instance of a resource referenced by the <topicref> (for example, to separate out the different versions of the topic, rather than combining them on output). If applicable, the @copy-to

value can include path information. The links and navigation associated with that instance will point to a copy of the topic with the file name you specified.

Applications *MAY* support `@copy-to` for references to local non-DITA resources.

The `@copy-to` attribute is not supported for references to resources where the effective value for `@scope` is "peer" or "external".

Use the `<linktext>` and `<shortdesc>` in the `<topicref>`'s `<topicmeta>` to provide a unique name and short description for the new copy.

@datatype (data-element attributes)
Describes the type of data contained in the `@value` attribute or within the `<data>` element. A typical use of `@datatype` will be the identifying URI for an XML Schema datatype.

@DITAArchVersion (architectural attributes)
Designates the version of the architecture that is in use. The default value will increase with each release of DITA. This attribute is in the namespace "http://dita.oasis-open.org/architecture/2005/". This attribute is defined with the XML data type CDATA, but uses a default value of the current version of DITA. The current default is "2.0".

@encoding (inclusion attributes)
Specifies the character encoding to use when translating the character data from the referenced content. The value should be a valid encoding name. If not specified, processors may make attempts to automatically determine the correct encoding, for example using HTTP headers, through analysis of the binary structure of the referenced data, or the `<?xml?>` processing instruction when including XML as text. The resource should be treated as UTF-8 if no other encoding information can be determined.

When `parse="xml"`, standard XML parsing rules apply for the detection of character encoding. The necessity and uses of `@encoding` for non-standard values of `@parse` are implementation-dependent.

@expanse (display attributes)
Determines the horizontal placement of the element. Allowable values are:

column
Aligns the element with the current column margin.

page
Places the element on the left page margin for left-to-right presentation, or right page margin for right-to-left presentation.

spread
Indicates that, if possible, the object should be rendered across a multi-page spread. If the rendition target does not have anything corresponding to spreads then spread has the same meaning as "page".

textline
Aligns the element with the left (for left to right presentation) or right (for right to left presentation) margin of the current text line and takes indention into account.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

In DITA tables, in place of the `@expanse` attribute used by other DITA elements, the `@pgwide` attribute is used in order to conform to the OASIS Exchange Table Model. The `@pgwide` attribute has a similar semantic ("1"=page width; "0"=resize to galley or column).

Some DITA processors or output formats might not be able to support all values.

@expiry (date attributes)
The date when the information should be retired or refreshed, entered as YYYY-MM-DD, where YYYY is the year, MM is the month from 01 to 12, and DD is the day from 01-31.

@format (link-relationship attributes)
The `@format` attribute identifies the format of the resource being referenced. See 10.8.3.5 The format attribute (386) for details on supported values.

@frame (display attributes)
Specifies which portion of a border should surround the element. Allowable values are:

all
Draw a box around the element

bottom
Draw a line after the element

none
Don't draw any lines around this element

sides
Draw a line at each side of the element

top
Draw a line before the element

topbot
Draw a line both before and after the element

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

Some DITA processors or output formats might not be able to support all values.

@golive (date attributes)
The publication or general availability (GA) date, entered as YYYY-MM-DD, where YYYY is the year, MM is the month from 01 to 12, and DD is the day from 01-31.

@href (link-relationship attributes)
Provides a reference to a resource. See 10.8.3.6 The href attribute (387) for detailed information on supported values and processing implications.

@keycol (simpletable attributes)
Defines the column that contains headings for each row. No value indicates no key column. When present, the numerical value causes the specified column to be treated as a vertical header.

@keyref
`@keyref` provides a redirectable reference based on a key defined within a map. See 10.8.3.7 The keyref attribute (388) for information on using this attribute.

@keyscope (common map attributes)
Specifies that the element marks the boundaries of a key scope. See 10.8.3.9 The keyscope attribute (389) for details on how to use the `@keyscope` attribute.

@linking (common map attributes)
Defines some specific linking characteristics of a topic's current location in the map. If the value is not specified locally, the value might cascade from another element in the map (for cascade rules, see 5.3.1 Cascading of metadata attributes in a DITA map (49)). Allowable values are:

targetonly
A topic can only be linked to and cannot link to other topics.

sourceonly
A topic cannot be linked to but can link to other topics.

normal
A topic can be linked to and can link to other topics. Use this to override the linking value of a parent topic.

none
A topic cannot be linked to or link to other topics.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

@name (data-element attributes)
Defines a unique name for the object.

@parse (inclusion attributes)
Specifies the processing expectations for the referenced resource. Processors must support the following values:

text

The contents should be treated as plain text. Reserved XML characters should be displayed, and not interpreted as XML markup.

xml

The contents of the referenced resource should be treated as an XML document, and the referenced element should be inserted at the location of the `<include>` element. If a fragment identifier is included in the address of the content, processors must select the element with the specified ID. If no fragment identifier is included, the root element of the referenced XML document is selected. Any grammar processing should be performed during resolution, such that default attribute values are explicitly populated. Prolog content must be discarded.

It is an error to use `parse="xml"` anywhere other than within `<foreign>` or a specialization thereof.

Processors may support other values for the `@parse` attribute with proprietary processing semantics. Processors should issue warnings and use `<fallback>` when they encounter unsupported `@parse` values. Non-standard `@parse` instructions should be expressed as URIs.

Note Proprietary `@parse` values will likely limit the portability and interoperability of DITA content, so should be used with care.

@processing-role (common map attributes)
Describes the processing role of the referenced topic. The processing default is "normal". If no value is specified, but the attribute is specified on an ancestor within a map or within the related-links section, the value will cascade from the closest ancestor. Allowable values are:

normal
Normal topic that is a readable part of the information.

resource-only
The topic is used as a resource for processing purposes. This topic should not be included in a rendered table of contents, and the topic should not be rendered on its own.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

@relcolwidth (simpletable attributes)
Specifies the width of each column in relationship to the width of the other columns. The value is a space separated list of relative column widths; each column width is specified as a positive integer or decimal number followed by an asterisk character.

For example, the value `relcolwidth="1* 2* 3*"` gives a total of 6 units across three columns. The relative widths are 1/6, 2/6, and 3/6 (16.7%, 33.3%, and 50%). Similarly, the value `relcolwidth="90* 150*"` causes relative widths of 90/240 and 150/240 (37.5% and 62.5%).

@rowsep (complex table attributes)
Row separator. A value of "0" indicates no separators; "1" indicates separators.

The `@rowsep` attribute is available on the following table elements: `<table>`, `<tgroup>`, `<row>`, `<colspec>`, and `<entry>`.

@rowheader (complex table attributes)
Indicates whether the entries in the respective column *SHOULD* be considered row headers. Allowable values are:

firstcol
Indicates that entries in the first column of the table are functionally row headers (analogous to the way that a `<thead>` element provides column headers). Applies when `@rowheader` is used on the `<table>` element.

headers
Indicates that entries of a column described using the `<colspec>` element are functionally row headers (for cases with more than one column of row headers). Applies when `@rowheader` is used on the `<colspec>` element.

norowheader
Indicates that entries in the first column have no special significance with respect to column headers. Applies when `@rowheader` is used on the `<table>` element.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

Note This attribute is not part of the OASIS Exchange Table model upon which DITA tables are based. Some DITA processors or output formats might not support all values.

The `@rowheader` attribute is available on the following table elements: `<table>` and `<colspec>`.

@scale (display attributes)
Specifies a percentage, selected from an enumerated list, that is used to resize fonts in relation to the normal text size. This attribute is primarily useful for print-oriented display.

The `@scale` attribute provides an acknowledged style-based property directly on DITA elements. For the `<table>` and `<fig>` elements, the intent of the property is to allow authors to adjust font sizes on the content of the containing element, primarily for print accommodation. An `<image>` in these contexts is to be scaled only by its own direct scale property. If not specifically scaled, such an `<image>` is unchanged by the scale property of its parent `<table>` or `<fig>`.

Allowable values are 50, 60, 70, 80, 90, 100, 110, 120, 140, 160, 180, 200, and -dita-use-conref-target (385). Some DITA processors or output formats might not be able to support all values.

@scope (link-relationship attributes)
The `@scope` attribute identifies the closeness of the relationship between the current document and the target resource. Allowable values are "local", "peer", "external", and "-dita-use-conref-target"; see 10.8.3.11 The scope attribute (391) for more information on these values.

@search (common map attributes)
Describes whether the target is available for searching. If the value is not specified locally, the value might cascade from another element in the map (for cascade rules, see 5.3.1 Cascading of metadata attributes in a DITA map (49)). Allowable values are:

yes

no

-dita-use-conref-target (385)

@specentry (specialization attributes)
The specialized entry attribute allows architects of specialized types to define a fixed or default header title for a specialized `<stentry>` element. Not intended for direct use by authors.

@specializations (architectural attributes)
Indicates the specialized attribute domains that are included in the grammar file. This attribute is defined with the XML data type CDATA. The value will differ depending on what domains are included in the current DTD or Schema; a sample value is `@props/audience @props/deliveryTarget @base/newBaseAtt`.

@spectitle (specialization attributes)
The specialized title attribute allows architects of specialized types to define a fixed or default title for a specialized element. Not intended for direct use by authors.

@toc (common map attributes)
Specifies whether a topic appears in the table of contents (TOC). If the value is not specified locally, the value might cascade from another element in the map (for cascade rules, see 5.3.1 Cascading of metadata attributes in a DITA map (49)). Allowable values are:

yes
The topic appears in a generated TOC.

no
The topic does not appear in a generated TOC.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

@type (link-relationship attributes)
Describes the target of a reference. See 10.8.3.12 The type attribute (391) for detailed information on supported values and processing implications.

@value (data-element attributes)
Specifies a value associated with the current property or element.

@valign (complex table attributes)
Indicates the vertical alignment of text in a table entry (cell). Allowable values are:

top
Align the text to the top of the table entry (cell).

bottom
Align the text to the bottom of the table entry (cell).

middle
Align the text to the middle of the table entry (cell).

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

The @valign attribute is available on the following table elements: <thead>, <tbody>, <row>, and <entry>.

@xml:space
This attribute is provided on <pre>, <lines>, and on elements specialized from those. It ensures that parsers in editors and transforms respect the white space, including line-end characters, that is part of the data in those elements. It is intended to be part of the default properties of these elements, and not for authors to change or delete. When defined, it has a fixed value of "preserve".

@xmlns:ditaarch (architectural attributes)
Declares the default DITA namespace. Although this is technically a namespace rather than an attribute, it is included here because it is specified as an attribute in the DTD grammar files distributed by OASIS. The value is fixed to "http://dita.oasis-open.org/architecture/2005/".

Related concepts

Cascading of metadata attributes in a DITA map (49)

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.

10.8.3 Complex attribute definitions

Several DITA attributes require more explanation. Those attributes are collected here.

10.8.3.1 The @conaction attribute

The @conaction attribute allows users to push content from one topic into another. It causes the @conref attribute to work in reverse, so that the content is pushed from the current topic into another, rather than pulled from another topic into the current one. Allowable values for @conaction are: "pushafter", "pushbefore", "pushreplace", mark"", and "-dita-use-conref-target".

Note In the descriptions below, the word *target* always refers to the element referenced by a @conref attribute.

There are three possible functions using the @conaction attribute: replacing an element, pushing content before an element, and pushing content after an element. The @conaction attribute always declares the desired function while the @conref attribute provides the target of the reference using the standard @conref syntax.

In each case, an element pushed using @conref must be of the same type as, or more specialized than, its target. If the pushed element is more specialized than the target, then it should be generalized when the @conref is resolved. This ensures that the content will be valid in the target topic.

- It is valid to push using @conref when the two elements involved are of the same type. For example, a <step> element can use the conref push feature with another <step> as the target of the @conref.
- The target element can be more general than the source. For example, it is legal to push a <step> element to replace a general list item (<li>); the <step> element should be generalized back to a list item during the process.
- It is not possible to push a more general element into a specialized context. For example, it is not legal to push a list item (<li>) in order to replace a <step>, because the list item allows many items that are not valid in the specialized context.

Replacing content in another topic

When the `@conaction` attribute is set to "pushreplace", the source element will replace the target specified on the `@conref` attribute. The pushed content remains in the source topic where it was originally authored.

For example, assume that a task in `example.dita` has the `@id` set to example"", and it contains a `<step>` element with the `@id` set to "b":

```
<task id="example" xml:lang="en">
  <title>Example topic</title>
  <taskbody>
    <steps>
      <step id="a"><cmd>A</cmd></step>
      <step id="b"><cmd>B</cmd></step>
      <step id="c"><cmd>C</cmd></step>
    </steps>
  </taskbody>
</task>
```

In order to replace the step with `id="b"`, another topic must combine a `@conaction` value of "pushreplace" with a `@conref` attribute that references this `<step>`:

```
<task id="other" xml:lang="en">
  ...
   <step conaction="pushreplace"
         conref="example.dita#example/b">
     <cmd>Updated B</cmd>
   </step>
  ...
</task>
```

The result will be an updated version of `example.dita` which contains the pushed `<step>`:

```
<task id="example" xml:lang="en">
  <title>Example topic</title>
  <taskbody>
    <steps>
      <step id="a"><cmd>A</cmd></step>
      <step id="b"><cmd>Updated B</cmd></step>
      <step id="c"><cmd>C</cmd></step>
    </steps>
  </taskbody>
</task>
```

When resolving a conref push action, attributes are resolved using the same precedence as for normal `@conref`, with one exception. Attributes on the element with the `@conref` attribute (in this case, the source doing the push) will take priority over those on the referenced element. The exception is that if the source element does not specify an ID, the ID on the referenced element remains; if the source element does specify an ID then that replaces the ID on the referenced element.

It is an error for two source topics to replace the same element. Applications *MAY* warn users if more than one element attempts to replace a single target.

Pushing content before or after another element

Setting the `@conaction` attribute to "pushbefore" allows an element to be pushed before the element referenced by the `@conref` attribute. Likewise, setting the `@conaction` attribute to "pushafter" allows an element to be pushed after the element referenced by the `@conref` attribute. Multiple sources can push content before or after the same target; the order in which that content is pushed is undefined.

When an element is pushed before or after a target, the resulting document will have at least two of that element. Because this is not always valid, a document attempting to push content before or after a target must take an extra step to ensure that the result will be valid. The extra step makes use of the `conaction="mark"` value.

When pushing before, the `@conref` attribute itself looks just as it did when replacing, but the `@conaction` attribute is set to mark"" because it is marking the target element. This element remains empty; its purpose is to ensure that it is legal to have more than one of the current element. Immediately before the element which marks the target, you will place the content that you actually want to push. This element will set the `@conaction` attribute to "pushbefore".

When pushing after, the procedure is the same, except that the order of the elements is reversed. The element with `conaction="pushafter"` comes immediately after the element which marks the target.

Attributes on the element which is pushed (the one with `conaction="pushbefore"`) must be retained on the target, apart from the `@conaction` attribute itself. If this causes the result document to end up with duplicate IDs, an application can recover by dropping the duplicate ID, modifying it to ensure uniqueness, or warning the user.

The following restrictions apply when pushing content before or after an element:

- The elements that use `conaction="mark"` and `conaction="pushbefore"` are the same type as each other and appear in sequence. This restriction prevents a topic from trying to push a `<body>` element before or after another `<body>` element, because it is not valid to have two body elements in sequence.
- Either the container elements of the source and target match, or the container of the source element is be a specialization of the target's container. This is also to ensure validity of the target; for example, while it is possible to include multiple titles in a `<section>`, it is not possible to do so in a figure. Comparing the parents prevents a second `<section>` title from being pushed before a figure title (the resulting figure would not be valid DITA). This restriction only applies to the "pushbefore" or" pushafter" actions, not to the "pushreplace" action.

When content is pushed from one topic to another, it is still rendered in the original context. Processors might delete the empty element that has the `conaction="mark"` attribute. In order to push content from a topic without actually rendering that topic on its own, the topic should be referenced from the map with the `@processing-role` attribute set to "resource-only".

Example: pushing an element before the target

The following example pushes a `<step>` before "b" in the `example.dita` file shown above.

```
<step conaction="pushbefore"><cmd>Do this before B</cmd></step>
<step conaction="mark" conref="example.dita#example/b">
  <cmd/>
</step>
```

The result contains the pushed `<step>` element before "b".

```
<task id="example" xml:lang="en">
  <title>Example topic</title>
  <taskbody>
    <steps>
      <step id="a"><cmd>A</cmd></step>
      <step><cmd>Do this before B</cmd></step>
      <step id="b"><cmd>B</cmd></step>
      <step id="c"><cmd>C</cmd></step>
    </steps>
  </taskbody>
</task>
```

Example: pushing an element after the target

Pushing an element after a target is exactly the same as pushing before, except that the order of the "mark" element and the pushed element are reversed.

```
<step conaction="mark" conref="example.dita#example/b">
  <cmd/>
</step>
<step conaction="pushafter"><cmd>Do this AFTER B</cmd></step>
```

In this case the resulting document has the pushed content after `<step>` b:

```
<task id="example" xml:lang="en">
  <title>Example topic</title>
  <taskbody>
    <steps>
      <step id="a"><cmd>A</cmd></step>
      <step id="b"><cmd>B</cmd></step>
      <step><cmd>Do this AFTER B</cmd></step>
      <step id="c"><cmd>C</cmd></step>
    </steps>
  </taskbody>
</task>
```

Combining @conaction with @conkeyref or @conrefend

The `@conkeyref` attribute can be used as an indirect way to specify a `@conref` target. If the `@conkeyref` attribute is specified on an element that also uses the `@conaction` attribute, the `@conkeyref` attribute is used to determine the target of the conref push (as it would normally be used to determine the target of `@conref`).

The conref push function does not provide the ability to push a range of elements, so it is an error to specify the `@conrefend` attribute together with the `@conaction` attribute. If the two are specified together an application can recover by warning the user, ignoring the `@conrefend` attribute, or with some other implementation strategy.

10.8.3.2 The @conrefend attribute

The `@conrefend` attribute is used when referencing a range of elements with the conref mechanism. The `@conref` or `@conkeyref` attribute references the first element in the range, while `@conrefend` references the last element in the range.

Using @conref together with @conrefend

The following markup rules apply when using or implementing `@conrefend`:

- The start and end elements of a range *MUST* be of the same type as the referencing element or generalizable to the referencing element.
- The start and end elements in a range *MUST* share the same parent, and the start element *MUST* precede the end element in document order.
- The parent of the referencing element *MUST* be the same as the parent of the referenced range or generalizable to the parent of the referencing element.

In addition, several other items must be taken into account:

- Processors will resolve the range by pulling in the start target and following sibling XML nodes across to and including the end target.
- As with `@conref`, if the `@conrefend` references a more specialized version of the referencing element, applications should generalize the target when resolving.

- It is not valid to use `@conrefend` to reference a more general version of an element (such as using `<step>` to reference an `<li>` element).
- Other nodes (such as elements or text) between the start and end of a range do not have to match the referencing element.
- With single conref, an `@id` attribute from the referenced element will not be preserved on the resolved content. With a range, an `@id` on both the start and the end elements will not be preserved. `@id` attributes on intermediate or child nodes should be preserved; if this results in duplicate `@id` values, an application can recover by changing the `@id`, warning the user, or implementing another strategy.
- With a single conref, attributes specified on the referencing elementcan be used to override attributes on the referenced element. With a conref range, the same is true, with the following clarifications:
  - When an `@id` attribute is specified on the referencing element, it will only be preserved on the first element of the resolved range.
  - When other attributes are specified, they will only apply to referenced elements of the same type. For example, if `<step>` is used to pull in a range of sequential `<step>` elements, locally specified attributes apply to all steps in the range. If `<ol>` is used to pull in a series of (`<ol>`, `<p>`, `<ol>`), locally specified attributes apply only to the `<ol>` elements in that range.

Example: reusing a set of list items

Figure 160: List example: Source topic.dita with ids

```
<topic id="x">
     ...
     <body>
         <ol>
             <li id="apple">A</li>
             <li id="bear">B</li>
             <li id="cat">C</li>
             <li id="dog">D</li>
             <li id="eel">E</li>
         </ol>
     </body>
 </topic>
```

Figure 161: List example: Reusing topic with conrefs

```
 <topic id="y">
     ...
     <body>
         <ol>
             <li>My own first item</li>
             <li conref="topic.dita#x/bear" conrefend="topic.dita#x/dog"/>
             <li>And a different final item</li>
         </ol>
     </body>
 </topic>
```

Figure 162: List example: Processed result of reusing topic

```
 <topic id="y">
     ...
     <body>
         <ol>
             <li>My own first item</li>
```

```
            <li>B</li>
            <li id="cat">C</li>
            <li>D</li>
            <li>And a different final item</li>
        </ol>
    </body>
</topic>
```

Example: Reusing a set of blocks

Figure 163: Block level example: Source topic.dita with ids

```
<topic id="x">
     ...
     <body>
   <p id="p1">First para</p>
      <ol id="mylist">
     <li id="apple">A</li>
     <li id="bear">B</li>
     <li id="cat">C</li>
     <li id="dog">D</li>
     <li id="eel">E</li>
   </ol>
   <p id="p2">Second para</p>
     </body>
 </topic>
```

Figure 164: Block level example: Reusing topic with conrefs

```
 <topic id="y">
     ...
     <body>
         <p conref="topic.dita#x/p1" conrefend="topic.dita#x/p2"/>
     </body>
 </topic>
```

Figure 165: Block level example: Processed result of reusing topic

```
 <topic id="y">
     ...
     <body>
    <p>First para</p>
       <ol id="mylist">
      <li id="apple">A</li>
      <li id="bear">B</li>
      <li id="cat">C</li>
      <li id="dog">D</li>
      <li id="eel">E</li>
    </ol>
    <p>Second para</p>
     </body>
</topic>
```

Using @conrefend together with @conkeyref

When the `@conkeyref` attribute is used in place of `@conref`, a key is used to address the target of the reference. The `@conrefend` attribute, which indicates the end of a `@conref` range, cannot use a key. Instead the map or topic element addressed by the key name component of the `@conkeyref` is used in place of whatever map or topic element is addressed by the `@conrefend` attribute.

For example, if the value of the `@conkeyref` attribute is "config/step1" and the value of the `@conrefend` is "defaultconfig.dita#config/laststep", the conref range will end with the step that has `id="laststep"` in

whatever topic is addressed by the key name "config". If the key name "config" is not defined, and the `@conref` attribute itself is not present for fallback, the `@conrefend` attribute is ignored.

Example: Combining @conrefend with @conkeyref

Figure 166: Defining and referencing a key with @conkeyref

In this example the key "xmp" is defined as the first topic in the file `examples.dita`.

```
<map>
  <!-- ... -->
  <keydef keys="xmp" href="examples.dita"/>
  <!-- ... -->
</map>

examples.dita:
<topic id="examples">
  <title>These are examples</title>
  <body>
    <ul>
      <li id="first">A first example</li>
      <li>Another trivial example</li>
      <li id="last">Final example</li>
    </ul>
  </body>
</topic>
```

To reuse these list items by using the key, the `@conkeyref` attribute combines the key itself with the sub-topic id (first) to define the start of the range. The `@conrefend` attribute defines a default high-level object along with the sub-topic id (last) that ends the range:

```
  <li conkeyref="xmp/first"
      conrefend="default.dita#default/last"/>
```

The `@conkeyref` attribute uses a key to reference the first topic in `examples.dita`, so the range begins with the object `examples.dita#examples/first`. The high-level object in the `@conrefend` attribute (`default.dita#default`) is replaced with the object represented by the key (the first topic in `examples.dita`), resulting in a range that ends with the object `examples.dita#examples/last`.

Figure 167: Combining @conref, @conkeyref, and @conrefend

When `@conref`, `@conkeyref`, and `@conrefend` are all specified, the key value takes priority.

```
  <li conkeyref="thisconfig/start"
      conref="standardconfig.dita#config/start"
      conrefend="standardconfig.dita#config/end"/>
```

- If the key "thisconfig" is defined as `mySpecialConfig.dita#myconfig`, then the range will go from the list item with `id="start"` to the list item with`id="end"` in the topic `mySpecialConfig.dita#myconfig`.
- If the key "thisconfig" is defined as `myConfig.dita`, then the range will go from the list item with `id="start"` to the list item with `id="end"` within the first topic in `myConfig.dita`.
- If the key "thisconfig" is not defined, then the unchanged `@conref` and `@conrefend` attributes are used as fallback. In that case, the range will go from the list item with `id="start"` to the list item with `id="end"` within the topic `standardconfig.dita#config`.

Error conditions

When encountering an error condition, an implementation can issue an error message.

Condition or Issue	Result
The `@conref` attribute cannot be resolved in the target document (the target element might have been removed or its id has changed).	The `@conref` is ignored.
The `@conrefend` attribute cannot be resolved in the target document (the target element might have been removed or its id has changed).	Range cannot be resolved, optional recovery processes the result as a simple conref.
Start and end elements are not siblings in the target document.	If the start element exists, optional recovery processes the result as a simple conref.
End element occurs before the start element in the target document.	If the start element exists, optional recovery processes the result as a simple conref.
An element has a `@conrefend` attribute but is missing the `@conref` attribute.	No result.

10.8.3.3 The @conkeyref attribute

The `@conkeyref` attribute provides an indirect content reference to topic elements, map elements, or elements within maps or topics. When the DITA content is processed, the key references are resolved using key definitions from DITA maps.

For content references from map elements to map elements or topic elements to topic elements, the value of the `@conkeyref` attribute is a key name, where the key must be bound to a map element (for references from map elements) or a topic element (for references from topic elements). For all other elements, the value of the `@conkeyref` attribute is a key name, an optional slash ("/"), and the ID of the target element, where the key name must be bound to the map or topic that contains the topic element.

When the key name specified by the `@conkeyref` attribute is not defined and the element also specifies a `@conref` attribute, the `@conref` attribute is used to determine the content reference relationship. If no `@conref` attribute is specified there is no content reference relationship.

Processors *SHOULD* issue a warning when a `@conkeyref` reference cannot be resolved and there is no `@conref` attribute to use as a fallback. Processors *MAY* issue a warning when a `@conkeyref` cannot be resolved to an element and a specified `@conref` is used as a fallback.

The `@conrefend` attribute, which defines the end of a conref range, cannot include a key. Instead the map or topic element addressed by the key name component of the `@conkeyref` is used in place of whatever map or topic element is addressed by the `@conrefend` attribute. See Using conrefend together with conkeyref (382) for more information and for examples of this behavior.

10.8.3.4 The @conref attribute

The `@conref` attribute is used to reference content that can be reused. It allows reuse of DITA elements, including topic or map level elements.

The value of the `@conref` attribute must be a URI reference to a DITA element. See 6.3 URI-based (direct) addressing (74) for details on specifying URI references to DITA elements. As with other DITA references, a `@conref` attribute that references a resource without an ID is treated as a reference to the first topic or map in the document.

Note When using the `@conref` attribute on an element, the content of that element is ignored. For example, if a phrase is marked up like this:

```
<ph conref="#topic/ph">Something</ph>
```

the word "Something" will be replaced by the content of the referenced <ph> element.

Related concepts

Content reference (conref) (112)

The DITA conref attributes provide mechanisms for reusing content. DITA content references support reuse scenarios that are difficult or impossible to implement using other XML-based inclusion mechanisms like XInclude and entities. Additionally, DITA content references have rules that help ensure that the results of content inclusion remain valid after resolution

10.8.3.4.1 Using the "-dita-use-conref-target" value

The value "-dita-use-conref-target" is available on enumerated attributes and can also be specified on other attributes. When an element uses @conref to pull in content, for any of its attributes assigned a value of "-dita-use-conref-target", the resulting value for those attributes is also pulled in from the referenced element.

Ordinarily, when an element uses @conref, any other attributes specified locally will be preserved when the reference is resolved. This causes problems when attributes are required, because required attributes must be specified regardless of whether the @conref attribute is present. The purpose of the "-dita-use-conref-target" value is to allow the author to specify a value for a required attribute while still allowing the conref resolution process to use the matching attribute from the referenced element. The value has the same result when the attribute is not required.

The" -dita-use-conref-target" token is allowed on any attribute where it is not prohibited by the XML grammar files or by the specification. For example, while @cols on the <tgroup> element is defined as being a number, this token is implicitly allowed in order to support conref processing for <tgroup>. However, the token is not allowed for the @id attribute on the <topic> element, because "-dita-use-conref-target" does not fit the syntax required by the XML grammar files.

This example shows a DITA map where the <topichead> element uses @conref. It specifies the @deliveryTarget attribute as well as the @toc attribute. In the resolved element, @deliveryTarget from the referencing element is not preserved because it uses "-dita-use-conref-target". The @toc attribute from the referencing element overrides the @toc attribute on the referenced element using normal conref resolution rules.

Figure 168: Before resolution

```
<map><title>Conref demonstration</title>
  <topichead id="heading"
             deliveryTarget="pdf"
             toc="yes"
             linking="normal">
    <topicmeta>
       <navtitle>This is a heading</navtitle>
    </topicmeta>
    <topicref href="topic.dita"/>
  </topichead>

  <topichead conref="#heading"
             deliveryTarget="-dita-use-conref-target"
             toc="no">
  </topichead>
</map>
```

Figure 169: Effective result post-resolution

```
<map><title>Conref demonstration</title>
  <topichead id="heading"
             deliveryTarget="pdf"
```

```
            toc="yes"
            linking="normal">
    <topicmeta>
       <navtitle>This is a heading</navtitle>
    </topicmeta>
    <topicref href="topic.dita"/>
  </topichead>

  <topichead deliveryTarget="pdf"
            toc="no"
            linking="normal">
    <topicmeta>
       <navtitle>This is a heading</navtitle>
    </topicmeta>
    <topicref href="topic.dita"/>
  </topichead>
</map>
```

Related concepts

Content reference (conref) (112)

The DITA conref attributes provide mechanisms for reusing content. DITA content references support reuse scenarios that are difficult or impossible to implement using other XML-based inclusion mechanisms like XInclude and entities. Additionally, DITA content references have rules that help ensure that the results of content inclusion remain valid after resolution

10.8.3.5 The @format attribute

The `@format` attribute identifies the format of the resource that is referenced. If no value is specified, but the attribute is specified on an ancestor within a map or within the related-links section, the value will cascade from the closest ancestor.

The following values for `@format` have special processing implications:

dita

The destination uses DITA topic markup or markup specialized from a DITA topic. Unless otherwise specified, when `@format` is set to "dita", the value for the `@type` attribute will be treated as topic"".

ditamap

The linked-to resource is a DITA map. It represents the referenced hierarchy at the current point in the referencing map. References to other maps can occur at any point in a map, but because relationship tables are only valid as children of a map, referenced relationship tables are treated as children of the referencing map.

Note If a `<topicref>` element that references a map contains child `<topicref>` elements, the processing behavior regarding the child `<topicref>` elements is undefined.

(no value)

The processing default is used. The processing default for the `@format` attribute is determined by inspecting the value of the `@href` attribute. If the `@href` attribute specifies a file extension, the processing default for the `@format` attribute is that extension, after conversion to lower-case and with no leading period. The only exception to this is if the extension is `.xml`, in which case the default value for `@format` is "dita". If there is no extension, but the `@href` value is an absolute URI whose scheme is "http" or "https", then the processing default is "html". In all other cases where no extension is available, the processing default is "dita."

If the actual format of the referenced content differs from the effective value of the `@format` attribute, and a processor is capable of identifying such cases, it *MAY* recover gracefully and treat the content as its actual format, but *SHOULD* also issue a message.

For DITA processors that support Lightweight DITA, the following values for `@format` have special processing implications:

xdita
The format of the resource is XDITA.

mdita
The format of the resource is MDITA.

hdita
The format of the resource is HDITA.

xditamap
The resource is an XDITA map.

hditamap
The resource is an HDITA map.

mditamap
The resource is an MDITA map.

For other formats, using the file extension without the "." character typically represents the format. For example, the following values are all possible values for `@format`:

html
The format of the linked-to resource is HTML or XHTML.

pdf
The format of the linked-to resource is PDF.

txt
The format of the linked-to resource is a text file.

10.8.3.6 The @href attribute

The `@href` attribute is used to reference another DITA topic or map, a specific element inside a DITA topic or map, an external Web page, or another non-DITA resource.

The value of a DITA `@href` attribute must be a valid URI reference [RFC 3986].

if the value of the `@href` attribute is not a valid URI reference, an implementation *MAY* generate an error message; it *MAY* recover from this error condition by attempting to convert the value to a valid URI reference.

Note that the path separator character in a URI is the forward slash (“/”); the backward slash character (“\”) is not permitted unescaped within URIs.

When an `@href` attribute references a DITA resource, an `@href` value that consists of a URI without a fragment identifier resolves to the document element in the referenced document. For the purposes of rendering, such as when a `<topicref>` reference to a DITA document is used to render the content as HTML, this means that all topics (and topic specializations) in the target document are included in the reference. For the purpose of linking, the reference resolves to the first (or only) topic (or topic specialization) in the document.

An `@href` value that consists of a URI with a fragment identifier must have a DITA local identifier as the portion after the hash. A DITA local identifier consists of *`topicID/elementID`* for a subelement of a topic, and of *elementID* for topics, maps, and subelements of a map. If the topic referenced by a DITA local identifier is for the same topic, then *topicID* can be replaced by a period; see 7.3.4 Processing xrefs and conrefs within a conref (114) for more information on how this syntax relates to conref resolution.

Note that certain characters—including but not limited to the hash sign ("#"), question mark ("?"), back slash ("\"), and space—are not permitted unescaped within URIs. Such characters must be percent-

encoded. Also note that the ampersand ("&") and less than (“<”) characters are not permitted in XML attribute values; they must be represented by appropriate character or entity references. Some tools might perform this encoding automatically, while other tools might require that users either avoid the special characters or manually insert the encoding.

Example: Common syntax for the @href attribute

The following table includes some examples of common `@href` syntax. Note that these examples represent only a few common scenarios and are not all inclusive.

Target	Syntax
The first topic in a DITA document	`href="file.dita"`
A specific topic in a DITA document	`href="file.dita#`*`topicid`*`"`
A non-topic element inside a DITA topic	`href="#`*`topicid/elementid`*`"`
A non-topic element inside the same DITA topic as the reference	`href="#./`*`elementid`*`"`
An element in a DITA map	`href="myMap.ditamap#`*`map-branch`*`"`
An image	`href="exampleImage.jpg"`
An external resource	`href="http://www.example.org"`

where:

- *topicid* is the value of the `@id` attribute on the DITA topic.
- *elementid* is the value of the `@id` attribute on the (non-topic) DITA element.
- *map-branch* is the value of the `@id` attribute on the DITA map element.

10.8.3.7 The @keyref attribute

The `@keyref` attribute provides an indirect, late-bound reference to topics, to collections of topics (ditabase), to maps, to referenceable portions of maps, to non-DITA documents, to external URIs, or to XML content contained within a key definition topic reference. When the DITA content is processed, the key references are resolved using key definitions from DITA maps.

For elements that only refer to topics or non-DITA resources, the value of the `@keyref` attribute is a key name. For elements that can refer to elements within maps or topics, the value of the `@keyref` attribute is a key name, a slash ("/"), and the ID of the target element, where the key name must be bound to either the map or topic that contains the target element.

Related concepts

Indirect key-based addressing (77)

DITA keys provide an alternative to direct addressing. The key reference mechanism provides a layer of indirection so that resources (for example, URIs, metadata, or variable text strings) can be defined at the DITA map level instead of locally in each topic.

10.8.3.8 The @keys attribute

A `@keys` attribute consists of one or more space-separated keys. Map authors define keys using a `<topicref>` or `<topicref>` specialization that contains the `@keys` attribute. Each key definition introduces an identifier for a resource referenced from a map. Keys resolve to the resources given as the `@href` value on the key definition `<topicref>` element, to content contained within the key definition `<topicref>` element, or both.

The `@keys` attribute uses the following syntax:

- The value of the `@keys` attribute is one or more space-separated key names.
- Key names consist of characters that are legal in a URI. The case of key names is significant.
- The following characters are prohibited in key names: "{", "}", "[", "]", "/", "#", "?", and whitespace characters.

A key cannot resolve to sub-topic elements, although a `@keyref` attribute can do so by combining a key with a sub-topic element id.

Related concepts

Indirect key-based addressing (77)

DITA keys provide an alternative to direct addressing. The key reference mechanism provides a layer of indirection so that resources (for example, URIs, metadata, or variable text strings) can be defined at the DITA map level instead of locally in each topic.

10.8.3.9 The @keyscope attribute

The `@keyscope` attribute consists of one or more space-separated key scope names. Map authors define the boundaries for key scopes by specifying the `@keyscope` attribute on `<map>` elements, `<topicref>` elements, or elements that are specializations of `<map>` or `<topicref>`. Such elements, their contents, and any locally-scoped content referenced from within the element, are considered to be part of the scope. Keys defined within a scope are only directly referenceable from within the same scope. They can be referenced from the parent scope using the scope's name, followed by a period, followed by the key name.

All key scopes are contiguous and non-intersecting. Within a root map, two distinct key scopes with the same name have no relationship with each other aside from that implied by their relative locations in the key scope hierarchy. They do not, for example, share key definitions. The only processing impact of a key scope's names is in defining the prefixes used when contributing qualified key names to the parent scope. For example, consider the following map segment:

```
<map>
  <topicgroup keyscope="xyz" id="scope1">
    <keydef keys="a" id="def1"/>
    <!-- other topic references -->
  </topicgroup>
  <topicgroup keyscope="xyz" id="scope2">
    <keydef keys="a" id="def2"/>
    <!-- other topic references -->
  </topicgroup>
  <!-- lots of other content -->
</map>
```

This map creates two distinct scopes that happen to use the same name ("xyz"). This results in the following:

- Each `<topicgroup>` sets a scope of "xyz" and includes a key "a". From outside of those two scopes, references to `keyref="xyz.a"` (key "a" within the scope "xyz") will always resolve to the first instance of that value, which is in the first `<topicgroup>`.
- Within the first `<topicgroup>`, content uses `keyref="a"` will resolve to the key in that branch (defined on the element with `id="def1"`).
- Within the second `<topicgroup>`, content uses `keyref="a"` will resolve to the key in that branch (defined on the element with `id="def2"`).

10.8.3.10 The @role and @otherrole attributes

The `@role` attribute defines the role the target topic plays in relationship with the current topic. For example, in a parent/child relationship, the role would be "parent" when the target is the parent of the current topic, and "child" when the target is the child of the current topic. This structure could be used to sort and classify links at display time.

Supported values for @role

Allowable values for the `@role` attribute are:

parent
Indicates a link to a topic that is a parent of the current topic.

child
Indicates a link to a direct child such as a directly nested or dependent topic.

sibling
Indicates a link between two children of the same parent topic.

friend
Indicates a link to a similar topic that is not necessarily part of the same hierarchy.

next
Indicates a link to the next topic in a sequence.

previous
Indicates a link to the previous topic in a sequence.

cousin
Indicates a link to another topic in the same hierarchy that is not a parent, child, sibling, next, or previous.

ancestor
Indicates a link to a topic above the parent topic.

descendant
Indicates a link to a topic below a child topic.

other
Indicates any other kind of relationship or role. Enter that role as the value for the `@otherrole` attribute.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

The `@otherrole` attribute is available to specify an alternate role that is not available in the list above, and should be used in conjunction with `role="other"`.

10.8.3.11 The @scope attribute

The `@scope` attribute identifies the closeness of the relationship between the current document and the target resource.

- Set `@scope` to "local" when the resource is part of the current set of content.
- Set `@scope` to "peer" when the resource is part of the current set of content but might not accessible at build time, or for maps to be treated as root maps for the purpose of creating map-to-map key references (peer maps). An implementation might open such resources in the same browser window to distinguish them from those with `@scope` set to "external".
- Set `@scope` to "external" when the resource is not part of the current information set and should open in a new browser window.
- See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information on "-dita-use-conref-target".

If no value is specified, but the attribute is specified on an ancestor within a map or within the related-links section, the value will cascade from the closest ancestor. The processing default is determined by the value of the `@href` attribute. In most cases, the processing default is "local". However the processing default is "external" whenever the absolute URI in the `@href` attribute begins with one of the following schemes:

- "http"
- "https"
- "ftp"
- "mailto"

Processors can consider additional URI schemes as "external" by default. Processors *MUST* always consider relative URIs as "local" by default.

10.8.3.12 The @type attribute

The `@type` attribute is used on linking elements to describe the target of a cross-reference. It also is used on the `<note>` element to describe the note type, as well as on several other elements for varying purposes.

The descriptions for the `@type` attribute on linking elements and on `<note>` are included in this section; for other elements, such as `<audience>`, `<copyright>`, and `<object>`, the description can be found with the topic for the specific element.

Using @type on a linking element

The `@type` attribute describes the target of a cross-reference and might generate cross-reference text based on that description. Only the `<xref>` element can link to content below the topic level: other types of linking can target whole topics, but not parts of topics. Typically `<xref>` should also be limited to topic-level targets, unless the output is primarily print-oriented. Web-based referencing works best at the level of whole topics, rather than anchor locations within topics.

If not explicitly specified on an element, the `@type` attribute value cascades from the closest ancestor element. If there is no explicit value for the `@type` attribute on any ancestor, a default value of “topic” is used.

During output processing for references to DITA topics (`format="dita"`), it is an error if the actual type of a DITA topic and the explicit, inherited, or default value for the `@type` attribute are not the same as or a specialization of the `@type` attribute value. In this case, an implementation *MAY* give an error message, and *MAY* recover from this error condition by using the `@type` attribute value.

During output processing for references to non-DITA objects (that is, either scope is “external" or format is neither "dita" nor "ditamap") or other cases where the type of the referenced item cannot be determined from the item itself, the explicit, inherited, or default value for the `@type` attribute is used without any validation.

When a referencing element is first added to or updated in a document, DITA-aware editors *MAY* set the `@type` attribute value based on the actual type of a referenced DITA topic.

If the `@type` attribute is specified when referencing DITA content, it should match one of the values in the referenced element's `@class` attribute. The `@type` value can be an unqualified local name (for example, "fig") or a qualified name exactly as specified in the `@class` attribute (for example, "mymodule/mytype"). Processors might ignore qualified names or consider only the local name.

For example, if the value is set to `type="topic"`, the link could be to a generic topic, or any specialization of topic, including concept, task, and reference. Applications *MAY* issue a warning when the specified or inherited `@type` attribute value does not match the target (or a specialization ancestor of the target).

Some possible values for use on the `<xref>` element and its specializations include:

fig
: Indicates a link to a figure.

table
: Indicates a link to a table.

li
: Indicates a link to an ordered list item.

fn
: Indicates a link to a footnote.

section
: Indicates a link to a section.

Other values that can be used on any linking element include:

concept, task, reference, topic
: Cross-reference to a topic type.

(no value)
: The processor should retrieve the actual type from the target if available. If the type cannot be determined, the default should be treated as "topic".

-dita-use-conref-target
: See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

Other values can be used to indicate other types of topics or elements as targets. Processing is only required to support the above list or specializations of types in that list. Supporting additional types as targets might require the creation of processing overrides.

Using @type in a <note> element

In a `<note>` element, this defines the type of note. For example, if the note is a tip, the word **Tip** might be used to draw the reader's attention to it. The values danger, warning, and notice have meanings that are based on ANSI Z535 and ISO 3864 regulations.

If `@type` is set to "other", the value of the `@othertype` attribute can be used. If you use `@othertype`, many processors will require additional information on how to process the value. Allowable values for the `@type` attribute are:

note
This is just a note.

attention
Please pay extra attention to this note.

caution
Care is required when proceeding.

danger
Important! Be aware of this before doing anything else. When used with the `<hazardstatement>` element, this indicates an imminently hazardous situation which, if not avoided, **will** result in death or serious injury.

fastpath
This note will speed you on your way.

important
This note is important.

notice
Indicates a potential situation which, if not avoided, might result in an **undesirable result or state**.

remember
Don't forget to do what this note says.

restriction
You can't do what this note says.

tip
This is a fine little tip.

warning
Indicates a potentially hazardous situation. When used with the `<hazardstatement>` element, this indicates a situation which, if not avoided, could result in death or serious injury.

trouble
Provides information about how to remedy a trouble situation.

other
This is something other than a normal note.

-dita-use-conref-target
See 10.8.3.4.1 Using the -dita-use-conref-target value (385) for more information.

11 Conformance

An implementation is a conforming implementation of DITA if the implementation meets the conditions that are described in Section 4.1. A document is a conforming DITA document if the document meets the conditions in that are described in Section 4.2.

Conformance to the DITA specification allows documents and document types that are used with different processors to produce the same or similar results with little or no reimplementation or modification. Conformance also allows DITA specializations to work with any conforming DITA application, with at least the same level of support available to unspecialized documents.

4.1 Conformance of DITA implementations

The DITA specification defines several core features, as summarized in the following list. Any implementation that supports a feature *MUST* conform to all rules laid out in the section that describes the feature.

1. Specialization-based processing, as described in X.
2. Resolving links to elements in DITA documents, as described in section X.
3. Resolving `@keyref` attributes to a key defined in a map, as described in section X.
4. Resolving `@keyref` attributes across key scopes, as described in section X.
5. Pulling content references, as described in 7.3 Content reference (conref) (112)
6. Pushing content references, as described in 7.3 Content reference (conref) (112).
7. Resolving conditional processing based on DITAVAL documents, as described in 7.4 Conditional processing (profiling) (117).
8. Resolving branch filtering markup, as described in 7.5 Branch filtering (122).
9. Resolving `@chunk` attributes, as described in 5.4 Chunking (57).

In addition, certain DITA elements have normative rules associated regarding how to render or process those elements.

1. `<desc>`, as described in 10.3.2.5 desc (214)
2. `<draft-comment>`, as described in 10.3.2.10 draft-comment (218)
3. `<image>`, as described in 10.3.2.18 image (224)
4. `<linklist>`, as described in 10.3.5.3 linklist (255)
5. `<pre>`, as described in 10.3.2.32 pre (237)
6. `<q>`, as described in 10.3.2.33 q (237)
7. `<related-links>`, as described in 10.3.1.5 related-links (205)
8. `<relcolspec>`, as described in 10.4.1.10 relcolspec (275)
9. `<reltable>`, as described in 10.4.1.6 reltable (272)
10. `<shortdesc>`, as described in 10.3.1.6 shortdesc (206)
11. `<title>`, as described in 10.3.1.7 title (208)
12. `<titlealt>`, as described in 10.3.1.8 titlealt (208)
13. `<topichead>`, as described in 10.6.7.6 topichead (348)

Conforming DITA implementations *SHOULD* include a conformance statement that gives the version of the DITA specification that is supported, indicate if all features from the list above are supported, and indicate that all normative rendering rules are supported.

If only a subset of features is supported, implementations *SHOULD* indicate which features are (or are not) supported. If an implementation supports rendering DITA elements but does not render all elements as described above, that application *SHOULD* indicate which elements are (or are not) supported.

Not all DITA features are relevant for all implementations. For example, a DITA editor that does not render content references in context does not need to conform to rules regarding the `@conref` attribute. However, any application that renders content references *MUST* conform to the rules described in7.3 Content reference (conref) (112).

Implementations that support only a subset of DITA features are considered conforming as long as all supported features follow the requirements that are given in the DITA specification. An implementation that does not support a particular feature *MUST* be prepared to interoperate with other implementations that do support the feature.

4.2 Conformance of DITA documents

A document conforms with the DITA standard if it meets all of the following conditions.

1. A DITA document that refers to document type shells distributed by OASIS *MUST* be valid according to both the grammar files and any assertions provided in the language reference.
2. If a DITA document refers to a custom document type shell, that shell *MUST* also conform to the rules laid out in X.X.X.X Rules for document-type shells.
3. If a DITA document's custom document type shell includes constraints, that shell *MUST* also conform to the rules laid out in X.X.X.X Constraint rules
4. If a DITA document uses specialized elements or attributes, those elements or attributes *MUST* also conform to the rules laid out in X.X.X Specialization rules for element types, X.X.X Specialization rules for attributes, and X.X.X Class attribute rules and syntax.

A Acknowledgments

(Non-normative) Many members of the OASIS DITA Technical Committee participated in the creation of this specification and are gratefully acknowledged.

Robert Anderson, Oracle
Deb Bissantz, Vasont Systems
Bill Burns, Healthwise
Carsten Brennecke, SAP
Bill Burns, Healthwise
Stan Doherty, Individual member
Kristen James Eberlein, Eberlein Consulting LLC
Carlos Evia, Virginia Tech
Nancy Harrison, Individual member
Alan Houser, Individual member
Scott Hudson, ServiceNow
Gershon Joseph, Precision Content
Eliot Kimber, Individual member
Zöe Lawson, Casenet LLC
Tom Magliery, JustSystems
Chris Nitchie, Individual member
Keith Schengili-Roberts, Individual member
Eric Sirois, IXIASOFT
Dawn Stevens, Comtech Services
Bob Thomas, Individual member
Frank Wegmann, Software AG

Comment by Kristen J Eberlein on 31 August 2020

From Chris Nitchie's stage three review of #351 "Add multimedia elements to base": “I'm wondering if we shouldn't recognize WHATWG in the acknowledgements topic, since we basically stole this design from them.”

B Aggregated RFC-2119 statements

This appendix contains all the normative statements from the DITA 2.0 specification. They are aggregated here for convenience in this non-normative appendix.

C Non-normative information

This section contains non-normative information, including topics about new features in DITA 2.0 and migrating to DITA 2.0.

C.1 About the specification source

The DITA specification is authored in DITA. It is a complex document that uses many DITA features, including key references (keyrefs), content references (conrefs), and controlled values set in a subject scheme map.

The source files for the DITA specification are managed in a GitHub repository that is maintained by OASIS; they also can be downloaded from OASIS.

The DITA Technical Committee used the following applications to work with the DITA source:

- DITA Open Toolkit
- <oXygen/> XML Editor and XMetaL Author Enterprise
- DITAweb
- Antenna House Formatter

C.2 Changes from DITA 1.3 to DITA 2.0

C.3 File naming conventions

The DITA OASIS Technical Committee uses certain conventions for the names of XML grammar files. We suggest using these conventions as a best practice to facilitate interchange of grammar files.

Globally unique identifiers

Vocabulary modules that are intended for use outside of a narrowly-restricted context should have one or more associated, globally-unique names by which the modules can be referenced without regard to their local storage location. The globally-unique names can be public identifiers, URNs, or absolute URLs.

Document type shells

Document type shells should be given a name that distinguishes their name, owner, or purpose; for example, `acme-concept.dtd`. The document type shells that are provided by the DITA Technical Committee use the root element of the primary specialization as the basis for the file name.

Module names

Each vocabulary module has a short name that is used to construct entity names and other names that are used in associated declarations. Modules also can have abbreviated names that further shorten the short name, for example "hi-d" for the "highlight" domain, where "software" is the short name and "hi-d" is the abbreviated name.

For structural modules, the module name should be the element type name of the top-level topic or map type defined by the module, such as "concept" or "bookmap".

For element domain modules, the module name should be a name that reflects the subject domain to which the domain applies, such as "highlight" or "software". Domain module names should be sufficiently unique that they are unlikely to conflict with any other domains.

DTD-based specialization modules

Use the following file-naming conventions for DTD-based specialization modules.

Module type	File name (entities)	File name (elements)	Example
Structural	*ModuleName*.ent	*ModuleName*.mod	concept.ent or concept.mod
Element domain	*DomainName*Domain.ent	*DomainName*Domain.mod	highlightDomain.ent or highlightDomain.mod
Attribute domain	*AttributeName*AttDomain.ent	Not applicable	deliveryTargetAttDomain.ent

where:

- *ModuleName* is the name of the element type, such as "concept" or "glossentry".
- *DomainName* is the name of the domain, for example, "highlight" or "utilities".
- *AttributeName* is the name of the specialized attribute, for example, "deliveryTarget".

RELAX NG-based specialization modules

Use the following file-naming conventions for RELAX NG-based specialization modules.

Module type	File name	Example
Structural	*ModuleName*Mod.*rng*	conceptMod.rng
Element domain	*DomainName*DomainMod.rng	highlightDomainMod.rng
Attribute domain	*AttributeName*AttDomain.rng	deliveryTargetAttDomainMod.rng

where:

- *ModuleName* is the name of the element type, such as "concept" or "glossentry".
- *DomainName* is the name of the domain, for example, "highlight" or "utilities".
- *AttributeName* is the name of the specialized attribute, for example, "deliveryTarget".

XSD-based specialization modules

Use the following file-naming conventions for XSD-based specialization modules.

Module	File name	Example
Structural modules: Element groups	*ModuleName*Grp.xsd	conceptGrp.xsd
Structural modules: All	*ModuleName*Mod.xsd	conceptMod.xsd

Module	File name	Example
other declarations		
Domain modules	*DomainName*.xsd	highlightDomain.xsd
Attribute domain	*AttributeName*AttDomain.xsd	deliveryTargetAttDomain.xsd

where:

- *ModuleName* is the name of the element type, such as "concept" or "glossentry".
- *DomainName* is the name of the domain, for example, "highlight" or "utilities".
- *AttributeName* is the name of the specialized attribute, for example, "deliveryTarget".

Constraint modules

Use the following file-naming conventions for constraint modules.

Structural modules

Structural constraint modules should be named using the following format:

DTD	*qualifierTagname*Constraint.mod
RELAX NG	*qualifierTagname*ConstraintMod.rng
XSD	*qualifierTagname*ConstraintMod.xsd

where:

- *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, "Topic".

For example, the file names for the constraint that is applied to the general task to create the strict task are strictTaskbodyConstraint.mod, strictTaskbodyConstraintMod.rng, or strictTaskbodyConstraintMod.xsd.

Domain modules

Domain constraint modules should be named using the following format:

DTD	*qualifierdomain*DomainConstraint.ent
RELAX NG	*qualifierdomain*DomainConstraintMod.rng
XSD	*qualifierdomain*DomainConstraintMod.xsd

where:

- *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".

For example, the file name for a constraint module that removes the syntax diagram from the programming domain might be noSyntaxDiagramProgrammingDomainConstraint.ent.

Because of restrictions on the redefine feature of XML Schema, it is sometimes necessary to use an intermediate level of redefinition, which requires a separate XSD document. In that case, the intermediate XSD document should be named *qualifierdomain*DomainConstraintsInt.xsd.

C.4 Migrating to DITA 2.0

C.5 Considerations for generalizing <foreign> elements

The <foreign> element can contain a mixture of DITA and non-DITA content. Non-DITA content that is contained within a <foreign> element cannot be generalized. However, the <foreign> element itself, as well as any DITA elements that it contains, can be generalized using normal rules.

If a <foreign> element contains non-DITA content, the non-DITA content can be exported to a separate file and replaced in-line with an <object> element. The @data attribute of the <object> element would reference the generated file, and the @type attribute of the <object> element would be set to the value "DITA-foreign".

If an <object> element is present within the <foreign> element during generalization, it is not included with the content that is exported to the separate file. This original <object>element is used to specify alternate content in publishing systems that cannot display the foreign content. It would not be modified except as the ordinary rules of generalization require it.

In the exported file, exported content would be enclosed within a root <foreign> element in order to accommodate the possibility that it might contain several main elements apart from the alternate content.

For easy recognition, the name of the exported file would start with "dita-generalized-", and it is recommended that the file name also contain the topic ID, specialization type, and element ID or generated identifier.

Example: Simple object generalization

For example, a DITA document could contain a specialization of <foreign> for MathML using the OASIS MathML domain. It could look like this:

```
<mathml class="+ topic/foreign mathml-d/mathml ">
  <m:math>
    <m:mi>x</m:mi><m:mo>+</m:mo><m:mn>3</m:mn>
  </m:math>
  <data>X plus three</data>
</mathml>
```

The <mathml> container is a DITA element, so it should be generalized using normal rules. The <m:math> element, which is not a DITA element, will be exported to another file. The <data> element will remain:

```
<foreign class="+ topic/foreign mathml-d/mathml ">
  <object data="dita-generalized-topicid_mathml1.xml" type="DITA-foreign"/>
  <data>X plus three</data>
</foreign>

Contents of dita-generalized-topicid_mathml1.xml:
<foreign class="+ topic/foreign mathml-d/mathml "
         xmlns:m="http://www.w3.org/1998/Math/MathML">
>
  <m:math>
    <m:mi>x</m:mi><m:mo>+</m:mo><m:mn>3</m:mn>
  </m:math>
</foreign>
```

Example: Multiple object generalization

An object might also contain multiple object elements:

```
<mathml class="+ topic/foreign mathml-d/mathml ">
  <m:math>
    <m:mi>x</m:mi><m:mo>+</m:mo><m:mn>3</m:mn>
  </m:math>
  <data>X plus three</data>
  <m:math>
    <m:mi>y</m:mi><m:mo>-</m:mo><m:mn>2</m:mn>
  </m:math>
</mathml>
```

The `<mathml>` container, which is a normal DITA element, should be generalized using normal rules. A file should generated for each set of elements bounded by the container and any existing object elements. In this case, two files will be generated, and two new object elements added to the source.

The modified source:

```
<foreign class="+ topic/foreign mathml-d/mathml ">
  <object data="dita-generalized-topicid_mathml1.xml" type="DITA-foreign"/>
  <data>X plus three</data>
  <object data="dita-generalized-topicid_mathml2.xml" type="DITA-foreign"/>
</foreign>
```

The contents of dita-generalized-topicid_mathml1.xml, the first exported file:

```
<foreign class="+ topic/foreign mathml-d/mathml "
         xmlns:m="http://www.w3.org/1998/Math/MathML">
  <m:math>
    <m:mi>x</m:mi><m:mo>+</m:mo><m:mn>3</m:mn>
  </m:math>
</foreign>
```

The contents of dita-generalized-topicid_mathml2.xml, the second exported file:

```
<foreign class="+ topic/foreign mathml-d/mathml "
         xmlns:m="http://www.w3.org/1998/Math/MathML">
  <m:math>
    <m:mi>y</m:mi><m:mo>-</m:mo><m:mn>2</m:mn>
  </m:math>
</foreign>
```

C.6 Element-by-element recommendations for translators

This topic contains a list of all OASIS DITA elements that are available in the edition. It includes recommendations on how to present the element type to translators, whether the element contents are likely to be suitable for translation, and whether the element has attributes whose values are likely to be suitable for translation. Examples of content that is not suitable for translation include code fragments and mailing addresses.

Since the distinction between block and inline elements is ultimately controlled by the container of the element and the processing associated with it, the same element might be a block in one context and an inline element in another. Specializing document types might vary this behavior according to the needs of the document type being created, and the distinctions given below are provided only as a guide to known behavior with the base DITA document types.

Notes on the tables below

- For specializations, the second column gives the ancestor element, and the third column gives a quick yes/no guide to indicate whether all behavior is inherited. If something is not inherited, the change will appear in bold.
- For any specialization not listed below, the suggested default is to fall back to the closest listed ancestor.
- The block/inline presentation column indicates whether the element is formatted as a single block.
- The block/inline translation column indicates whether the element represents a complete translatable segment. For example, the element `<cmd>` is presented inline with other elements, but represents a complete translation segment.
- Items marked as block*** are blocks on their own, but might appear in the middle of a segment. They should not break the flow of the current segment. These are considered "subflow" elements for translation. We recommend that, when possible, these elements should only be placed at sentence boundaries to aid in translation.
- For all elements, the `@translate` attribute will override the suggested default translation setting. So, a translation setting of "yes" or "no" in the table below does not guarantee that an element will always, or never, be translated.
- If an element has translatable attributes, they are listed in the last column. Note that the `@spectitle` and `@specentry` attributes are described with a footnote.
- The `<keyword>` element (as well as specializations of `<keyword>`) is an inline, phrase-like element when it appears in the body of a document. It can also appear in the `<keywords>` element in `<topicmeta>` (for maps) or in the `<prolog>` (for topic). When it appears in the `<keywords>` element, each `<keyword>` represents an individual segment, and is not part of a larger segment; in that location, `<keyword>` can be considered a "subflow" element.

Topic elements

Element name	Specialized from	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
`<abstract>`	N/A	block	block	yes	
`<alt>`	N/A	block***Footnote.	block	yes	
`<audience>`	N/A	block (metadata)	block	yes	
`<audio>`	N/A	block	block	yes	
`<author>`	N/A	block (metadata)	block	yes	

[1] This element is considered a "subflow" element for translation. If it is located in the middle of a translation segment, it should not be translated as part of that segment. For example, `<indexterm>`, `<fn>`, and `<draft-comment>` might divide a sentence in two, but should be treated as blocks, and should not interrupt the sentence.

[2] The `@spectitle` and `@specentry` attributes can contain translatable text. The direct use of fixed-in-the-DTD text by tools is discouraged, in favor of using the value as a look up string to find the translation outside of the file, using accepted localization methods for generated text.

[3] The block vs. inline designation for the `<foreign>` element is likely to change for some specializations.

[4] The `<desc>`, `<object>`, and `<image>` elements inside `<foreign>` should still be translatable; they provide an alternative display if the foreign content cannot be processed.

Element name	Specialized from	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
<body>	N/A	block	block	yes	
<bodydiv>	N/A	block	block	yes	
<brand>	N/A	block (metadata)	block	yes	
<category>	N/A	block (metadata)	block	yes	
<cite>	N/A	inline	inline	yes	
<colspec>	N/A	n/a	n/a	n/a	
<component>	N/A	block (metadata)	block	yes	
<copyrholder>	N/A	block (metadata)	block	yes	
<copyright>	N/A	block (metadata)	block	yes	
<copyryear>	N/A	block (metadata)	block	yes	
<created>	N/A	block (metadata)	block	yes	
<critdates>	N/A	block (metadata)	block	yes	
<data>	N/A	N/A (metadata)	block	no (likely to change for some specializations)	
<dd>	N/A	block	block	yes	
<ddhd>	N/A	block	block	yes	
<desc>	N/A	block	block	yes	
	N/A	block	block	yes	
<dl>	N/A	block	block	yes	@spectitleFootnote.
<dlentry>	N/A	block	block	yes	
<dlhead>	N/A	block	block	yes	
<draft-comment>	N/A	block***Footnote.	block	**no**	
<dt>	N/A	block	block	yes	
<dthd>	N/A	block	block	yes	
<entry>	N/A	block	block	yes	
<example>	N/A	block	block	yes	@spectitleFootnote.
<fallback>	N/A	block	block	yes	
<featnum>	N/A	block (metadata)	block	yes	
<fig>	N/A	block	block	yes	@spectitleFootnote.

[5] The <desc>, <object>, and <image> elements inside <foreign> should still be translatable; they provide an alternative display if the foreign content cannot be processed.

Element name	Specialized from	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
<figgroup>	N/A	block	block	yes	
<fn>	N/A	block***Footnote.	block	yes	
<foreign> [6]	N/A	blockFootnote.	blockFootnote.	noFootnote.	
<image>	N/A	block when @placement= break, otherwise inline	block when @placement= break, otherwise inline	yes	
<include>	N/A	inline	inline	yes	
<index-see>	N/A	block***Footnote.	block	yes	yes
<index-see-also>	N/A	block***Footnote.	block	yes	yes
<indexterm>	N/A	block***Footnote.	block	yes	
<keytext>	N/A	block	block	yes	
<keyword>	N/A	inline	inline (except when within <keywords> – see note above the table)	yes	
<keywords>	N/A	block	block	yes	
	N/A	block	block	yes	
<lines>	N/A	block	block	yes	@spectitleFootnote.
<link>	N/A	block	block	yes	
<linkinfo>	N/A	block	block	yes	
<linklist>	N/A	block	block	yes	@spectitleFootnote.
<linkpool>	N/A	block	block	yes	
<linktext>	N/A	block	block	yes	
<lq>	N/A	block	block	yes	@reftitle
<media-source>	N/A	block	block	n/a	
<media-track>	N/A	block	block	n/a	
<metadata>	N/A	block (metadata)	block	yes	
<no-topic-nesting>	N/A	n/a	n/a	n/a	

[6] The block vs. inline designation for the <foreign> element is likely to change for some specializations.

Element name	Specialized from	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
<note>	N/A	block	block	yes	@othertype, @spectitleFootnote.
<object>	N/A	block	block	yes	@standby
	N/A	block	block	yes	@spectitleFootnote.
<othermeta>	N/A	block (metadata)	block	yes	@content
<p>	N/A	block	block	yes	
<param>	N/A	block	block	n/a	
<permissions>	N/A	block (metadata)	block	yes	
<ph>	N/A	inline	inline	yes	
<platform>	N/A	block (metadata)	block	yes	
<pre>	N/A	block	block	yes	@spectitleFootnote.
<prodinfo>	N/A	block (metadata)	block	yes	
<prodname>	N/A	block (metadata)	block	yes	
<prognum>	N/A	block (metadata)	block	yes	
<prolog>	N/A	block (metadata)	block	yes	
<publisher>	N/A	block (metadata)	block	yes	
<q>	N/A	inline	inline	yes	
<related-links>	N/A	block	block	yes	
<required-cleanup>	N/A	block***Footnote.	block	**no**	
<resourceid>	N/A	block (metadata)	block	yes	
<revised>	N/A	block (metadata)	block	yes	
<row>	N/A	block	block	yes	
<section>	N/A	block	block	yes	@spectitleFootnote.
<sectiondiv>	N/A	block	block	yes	
<series>	N/A	block (metadata)	block	yes	
<shortdesc>	N/A	block	block	yes	
<simpletable>	N/A	block	block	yes	@spectitleFootnote.
<sl>	N/A	block	block	yes	@spectitleFootnote.
<sli>	N/A	block	block	yes	

Element name	Specialized from	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
<source>	N/A	block (metadata)	block	yes	
<state>	N/A	inline	inline	yes	@value
<stentry>	N/A	block	block	yes	@specentryFootnote.
<sthead>	N/A	block	block	yes	
<strow>	N/A	block	block	yes	
<table>	N/A	block	block	yes	
<tbody>	N/A	block	block	yes	
<term>	N/A	inline	inline	yes	
<text>	N/A	inline	inline	yes	
<tgroup>	N/A	block	block	yes	
<thead>	N/A	block	block	yes	
<title>	N/A	block	block	yes	
<titlealt>	N/A	block	block	yes	
<tm>	N/A	inline	inline	yes	
<topic>	N/A	block	block	yes	
	N/A	block	block	yes	@spectitleFootnote.
<unknown>	N/A	block	block	no	
<video>	N/A	block	block	yes	
<vrm>	N/A	block (metadata)	block	yes	
<vrmlist>	N/A	block (metadata)	block	yes	
<xref>	N/A	inline	inline	yes	

Map elements

Element name	Specialized from	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
<anchor>	N/A	n/a	n/a	n/a	
<map>	N/A	block	block	yes	
<navref>	N/A	n/a	n/a	n/a	
<relcell>	N/A	block	block	yes	
<relcolspec>	N/A	block	block	yes	
<relheader>	N/A	block	block	yes	
<relrow>	N/A	block	block	yes	
<reltable>	N/A	block	block	yes	

Element name	Specialized from	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
<shortdesc>	N/A	block	block	yes	
<topicmeta>	N/A	block	block	yes	
<topicref>	N/A	block	block	yes	
<ux-window>	N/A	N/A (empty)	N/A (empty)	no	

Alternative Title Elements

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
<linktitle>	<titlealt>	yes	N/A (metadata)	block	yes	
<navtitle>	<titlealt>	yes	N/A (metadata)	block	yes	
<searchtitle>	<titlealt>	yes	N/A (metadata)	block	yes	
<subtitle>	<titlealt>	yes	block	block	yes	
<titlehint>	<titlealt>	yes	N/A (metadata)	block	yes	

Emphasis domain elements (emphasis-d)

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
	<ph>	yes	inline	inline	yes	
		yes	inline	inline	yes	

Hazard statement domain (hazard-d elements)

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
<consequence>		yes	block	block	yes	
<hazardstatement>	<note>	yes	block	block	yes	@othertype, @spectitle Footnote.
<hazardsymbol>	<image>	yes	block when @placement= break, otherwise inline	block when @placement= break, otherwise inline	yes	
<howtoavoid>		yes	block	block	yes	

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
<messagepanel>		yes	block	block	yes	@spectitle Footnote.
<typeofhazard>		yes	block	block	yes	

Highlight domain elements (hi-d)

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
<b>	<ph>	yes	inline	inline	yes	
<line-through>		yes	inline	inline	yes	
<i>	<ph>	yes	inline	inline	yes	
<overline>	<ph>	yes	inline	inline	yes	
<sub>	<ph>	yes	inline	inline	yes	
<sup>	<ph>	yes	inline	inline	yes	
<tt>	<ph>	yes	inline	inline	yes	
<u>	<ph>	yes	inline	inline	yes	

Utilities domain elements

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
<area>	<figgroup>	yes	block	block	yes	
<coords>	<ph>	**NO**	inline	inline	**no**	
<imagemap>	<fig>	yes	block	block	yes (can contain translatable alternate text)	@spectitle Footnote.
<shape>	<keyword>	**NO**	inline	inline	**no**	
<sort-as>	<data>	**NO**	block***Footnote .	block	yes	

Classification domain elements (classify-d)

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
<subjectCell>	<relcell>	yes	block	block	yes	
<subjectref>	<topicref>	yes	block	block	yes	
<topicapply>	<topicref>	yes	block	block	yes	
<topicCell>	<relcell>	yes	block	block	yes	
<topicsubject>	<topicref>	yes	block	block	yes	
<topicSubjectHeader>	<relrow>	yes	block	block	yes	
<topicSubjectRow>	<relrow>	yes	block	block	yes	
<topicSubjectTable>	<reltable>	yes	block	block	yes	

DITAVALref domain elements

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
<ditavalmeta>	<topicmeta>	yes	block	block	yes	
<ditavalref>	<topicref>	yes	block	block	yes	
<dvrKeyscopePrefix>	<data>	yes	N/A (metadata)	block	no	
<dvrKeyscopeSuffix>	<data>	yes	N/A (metadata)	block	no	
<dvrResourcePrefix>	<data>	yes	N/A (metadata)	block	no	
<dvrResourceSuffix>	<data>	yes	N/A (metadata)	block	no	

Map group domain elements (mapgroup-d)

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
<anchorref>	<topicref>	yes	block	block	yes	
<keydef>	<topicref>	yes	block	block	yes	
<mapref>	<topicref>	yes	block	block	yes	
<mapresources>	<topicref>	yes	block	block	yes	
<topicgroup>	<topicref>	yes	block	block	yes	

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
<topichead>	<topicref>	yes	block	block	yes	

DITAVAL elements

The DITAVAL elements are not specialized, and are not rendered on their own, so related columns are dropped from this table. There are no translatable attributes in the DITAVAL element set.

The only element that directly contains text for translation is <alt-text>.

Element name	Block/Inline (translation)	Translatable content?
<alt-text>	block	yes
<endflag>	block	yes (inside nested elements)
<prop>	block	yes (inside nested elements)
<revprop>	block	yes (inside nested elements)
<startflag>	block	yes (inside nested elements)
<style-conflict>	block	N/A *(empty element)*
<val>	block	yes (inside nested elements)

C.7 Formatting expectations

DITA is a standard that supports the creation of human-readable content. Accordingly, DITA defines fundamental document components. Since there is a reasonable expectation that such document components be rendered consistently, we suggest the following formatting conventions.

Table 8: Formatting expectations for DITA elements

Element	Suggested formatting
<b>	Apply bold highlighting to the contents of the <b> element.
<cite>	Set citations apart from the surrounding text by a form of highlighting, for example, italics.
<dd>	See <dl>.
<dl>	Apply the following conventions: • The term (<dt>) is against the starting margin of the page or column. • The description or definition (<dd>) is either indented and on the next line or on the same line after the term. • The <dlhead> looks like a table heading row.
<dlhead>	See <dl>.
<dt>	See <dl>.
<i>	For Western languages, apply italic highlighting to the contents of the <i> element.

Element	Suggested formatting
	Apply the following conventions: • In ordered lists, list items are indicated by numbers or alphabetical characters. • In unordered lists, list items are indicated by bullets or dashes.
<lines>	Render the contents of <lines> elements in a non-monospaced font.
<line-through>	Render the contents of the <line-through> element with a line struck through.
<lq>	Render the contents of the <lq> element as an indented block.
<note>	Render a label for notes. The content of the label depends on the values of the @type attribute. A note typically is formatted in a way that stands out from the surrounding content.
	See .
<overline>	Render a line above the contents of the <overline> element.
<pre>	Render the content of a <pre> element in a monospaced font.
<sl>	See <sli>.
<sli>	Apply the following conventions: • The content of each simple list item is placed on a separate line. • The lines are not distinguished by numbers, bullets, or other icons.
<sub>	Render the contents of the <sub> element lower in relationship to the surrounding text and in a smaller font.
<sup>	Render the contents of the <sup> element higher in relationship to the surrounding text and in a smaller font.
<tt>	Render the contents of the <tt> element in a monospaced font.
<u>	Apply underlining to the contents of the <u> element.

C.8 DTD public identifiers

Each document-type shell (`.dtd` file) or module component (`.mod` or `.ent` file) has a public identifier. The public identifier can reference either the latest version or a specific version of the document-type shell or module component.

The public identifiers for the DTD files that are maintained by OASIS use the following format:

```
"-//OASIS//DTD DITA version information-type//EN"
```

where:

- *version* either is the DITA specific version number (for example, 2.0, 1.3, or 1.2), 2.x (representing the latest version of DITA 2.x), or 1.x (representing 1.3, which is the final version of 1.x). Ommitting the version number entirely is also equivalent to the final release of DITA 1.x.
- *information-type* is the name of the topic or map type, for example, Concept or BookMap.

Note that "OASIS" is the owner identifier; this indicates that the artifacts are owned by OASIS. The keyword "DITA" is a convention that indicates that the artifact is DITA-related.

C.9 Domains and constraints in the OASIS specification

This section provides a summary of the domains and constraints that are available as part of the OASIS specification, as well as a summary of how they are used.

C.9.1 Domains and constraints in the OASIS specification

OASIS distributes grammar files for a set of domains and constraints.

A designation of (map) after the domain name indicates that the domain only specializes map elements; a designation of (topic) indicates that the domain specializes elements that are only available in topic or that it can only be used in topics. A designation of (map/topic) indicates that the domain specializes elements that are common to both maps and topics, so could be used in either even if it is generally intended for one or the other. Attribute domains can always be used in both topics and maps.

Table 9: Base domains

Domain	Description	Short name
Classify (map)	For associating content in a map with subjects in a subject scheme.	classify-d
`@deliveryTarget` attribute	Attribute for filtering based on delivery target.	N/A
DITAVALref (map)	For filtering a branch of a map.	ditavalref-d
Hazard statements (map/topic)	For providing detailed information about safety hazards.	hazard-d
Highlighting (map/topic)	For highlighting when the appropriate semantic element does not exist yet.	hi-d
Indexing (map/topic)	For extended indexing functions such as see and see-also.	indexing-d
Map group (map)	Utility elements for use in maps.	mapgroup-d
Utilities (map/topic)	For providing image maps, sort keys, and other useful structures.	ut-d

C.9.2 Base domains: Where they are used

This section provides a summary of which document types use each of the base OASIS domains.

Domain	What includes it	What does NOT include it
Classify (map)		• Base map, base topic
`@deliveryTarget` attribute (base)	• Base map, base topic	• N/A
DITAVALref (map)	• Base map	• Base topic
Hazard statement (map/topic)	• Base map, base topic	
Highlighting (map/topic)	• Base map, base topic	• N/A
Indexing (map/topic)	• Base map, base topic	
Map group (map)	• Base map	• Base topic

Domain	What includes it	What does NOT include it
Utilities (map/topic)	• Base map, base topic	• N/A

C.9.3 Base document types: Included domains

This topic provides a summary of which domains are used in each of the base document types.

Table 10: Domain usage in base document types

Document type	Includes these domains	Does not include
base map	• `@deliveryTarget` attribute • Hazard statement (map/topic) • Highlighting (map/topic) • Indexing (map/topic) • Utilities (map/topic) • DITAVALref (map) • Map group (map)	
base topic	• `@deliveryTarget` attribute • Hazard statement (map/topic) • Highlighting (map/topic) • Indexing (map/topic) • Utilities (map/topic)	• Base domains – DITAVALref (map) – Map group (map)

C.10 Processing interoperability considerations

The DITA specification does not require processors to perform filtering, content reference resolution, key space construction, and other processing related to base DITA semantics in any particular order. This means that different conforming DITA processors might produce different results for the same initial data set and filtering conditions. DITA users and DITA implementers need to be aware of these potential differences in behavior when DITA content will be processed by different processors.

In general, in any situation in which two elements interact during processing, applying filtering before or after the processing is done can result in different results when either or both of the elements is conditional.

For conditional elements, an element is "applicable" if it is filtered in and "inapplicable" if it is filtered out.

Filtering and content reference resolution

When two elements are merged as result of a content reference, the attributes of the two elements are combined. By default, the attributes of the referencing element take precedence over the referenced element. However, any attribute can specify the value "-dita-use-conref-target", which causes the referenced element attribute to take precedence. This means that the effective value of filtering attributes might reflect either the referencing element or the referenced element depending on how each attribute is configured on the referencing element. This in turn means that, in certain cases, filtering before resolving content references will produce a different result than when filtering is applied after resolving content references.

In two cases, the order in which filtering is applied results in either an element being in the effective result or an element not being in the effective result. There is a third case in which there will be either an empty element (and unresolvable content reference) or no element.

In the case where a referenced element is not applicable and the referencing element is explicitly applicable for the same condition (that is, both elements specify values for the same filtering attribute and the referencing element is applicable), if content references are resolved *before* filtering, the content reference is resolved and the effective value of the referencing element reflects the referenced element. If content referencing is resolved *after* filtering, the referenced element is filtered out and the content reference cannot be resolved, typically generating an error.

The same scenario results in different results for the case of conref push. An applicable, referencing element can use conref push to replace another element that would otherwise be filtered out. If content references are resolved *before* filtering, the content is pushed and the effective value of the referenced element reflects the referencing element. If content referencing is resolved *after* filtering, the referenced element will be filtered out and the content reference can no longer be resolved.

If the referencing element is not conditional and the referenced element is inapplicable, filtering applied before content reference resolution results in an unresolvable content reference. If filtering is applied after content resolution, the explicit condition on the referenced element becomes the effective value for that condition following content resolution and the result is then filtered out. The difference in these two cases is that in the first case the content reference cannot be resolved, resulting in a processing error and a potentially nonsensical element if the referencing element has required subelements (for example, a content reference from a topic to another topic, where the referencing topic must have a title subelement), but in the second case the element is filtered completely out.

Different processing orders might also provide different results in the case where pushed content is wrapped in an element that is filtered out. If filtering is applied before content resolution, that entire block of content (the wrapper and the content to be pushed) is filtered out before the content reference is resolved. If filtering is applied after content resolution, the push action will be resolved first so that content appears in the referenced location, after which the referencing element (along with its wrapper) is filtered from the original source location.

Filtering and key space resolution

See Keys and conditional processing (84) for a discussion of effective key definitions and conditional processing.

As an implementation detail for key-space-constructing processors, if filtering is applied before constructing the key space, then the set of effective key definitions is simply the first definition of each unique key name. However, if filtering is applied after key space construction, and in particular, if a processor needs to allow dynamic resolution of keys based on different filtering specifications applied to the same constructed key space, then the set of effective key definitions is the first definition of each pair of unique key name and unique selection specification set. This second form of constructed key space would be needed by processors such as editors and content management systems that need to quickly provide different filtering-specific key bindings without reconstructing the entire key space for each new set of filtering conditions.

For example, given a map that contains two definitions for the key "topic-01", one with an `@audience` value of "expert" and one with an `@audience` value of "novice", a filter-first processor would only have at most one effective key definition for the key name "topic-01", whichever of the two definitions was filtered in by the active filter specification and was the first definition encountered (if both happen to be filtered in). In a processor that supports dynamic key definition filtering, there would be two effective definitions for the key name "topic-01", one for `@audience` of "expert" and one for `@audience` of "novice". The processor would also need to maintain knowledge of the definition order of the two key definitions in order

to correctly handle the case when both "expert" and "novice" are applicable for a given key access request (in which case, whichever of the two definitions was first would be used as the effective value of the key).

Link resolution

If a cross reference, link, or other linking element is resolved to its target before filtering and the target is subsequently filtered out, the link would be to a non-existent target but might reflect properties of the target (for example, a cross reference link text might reflect the target title). If the link is resolved after filtering is applied and the target is filtered out, the link is to a non-existent target, which will result in a different link text. The rendition effect for the navigation link is the same: the link cannot be navigated because the target does not exist in the rendered result.

Topicref resolution

Resolution of `<topicref>` elements before filtering can result in use of topic-provided navigation titles or metadata that would not be used if the target topic was filtered out before resolution. In both cases, the topicref as rendered would be to a missing topic.

Copy-to processing

If copy-to processing is done before filtering, two `<topicref>` elements, only one of which is applicable, could specify the same `@copy-to` target, leading to a conflict and a potential ambiguity about which governs. If the `<topicref>` elements are filtered before `@copy-to` processing, the conflict does not occur.

C.11 Specialization design, customization, and the limits of specialization

DITA specialization imposes certain restrictions. An inherent challenge in designing DITA vocabulary modules and document types is understanding how to satisfy markup requirements within those restrictions and, when the requirements cannot be met by a design that fully conforms to the DITA architecture, how to create customized document types that diverge from the DITA standard as little as possible.

DITA imposes the following structural restrictions:

- All topics must have titles.
- Topic body content must be contained within a body element.
- Section elements cannot nest.
- Metadata specific to an element type must be represented using elements, not attributes.

When markup requirements cannot be met within the DITA architecture, there still might be an interest in using DITA features and technology, or a business need for interoperability with conforming DITA documents and processors. In this case, the solution is to create *customized document types*. Customized document types are document types that do not conform to the DITA standard. To reduce the cost of producing conforming documents from non-conforming documents, custom document types should minimize the extent to which they diverge from the DITA standard.

Typical reasons for considering custom document types include the following:

- Optimizing markup for authoring
- Supporting legacy markup structures that are not consistent with DITA structural rules, for example, footnotes within titles

- Defining different forms of existing structures, such as lists, where the DITA-defined structures are too constrained
- Providing attributes required by specific processors, such as CMS-defined attributes for maintaining management metadata
- Embedding tool-imposed markup in places that do not allow the `<foreign>` or `<unknown>` elements

Remember that customized document types do not conform to the DITA standard, and thus are not considered DITA. In many of the cases above, it is possible to define document types that conform to the DITA standard. Explore this fully before developing customized document types.

Optimizing document types

Conforming DITA grammar files are modular, which facilitates exchange of vocabulary modules and constraints and simplifies the process of assembling document type shells. In some cases there might be a reason to avoid the modular approach and use an optimized document type composed of a single file (or a smaller number of files). For example, this could be advantageous in situations where validation occurs over a network.

In an optimized DTD, entities might also be resolved to further optimize processing or validation. This could speed up processing for environments that process and validate large numbers of DITA maps and topics.

An optimized document type will still allow for the creation of conforming DITA content that follows all other rules in the DITA specification. In these cases it is still possible to create a document type that conforms completely to standard DITA coding practices. Maintaining a conforming copy ensures that the optimized document type is still conforming to the standard, and might also ease interchange with tools that expect conforming document types.

Creating custom document types for non-standard markup

When the relaxed content models for DITA elements are inappropriately open for authoring purposes, document type shells can remove undesirable domains or use constraint modules to restrict content models. If content models are not relaxed enough, and markup requirements include content models that are *less* constrained than those defined by DITA, custom document types might be the only option.

Customized document types do not conform to the DITA standard. Preprocessing can ensure compatibility with existing publishing processes, but it does not ensure compatibility with DITA-supporting authoring tools or content management systems. However, when an implementation is being heavily customized, a customized document type can help isolate and control the consequences of non-standard design.

For example, if an authoring group requires the `<p>` element to be spelled out as `<paragraph>`, the document type could be customized to change `<p>` to `<paragraph>` for authoring purposes. Such documents then could be preprocessed to rename `<paragraph>` back to `<p>` before they are fed into a standard DITA publishing process.

Because DITA document types are designed to enable constraints, such customized documents can still take advantage of existing override schemes. While still not valid DITA, a document type shell could be set up that implements local requirements (such as adding global CMS-defined attributes), and then imports an otherwise valid document type shell. This helps isolate non-compliant portions of the document type, while reusing as much as possible of the original DITA grammar.

Specialization design considerations

Requirements for new markup often appear to be incompatible with DITA architectural rules or existing markup, especially when mapping existing non-DITA markup practice to DITA, where the existing markup might have used structures that cannot be directly expressed in DITA. For example, you might need markup for a specialized form of list where the details are not consistent with the base model for DITA lists.

In this case you have two alternatives, one that conforms to DITA and one that does not.

- Specialize from more generic base elements or attributes.
- Define non-conforming structures and map them to conforming DITA structures as necessary for processing by DITA-aware processors or for interchange as conforming DITA documents.

Specializing from more generic base elements, such as defining a list using specializations of `<ph>` or `<div>`, while technically conforming, might still impede interchange of such documents. Generic DITA processors will have no way of knowing that what they see as a sequence of phrases or divisions is really a list and should be rendered in a manner similar to standard DITA lists. However, your documents will be reliably interchangeable with conforming DITA systems.

Defining non-conforming markup structures means that the resulting documents are not conforming DITA documents. They cannot be reliably processed by generic DITA-aware processors or interchanged with other DITA systems. However, as long as the documents can be transformed into conforming DITA documents without undue effort, interchange and interoperability requirements can be satisfied as needed. For example, a content management system could add its own required markup for management metadata, but strip the metadata when delivering content to conforming DITA processors.

Note that even if one uses the DITA-defined types as a starting point, any change to those base types not accomplished through specialization or the constraint feature defines a completely new document type that has no normative relationship to the DITA document types, and cannot be considered in any way to be a conforming DITA application. In particular, the use of DITA specialization from non-DITA base types does not produce DITA-conforming vocabularies.

Specialize from generic elements or attributes

Most DITA element types have relaxed content models that are specifically designed to allow a wide set of options when specializing from them. However, some DITA element types do impose limits that might not be acceptable or appropriate for a specific markup application. In this case, consider specializing from a more generic base element or attribute.

Generic elements are available in DITA at every level of detail, from whole topics down to individual keywords, and the generic `@base` attribute is available for attribute domain specialization.

For example, if you want to create a new kind of list but cannot usefully do so specializing from `<ul>`, `<ol>`, `<sl>`, or `<dl>`, you can create a new set of list elements by specializing nested `<div>` elements. This new list structure will require specialized processing to generate appropriate output styling, because it is not semantically tied to the other lists by ancestry. Nevertheless, it will remain a valid DITA specialization, with the standard support for generalization, content referencing, conditional processing, and more.

The following base elements in `<topic>` are generic enough to support almost any structurally-valid DITA specialization:

<topic>
: Any content unit that has a title and associated content

<section>
: Any non-nesting division of content within a topic, titled or not

<p>
Any non-nesting non-titled block of content below the section level

<fig>
Any titled block of content below the section level

<ul>, <ol>, <dl>, <sl>, <simpletable>
Any structured block of content that consists of listed items in one or more columns

<ph>
Any division of content below the paragraph level

<text>
Text within a phrase

<keyword>
Any non-nesting division of content below the paragraph level

<data>
Any content that acts as metadata rather than core topic or map content

<foreign>
Any content that already has a non-DITA markup standard, but still needs to be authored as part of the DITA document. Processors should attempt to render this element, if at all possible.

<unknown>
Any non-standard markup that does not fit the DITA model, but needs to be managed as part of a DITA document. Processors should not attempt to render this element.

<bodydiv>
A generic, untitled, nestable container for content within topic bodies

<sectiondiv>
A generic, untitled, nestable container for content within sections

** **
A generic, untitled, nestable container for content within topic bodies or sections

The following attributes in topic are suitable for domain specialization to provide new attributes that are required throughout a document type:

@props
Any new conditional processing attribute

@base
Any new attribute that is universally available, has a simple syntax (space-delimited alphanumeric values), and does not already have a semantic equivalent

Whenever possible, specialize from the element or attribute that is the closest semantic match.

D Revision history

The following table contains information about revisions to this document.

Revision	Date	Editor	Description of changes
01	10 May 2019	Kristen James Eberlein	Generated working draft #01. Contains updated TOC that reduces level of topic nesting.
02	17 May 2019	Kristen James Eberlein	Generated working draft #02. Contains reworked markup and styling for RFC-2119 statements.
03	24 May 2019	Kristen James Eberlein	Generated working draft #03. Contains a non-normative appendix of all conformance statements (generated with prototype code).
04	03 June 2019	Kristen James Eberlein	Generated working draft #04. Draft comments in "Attribute generalization" and "<lines>" topics resolved per TC call on 28 May 2019. TOC reorganized to move purely illustrative content from "DITA markup" and into "Overview of DITA".
05	05 July 2019	Kristen James Eberlein	Generated working draft #05. <topicgroup> and <topichead> topics reworked per TC call on 02 July 2019. Renamed "DITA markup" to "DITA processing". Added "Chunking" content and a new "DITA maps and their usage" topic. The new topic contains a list of material to cover.
06	15 July 2019	Kristen James Eberlein	Generated working draft #06 Conditional processing applied to element and attribute topics shared with LwDITA. Attribute definitions listed in alphabetical order. Attribute values tagged with <keyword> and rendered styled with quotation marks.
07	02 August 2019	Kristen James Eberlein	Generated working draft #07. Includes rework of indexing content
08	05 August 2019	Kristen James Eberlein	Generated working draft #08. Includes completed edits of multimedia domain topics (based on DITAweb review comments).

Revision	Date	Editor	Description of changes
09	13 August 2019	Kristen James Eberlein	Generated working draft #09. Contains revised indexing content, based on reviews by Robert Anderson, Stan Doherty, Eliot Kimber, and Joyce Lam. Added (as yet incomplete) "DITA attributes, A to Z" topic.
10	20 August 20201	Kristen James Eberlein	Generated working draft #10. Includes additional edits to indexing content, plus new placeholder topic about effective attribute values.
11	27 August 2019	Kristen James Eberlein	Generated working draft #11. Includes fully implemented issue #253: Remove indexing domain.
12	29 August 2020	Kristen James Eberlein	Generated working draft #12. Includes bug fixes, spelling corrections, changes in organizational affiliation, updates to acknowledgments, and implementation of the following DITA 2.0 proposals: • #29 Add `<mapresources>` • #34 Remove `<topicset>` and `<topicsetref>` • #217 Remove `@domains` attribute • #258: Add `@outputclass` to DITAVAL • #277: Change specialization base of `<imagemap>` • #278 Remove `@lockmeta` • #292 Add attributes and `<title>` to `<simpletable>` • #297 Allow `<example>` in more places

Index

Special Characters

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

X