Darwin Information Typing Architecture (DITA) Version 1.3 Part 1: Base Edition

OASIS Standard

17 December 2015

Specification URIs

This version:

http://docs.oasis-open.org/dita/dita/v1.3/os/part1-base/dita-v1.3-os-part1-base.html (Authoritative version)

http://docs.oasis-open.org/dita/dita/v1.3/os/part1-base/dita-v1.3-os-part1-base.pdf

Previous version:

http://docs.oasis-open.org/dita/dita/v1.3/cos01/part1-base/dita-v1.3-cos01-part1-base.html (Authoritative version)

http://docs.oasis-open.org/dita/dita/v1.3/cos01/part1-base/dita-v1.3-cos01-part1-base.pdf

Latest version:

http://docs.oasis-open.org/dita/dita/v1.3/dita-v1.3-part1-base.html (Authoritative version)

http://docs.oasis-open.org/dita/dita/v1.3/dita-v1.3-part1-base.pdf

Technical Committee:

OASIS Darwin Information Typing Architecture (DITA) TC

Chair:

Kristen James Eberlein (kris@eberleinconsulting.com), Eberlein Consulting LLC

Editors:

Robert D. Anderson (robander@us.ibm.com), IBM

Kristen James Eberlein (kris@eberleinconsulting.com), Eberlein Consulting LLC

Additional artifacts:

This prose specification is one component of a work product that also includes:

- Darwin Information Typing Architecture (DITA) Part 0: Overview. http://docs.oasis-open.org/dita/dita/v1.3/os/part0-overview/dita-v1.3-os-part0-overview.html.
- Darwin Information Typing Architecture (DITA) Part 1: Base Edition (this document). http://docs.oasis-open.org/dita/dita/v1.3/os/part1-base/dita-v1.3-os-part1-base.html. This edition contains topic and map; it is designed for implementers and users who need only the most fundamental pieces of the DITA framework.
- Darwin Information Typing Architecture (DITA) Part 2: Technical Content Edition. http://docs.oasis-open.org/dita/dita/v1.3/os/part2-tech-content/dita-v1.3-os-part2-tech-content.html. This edition contains the base architecture plus the technical-content specializations; it is designed for authors who use information typing and document complex applications and devices.
- Darwin Information Typing Architecture (DITA) Part 3: All-Inclusive Edition. http://docs.oasis-open.org/dita/dita/v1.3/os/part3-all-inclusive/dita-v1.3-os-part3-all-inclusive.html. This edition contains the base

architecture, technical content, and the learning and training specializations. It is designed for implementers who want all OASIS-approved specializations, as well as users who develop learning and training materials.

- ZIP file that contains a CHM version of the written specification for this part. http://docs.oasis-open.org/dita/dita/v1.3/os/part1-base/dita-v1.3-os-part1-base-chm.zip

- ZIP file that contains an HTML version of the written specification for this part. http://docs.oasis-open.org/dita/dita/v1.3/os/part1-base/dita-v1.3-os-part1-base-html.zip

- ZIP file that contains the grammar files for this part. http://docs.oasis-open.org/dita/dita/v1.3/os/part1-base/dita-v1.3-os-part1-base-grammars.zip

- ZIP file that contains the DITA source for this part. http://docs.oasis-open.org/dita/dita/v1.3/os/part1-base/dita-v1.3-os-part1-base-dita.zip

For more information about the editions and what they contain, see Editions.

Abstract:

The Darwin Information Typing Architecture (DITA) 1.3 specification defines both a) a set of document types for authoring and organizing topic-oriented information; and b) a set of mechanisms for combining, extending, and constraining document types.

Status:

This document was last revised or approved by the OASIS Darwin Information Typing Architecture (DITA) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dita#technical.

TC members should send comments on this specification to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at https://www.oasis-open.org/committees/comments/index.php?wg_abbrev=dita.

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).

Citation format:

When referencing this specification the following citation format should be used:

[DITA-v1.3-part1-base]

Darwin Information Typing Architecture (DITA) Version 1.3 Part 1: Base Edition. Edited by Robert D. Anderson and Kristen James Eberlein. 17 December 2015. OASIS Standard. http://docs.oasis-open.org/dita/dita/v1.3/os/part1-base/dita-v1.3-os-part1-base.html. Latest version: http://docs.oasis-open.org/dita/dita/v1.3/dita-v1.3-part1-base.html.

Notices

Copyright © OASIS Open 2015. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full *Policy* may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of *OASIS*, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see *https://www.oasis-open.org/policies-guidelines/trademark* for above guidance.

Table of contents

1 Introduction to DITA 1.3

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

1.1 About the DITA specification: Base edition

The base edition of the DITA specification is the smallest edition. It is designed for application developers and users who need only the most fundamental pieces of the DITA framework.

Base edition

The following graphic illustrates the contents of the base edition; it also highlights how the base edition relates to the other editions.
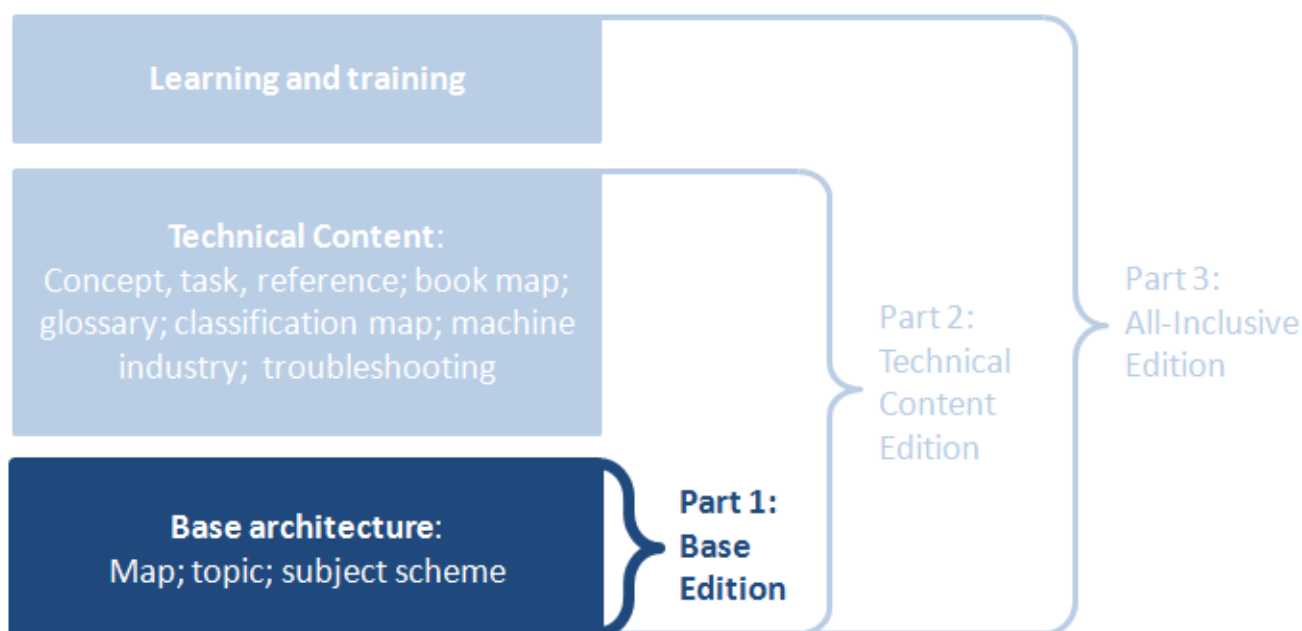
Base edition (this edition)

The base edition contains topic, map, and subject scheme map. It is the smallest edition; it is designed for application developers and users who need only the most fundamental pieces of the DITA framework.

Technical content edition

The technical content edition includes the base architecture **and** the specializations usually used by technical communicators: concept, task, and reference topics; machine industry task; troubleshooting topic; bookmap; glossaries; and classification map. It is the medium-sized edition; it is designed for authors who use information typing and document complex applications and devices, such as software, hardware, medical devices, machinery, and more.

All-inclusive edition

The all-inclusive edition contains the base architecture, the technical content pieces, **and** the learning and training specializations. It is the largest edition; it is designed for implementers who want all OASIS-approved specializations, as well as users who develop learning and training materials.

XML grammar files

The DITA markup for DITA vocabulary modules and DITA document types is available in several XML languages: RELAX NG (RNG), XML Document-Type Definitions (DTD), and W3C XML Schema (XSD).

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

DITA written specification

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

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

The DITA written specification is available in the following formats; the XHTML version is authoritative:

- XHTML (available from the OASIS Web site)
- CHM
- PDF
- DITA source
- ZIP of XHTML (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]**.

MUST

This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.

MUST NOT

This phrase, or the phrase "SHALL NOT", means that the definition is an absolute prohibition of the specification.

SHOULD

This word, or the adjective "RECOMMENDED", means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

SHOULD NOT

This phrase, or the phrase "NOT RECOMMENDED", means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

MAY

This word, or the adjective "OPTIONAL", means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option must be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option must be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides).

1.3 Normative references

[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*>.

[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/ .

1.4 Non-normative references

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

[ciq-v3.0]

OASIS Committee Specification 02, *Customer Information Quality Specifications Version 3.0. Name (xNL), Address (xAL), Name and Address (xNAL) and Party (xPIL)*, *http://www.oasis-open.org/committees/download.php/29877/ OASIS%20CIQ%20V3.0%20CS02.zip*, 20 September 2008.

[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.

[MathML 3.0]

Mathematical Markup Language (MathML) Version 3.0 2nd Edition, D. Carlisle, P. D. F. Ion, R. Miner, Editors, W3C Recommendation, 10 April 2014, http://www.w3.org/TR/2014/REC-MathML3-20140410/ . *Latest version* available at http://www.w3.org/TR/MathML3/ .

[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.

[SVG 1.1]

Scalable Vector Graphics (SVG) 1.1 (Second Edition), E. Dahlström, P. Dengler, A. Grasso, C. Lilley, C. McCormack, D. Schepers, J. Watt, J. Ferraiolo, J. Fujisawa, D. Jackson, Editors, W3C Recommendation, 16 August 2011, http://www.w3.org/TR/2011/REC-SVG11-20110816/ . *Latest version* available at http://www.w3.org/TR/SVG11/ .

[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.

[XSD 1.0 Structures]

XML Schema Part 1: Structures Second Edition, H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn, Editors, W3C Recommendation, 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/ . *Latest version* available at http://www.w3.org/TR/xmlschema-1/ .

[XSD 1.0 Datatypes]

XML Schema Part 2: Datatypes Second Edition, P. V. Biron, A. Malhotra, Editors, W3C Recommendation, 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/ . *Latest version* available at http://www.w3.org/TR/xmlschema-2/ .

[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.

1.5 Formatting conventions in the XHTML version of the specification

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

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 are not normative; they contain the content of the `<shortdesc>` element for the child topic, which is rendered in a normative context as the first paragraph of the topic; the content is identical in both renditions. 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 XHTML version of the specification:
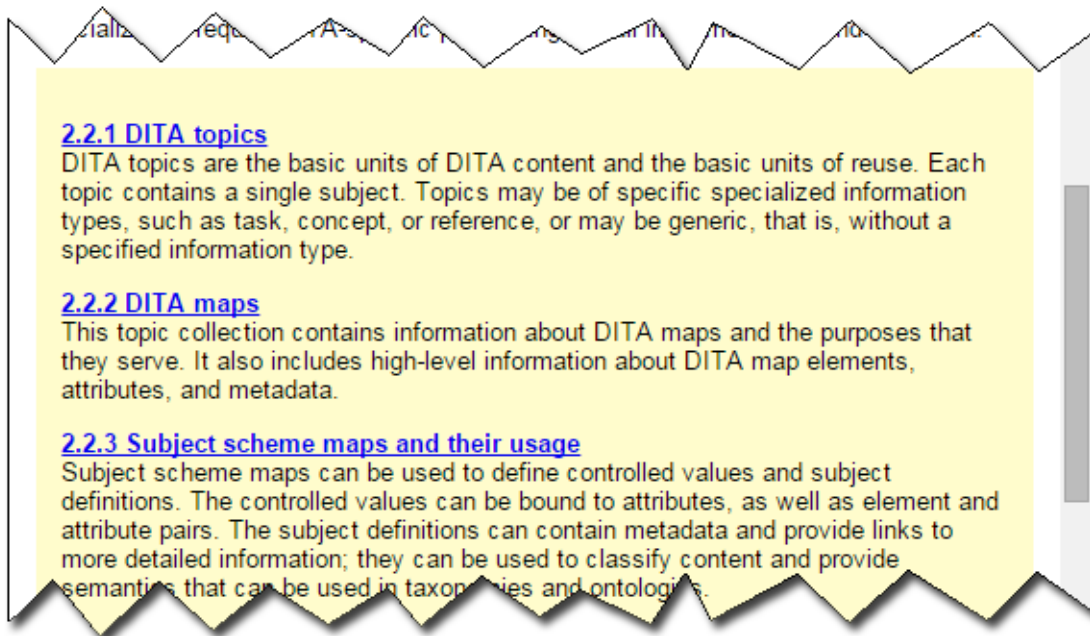
Figure 1: Link previews

Navigation links

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

Parent topic

Takes readers to the parent topic, which 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 XHTML version of the specification:
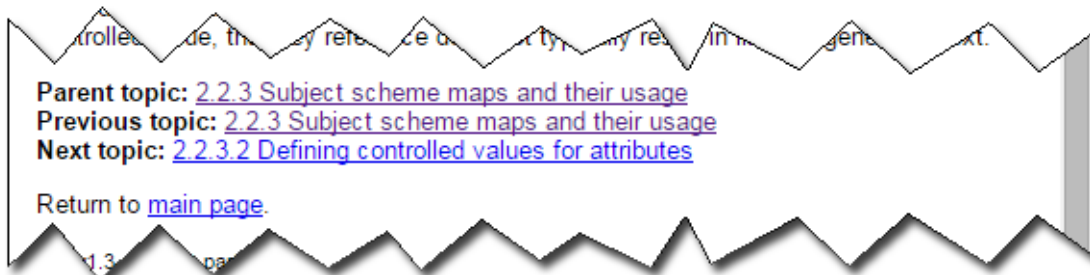
Figure 2: Navigation links

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

2 Architectural Specification: Base edition

The architectural specification portion of the DITA specification outlines the framework of DITA. It contains an overview of DITA markup; addressing; processing; configuration, specialization, generalization, and constraints; as well as information about coding DITA grammar files.

2.1 Introduction to 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.

2.1.1 DITA terminology and notation

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

Notation

The following conventions are used throughout the specification:

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.

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 should be considered 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.

Basic DITA terminology

The following terminology is used to discuss basic DITA concepts:

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, XSD, 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.

Specialization terminology

The following 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.

DITA modules

The following 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.

Linking and addressing terms

The following terminology is used to discuss linking and addressing terms:

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.

Terminology related to keys

The following 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 *Using keys for addressing (69)* 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.

Map terms

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.

2.1.2 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. See *DITA topics (26)* 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. DITA maps *SHOULD* have `.ditamap` as the file extension. See *DITA maps (31)* 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 ...? 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 (28)* 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 (62)* 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 (90)* 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 (95)* 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 constraint modules. In addition, a document type shell specifies whether and how topics can nest. See *Configuration (124)* 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 (126)* 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 (139)* for more information.

2.1.3 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`

2.1.4 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.

2.2 DITA markup

Topics and maps are the basic building blocks of the Darwin Information Typing Architecture (DITA). Metadata attributes and values can be added to DITA topics and maps, as well as to elements within topics, to allow for conditional publishing and content reuse.

DITA topics and maps are XML documents that conform to the XML specification. As such, they can be viewed, edited, validated, and processed with standard XML tools, although some DITA-specific features, such as content reference, key reference, and specialization require DITA-specific processing for full implementation and validation.

2.2.1 DITA topics

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

2.2.1.1 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.

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 should be 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.

2.2.1.2 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 also 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.

2.2.1.3 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 should be avoided because they limit the reusability 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* *(66)*.

2.2.1.4 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 ...? 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.

> **Note:** The specializations of `<topic>`, such `<concept>`, `<task>`, `<reference>`, and `<learningContent>` are not included in the DITA 1.3 base edition.

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, the currently-defined information type should be used, either directly, or as a base for specialization. For example, information that is procedural in nature should 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.

2.2.1.5 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 should only be used 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, new specialized topic types should be specialized from appropriate ancestors to meet authoring and output requirements.

2.2.1.6 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 aren't 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 reusability of 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.

2.2.1.7 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. Content that requires a title should 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. Content that requires a title should use `<section>` or `<example>`.

<div>

The `<div>` element enables the arbitrary grouping of content within a topic. The `<div>` element does not include a title. Content that requires a title should 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.

2.2.2 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.

2.2.2.1 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. DITA maps *SHOULD* have `.ditamap` as the file extension.

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.

2.2.2.2 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.

2.2.2.3 DITA map elements

A DITA map describes the relationships among a set of DITA topics. The DITA map and map-group domain elements organize topics into hierarchies, groups, and relationships; they also define keys.

A DITA map is composed of the following elements:

<map>

The `<map>` element is the root element of the DITA map.

<topicref>

The `<topicref>` elements are the basic elements of a map. A `<topicref>` element can reference a DITA topic, a DITA map, or a non-DITA resource. A `<topicref>` element also can have a title, short description, and the same kind of prolog-level metadata that is available in topics.

The `<topicref>` elements can be nested to create a hierarchy, which can be used to define a table of contents (TOC) for print output, online navigation, and parent/child links. Hierarchies can be annotated using the `@collection-type` attribute to define a particular type of relationship, such as a set of choices, a

sequence, or a family. These collection types can affect link generation, and they might be interpreted differently for different outputs.

<reltable>

Relationship tables are defined with the `<reltable>` element. Relationship tables can be used to define relationships among DITA topics or among DITA topics and non-DITA resources. In a relationship table, the columns define common attributes, metadata, or information types (for example, task or troubleshooting) for the resources that are referenced in that column. The rows define relationships between the resources in different cells of the same row.

The `<relrow>`, `<relcell>`, `<relheader>`, and `<relcolspec>` elements are used to define the components of the relationship table. Relationships defined in the relationship table also can be further refined by using the `@collection-type` attribute.

<topicgroup>

The `<topicgroup>` element defines a group or collection outside of a hierarchy or relationship table. It is a convenience element that is equivalent to a `<topicref>` element without an `@href` attribute or navigation title. Groups can be combined with hierarchies and relationship tables, for example, by including a `<topicgroup>` element within a set of siblings in a hierarchy or within a table cell. The `<topicref>` elements so grouped can then share inherited attributes and linking relationships with no effect on the navigation or table of contents.

<topicmeta>

Most map-level elements, including the map itself, can contain metadata inside the `<topicmeta>` element. Metadata typically is applied to an element and its descendants.

<ux-window>

The `<ux-window>` element enables authors to define windowing information for the display of output topics that are appropriate to the delivery platform. Window management is important in user assistance and help system outputs, as well as for other hypertext and electronic delivery modes.

<topichead>

The `<topichead>` element provides a navigation title; it is a convenience element that is equivalent to a `<topicref>` element with a navigation title but no associated resource.

<anchor>

The `<anchor>` element provides an integration point that another map can reference in order to insert its navigation into the referenced map's navigation tree. For those familiar with Eclipse help systems, this serves the same purpose as the `<anchor>` element in that system. It might not be supported for all output formats.

<navref>

The `<navref>` element represents a pointer 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.

<keydef>

Enables authors to define keys. This element is a convenience element; it is a specialization of `<topicref>` that sets the default value of the `@processing-role` attribute to "resource-only". Setting the `@processing-role` attribute to resource-only ensures that the resource referenced by the key definition is not directly included in the navigation that is defined by the map.

<mapref>

Enables authors to reference an entire DITA map, including hierarchy and relationship tables. This element is a convenience element; it is a specialization of `<topicref>` that sets the default value of the `@format` attribute to "ditamap". The `<mapref>` element represents a reference from a parent map to a subordinate map.

<topicset>

Enables authors to define a branch of navigation in a DITA map so that it can be referenced from another DITA map.

<topicsetref>

Enables authors to reference a navigation branch that is defined in another DITA map.

<anchorref>

Enables authors to define a map fragment that is pushed to the location defined by an anchor.

2.2.2.4 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 these 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 (such as `<reltable>` or `<topicref>`), the behavior of the attribute is reserved for future use.

@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 should participate 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.

@navtitle

Specifies a navigation title. This is a shorter version of the title that is used in the navigation only. By default, the `@navtitle` attribute is ignored; it serves only to help the DITA map author keep track of the title of the topic.

> **Note:** The `@navtitle` attribute is deprecated in favor of the `<navtitle>` element. When both a `<navtitle>` element and a `@navtitle` attribute are specified, the `<navtitle>` element should be used.

@locktitle

If `@locktitle` is set to "yes", the `<navtitle>` element or `@navtitle` attribute is used if it is present. Otherwise, the `<navtitle>` element or `@navtitle` attribute is ignored and the navigation title is retrieved from the referenced file.

> **Note:** The `@navtitle` attribute is deprecated in favor of the `<navtitle>` element. When both a `<navtitle>` element and a `@navtitle` attribute are specified, the `<navtitle>` element should be used.

@print

Specifies whether the topic should be included in printed output.

> **Note:** Beginning with DITA 1.3, the `@print` attribute is deprecated. It is replaced with a conditional processing attribute: `@deliveryTarget`. See *@deliveryTarget* for more details.

@search

Specifies whether the topic should be 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 *Chunking (114)*.

@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 *Chunking (114)*.

@processing-role

Specifies whether the topic or map referenced should be 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 should be 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. In addition to the following specified values, processors also *MAY* define additional values.

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).

For more information, see *Example: How the cascade attribute functions (39)*.

@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 *Indirect key-based addressing (66)*.

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:

- `@product`, `@platform`, `@audience`, `@otherprops`, `@rev`, `@status`, `@importance`
- `@dir`, `@xml:lang`, `@translate`
- `@id`, `@conref`, `@conrefend`, `@conkeyref`, `@conaction`
- `@props` and any attribute specialized from `@props` (such as `@deliveryTarget`)
- `@search`

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

2.2.2.5 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.

2.2.2.5.1 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>
```

2.2.2.5.2 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

2.2.2.5.3 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:

```
<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>
```

Figure 3: Simple linking example

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:

```
<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>
```

Figure 4: Linking example with the @linking attribute

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

2.2.2.5.4 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="intro.dita" navtitle="Introduction"/>
  <topicref href="setup.dita" navtitle="Setting up the product"/>
  <topicref href="linux-instructions.dita" navtitle="Linux instructions" platform="linux"/>
</map>
```

The values of the `@platform` attribute set at the map level cascade throughout the map and apply to the "Introduction" and "Setting up the product" topics. However, since the value of the `@cascade` attribute is set to "nomerge", the value of the `@platform` attribute for the "Linux instructions" 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="intro.dita" navtitle="Introduction"/>
  <topicref href="setup.dita" navtitle="Setting up the product"/>
  <topicref href="linux-instructions.dita" navtitle="Linux instructions" platform="linux"
cascade="nomerge"/>
</map>
```

2.2.3 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.

2.2.3.1 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.

2.2.3.2 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:

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

- The `<shortdesc>` element can provide a definition.

In addition, the `<subjectdef>` element can reference a more detailed definition of the subject, for example, another DITA topic or an external resource..

The following behavior is expected of processors:

- Authoring tools *SHOULD* use these lists of controlled values to provide lists from which authors can select values when they specify attribute values.

- Authoring tools *MAY* give an organization a list of readable labels, a hierarchy of values to simplify selection, and a shared definition of the value.

- An editor *MAY* support accessing and displaying the content of the subject definition resource in order to provide users with a detailed explanation of the subject.

- Tools *MAY* produce a help file, PDF, or other readable catalog to help authors better understand the controlled values.

Example: Controlled values that provide additional information about the subject

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

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

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

2.2.3.3 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>
```

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

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

2. Otherwise, if the parent value in the taxonomy has a rule, that matches.

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

The following behavior is expected of processors:

• Processors *SHOULD* be aware of the hierarchies of attribute values that are defined in subject scheme maps for purposes of filtering, flagging, or other metadata-based categorization.

- Processors *SHOULD* validate that the values of attributes that are bound to controlled values contain only valid values from those sets. (The list of controlled values is not validated by basic XML parsers.) If the controlled values are part of a named key scope, the scope name is ignored for the purpose of validating the controlled values.
- Processors *SHOULD* check that all values listed for an attribute in a DITAVAL file are bound to the attribute by the subject scheme before filtering or flagging. If a processor encounters values that are not included in the subject scheme, it *SHOULD* issue a warning.

Example: A hierarchy of controlled values and conditional processing

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

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

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

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

2.2.3.5 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.

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

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

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

2.2.3.7 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.

2.2.3.8 Examples of subject scheme maps

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

2.2.3.8.1 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.

```
<!-- This examples uses @navtitle rather than <navtitle> solely
to conserve space. Best practises for translate include using <navtitle>. -->
<subjectScheme>
  <subjectdef keys="os" navtitle="Operating system">
```

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

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

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

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

att="platform" val="linux"	att="platform" val="redhat"	How platform="redhat" is evaluated	How platform="linux" is evaluated
	action="flag"	Included and flagged with the "redhat" flag, because a flag is available that is specifically for "redhat".	Included and flagged with the "linux" flag.
	Unspecified	Included and flagged with the "linux" flag, because "linux" is flagged and "redhat" is a type of linux	Included and flagged with the "linux" flag.
Unspecified	action="exclude"	Excluded, because all "redhat" content is excluded	If the default for @platform values is "include", this is included. If the default for @platform values is "exclude", this is excluded.
	action="include"	Included.	Included, because all "redhat" content is included, and general 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.

2.2.3.8.2 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" navtitle="Operating system">
    <subjectdef keys="linux" navtitle="Linux">
      <subjectdef keys="redhat" navtitle="RedHat Linux"/>
      <subjectdef keys="suse" navtitle="SuSE Linux"/>
    </subjectdef>
    <subjectdef keys="windows" navtitle="Windows"/>
    <subjectdef keys="zos" navtitle="z/OS"/>
  </subjectdef>
```

```
    <enumerationdef>
      <attributedef name="platform"/>
      <subjectdef keyref="os"/>
    </enumerationdef>
</subjectScheme>
```

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

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

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

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

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

2.2.3.8.3 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="sw" navtitle="Software">
    <subjectdef keyref="os"/>
    <subjectdef keys="app" navtitle="Applications">
      <subjectdef keys="apacheserv" navtitle="Apache Web Server"/>
      <subjectdef keys="mysql" navtitle="MySQL Database"/>
    </subjectdef>
  </subjectdef>
</subjectScheme>
```

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

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

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

2.2.3.8.4 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>
```

2.2.4 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.

2.2.4.1 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.

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.

> **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.

Related Links
Dublin Core Metadata Initiative (DCMI)

2.2.4.2 Metadata attributes

Certain attributes are common across most DITA elements. These attributes support content referencing, conditional processing, application of metadata, and globalization and localization.

2.2.4.2.1 Conditional processing attributes

The metadata attributes specify properties of the content that can be used to determine how the content should be 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.

Filtering and flagging attributes

The following 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". This attribute is a replacement for the now deprecated `@print` attribute.

　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.

Other metadata 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, it should be used 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.

Related Links

3.10.1.2 Metadata attribute group (348)
The metadata attribute group includes common metadata attributes, several of which support conditional processing (filtering and flagging) or the creation of new attribute domain specializations.

3.9 DITAVAL elements (339)
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`.

2.4.3 Conditional processing (profiling) (95)
Conditional processing, also known as profiling, is the filtering or flagging of information based on processing-time criteria.

2.2.4.2.2 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 should be rendered.

2.2.4.2.3 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. Every DITA element (except the `<dita>` element that is used as the root of a ditabase document) *MUST* declare a `@class` attribute.

@domains

This attribute identifies the domain modules (and optionally the structural modules) that are used in a map or topic. Each module also declares its module dependencies. The root element of every topic and map *MUST* declare a `@domains` attribute.

@DITAArchVersion

This attribute identifies the version of the DITA architecture that is used by the XML grammar. The root element of every topic and map *MUST* declare a `@DITAArchVersion` attribute. 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.

Related Links

2.4.7 Processing documents with different values of the domains attribute (122)

When DITA elements are copied from one document to another, processors need to determine the validity of the copied elements. This copying might occur as the result of a content reference (conref) or key reference (keyref), or it might occur in the context of an author editing a DITA document.

2.2.4.3 Metadata in maps and topics

Topic metadata can be specified in a DITA map as well as in the topics that the map references. By default, metadata in the map supplements or overrides metadata that is specified at the topic level, unless the `@lockmeta` attribute of the `<topicmeta>` element is set to "no".

Where metadata about topics can be specified

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:

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 provide alternate titles and a short description. The alternate titles can override their equivalent titles in the topic. The short description in the map *MAY* override the short description in the topic if the `<topicref>` element specifies a `@copy-to` attribute.

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.

How metadata set at both the map and topic level intersects

In a topic, the metadata elements apply to the entire topic. In a map, they supplement or override any metadata that is provided in the referenced topics. 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. The `@lockmeta` attribute on the `<topicmeta>` element controls whether map-specified values override values in the referenced topic.

The `<navtitle>` element is an exception to the rule of how metadata specified by the `<topicmeta>` element cascades. The content of the `<navtitle>` element is used as a navigation title only if the `@locktitle` attribute of the parent `<topicref>` element is set to "yes".

2.2.4.4 Cascading of metadata attributes in a DITA map

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

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

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

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

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

Merging of cascading attributes

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

cascade="merge"

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

cascade="nomerge"

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

Implementers *MAY* define their own custom, implementation-specific tokens. To avoid name conflicts between implementations or with future additions to the standard, implementation-specific tokens *SHOULD* consist of a

prefix that gives the name or an abbreviation for the implementation followed by a colon followed by the token or method name.

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

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

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

Examples of the @cascade attribute in use

Consider the following code examples:

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

Figure 5: Map A

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

Figure 6: Map B

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

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

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

Figure 7: Map C

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

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

Order for processing cascading attributes in a map

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

For example, in the case of `<topicref toc="yes">`, applications *MUST* stop at item *2 (55)* 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 (55)* below when evaluating the `@toc` attribute.

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

1. The `@conref` and `@keyref` attributes are evaluated.

2. The explicit values specified in the document instance are evaluated. For example, a `<topicref>` element with the `@toc` attribute set to "no" will use that value.

3. The default or fixed attribute values are evaluated. For example, the `@toc` attribute on the `<reltable>` element has a default value of "no".

4. The default values that are supplied by a controlled values file are evaluated.

5. The attributes cascade.

6. The processing-supplied default values are applied.

7. After the attributes are resolved within the map, they cascade to referenced maps.

   Note: The processing-supplied default values do not cascade to other maps. For example, most processors will supply a default value of `toc="yes"` when no `@toc` attribute is specified. However, a processor-supplied default of `toc="yes"` *MUST* not override a value of `toc="no"` that is set on a referenced map. If the `toc="yes"` value is explicitly specified, is given as a default through a DTD, XSD, RNG, or controlled values file, or cascades from a containing element in the map, it *MUST* override a `toc="no"` setting on the referenced map. See *Map-to-map cascading behaviors (58)* for more details.

8. Repeat steps *1 (55)* to *4 (55)* 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.

2.2.4.5 Reconciling topic and map metadata elements

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

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

How does it apply to the topic?

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

Does it cascade to other topics in the map?

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

What is the purpose when specified on the <map> element?

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

Table 1: Topicmeta elements and their properties

Element	How does it apply to the topic?	Does it cascade to child `<topicref>` elements?	What is the purpose when set on the `<map>` element?
`<audience>`	Add to the topic	Yes	Specify an audience for the entire map
`<author>`	Add to the topic	Yes	Specify an author for the entire map
`<category>`	Add to the topic	Yes	Specify a category for the entire map
`<copyright>`	Add to the topic	Yes	Specify a copyright for the entire map
`<critdates>`	Add to the topic	Yes	Specify critical dates for the entire map
`<data>`	Add to the topic	No, unless specialized for a purpose that cascades	No stated purpose, until the element is specialized
`<data-about>`	Add the property to the specified target	No, unless specialized for a purpose that cascades	No stated purpose, until the element is specified
`<foreign>`	Add to the topic	No, unless specialized for a purpose that cascades	No stated purpose, until the element is specified
`<keywords>`	Add to the topic	No	No stated purpose
`<linktext>`	Not added to the topic; applies only to links created based on this occurrence in the map	No	No stated purpose
`<metadata>`	Add to the topic	Yes	Specify metadata for the entire map
`<navtitle>`	Not added to the topic; applies only to navigation that is created based on this occurrence in the map. The navigation title will be used whenever the `@locktitle` attribute on the containing `<topicref>` element is set to "yes".	No	No stated purpose
`<othermeta>`	Add to the topic	No	Define metadata for the entire map
`<permissions>`	Add to the topic	Yes	Specify permissions for the entire map

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?
`<prodinfo>`	Add to the topic	Yes	Specify product info for the entire map
`<publisher>`	Add to the topic	Yes	Specify a publisher for the map
`<resourceid>`	Add to the topic	No	Specify a resource ID for the map
`<searchtitle>`	Replace the one in the topic. If multiple `<searchtitle>` elements are specified for a single target, processors can choose to issue a warning.	No	No stated purpose
`<shortdesc>`	Only added to the topic when the `<topicref>` element specifies a `@copy-to` attribute. Otherwise, it applies only to links created based on this occurrence in the map. **Note:** Processors *MAY* or *MAY NOT* implement this behavior.	No	Provide a description of the map
`<source>`	Add to the topic	No	Specify a source for the map
`<unknown>`	Add to the topic	No, unless specialized for a purpose that cascades	No stated purpose, until the element is specified
`<ux-window>`	Not added to the topic	No	Definitions are global, so setting at map level is equivalent to setting anywhere else.

Example of metadata elements cascading in a DITA map

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

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

```
        <topicref href="purpose_ditamaps.dita" navtitle="Purpose of DITA maps"></topicref>
        <!-- ... -->
      </topicref>
</map>
```

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

2.2.4.6 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.

2.2.4.6.1 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:

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

Note that the above list excludes the following attributes:

@format

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

@xml:lang and @dir

Cascading behavior for `@xml:lang` is defined in *The xml:lang attribute (119)*. The `@dir` attribute work the same way.

@scope

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

The `@class` attribute is used to determine the processing roles that cascade from map to map. See *Cascading of roles from map to map (60)* 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"/>
  <topicref href="c.ditamap#branch1" format="ditamap" print="no"/>
  <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="branch1"` within the map `c.ditamap` is treated as if `print="no"` is specified on that element. This means that the topics within the branch with `id="branch1"` do not appear in the printed output generated by `test.ditamap` (except for nested branches within that branch that explicitly set `print="yes"`).

- 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.

2.2.4.6.2 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 *Reconciling topic and map metadata elements (55)*.

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

For example, consider the following code examples:

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

Figure 8: test-2.ditamap

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

Figure 9: b.ditamap

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

- Because the `<shortdesc>` element does not cascade, it does not apply to the DITA topics that are referenced in `a.ditamap`.

- Because the `<audience>` element cascades, the `<audience>` element in the reference to `b.ditamap` combines with the `<audience>` element that is specified at the top level of `b.ditamap`. The result is that the `b-1.dita` topic and `b-2.dita` topic are processed as though hey each contained the following child `<topicmeta>` element:

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

2.2.4.6.3 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.

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

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

Example of cascading roles between maps

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

Referenced map contains a single top-level <topicref> element

The entire branch functions as if it were included in the bookmap; the top-level `<topicref>` element is processed as if it were the `<chapter>` element.

Referenced map contains multiple top-level <topicref> elements

Each top-level `<topicref>` element is processed as if it were a `<chapter>` element (the referencing element).

Referenced map contains a single <appendix> element

The `<appendix>` element is processed as it were a `<chapter>` element.

Referenced map contains a single <part> element, with nested <chapter> elements.

The `<part>` element is processed as it were a chapter element. Nested `<chapter>` elements might not be understandable by processors; applications *MAY* recover as described above.

<chapter> element references a single <topicref> element rather than a map

The referenced `<topicref>` element is processed as if it were a `<chapter>` element.

2.2.4.7 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 Links

The `<resourceid>` *element provides an identifier for applications that must use their own identifier scheme, such as context-sensitive help systems and databases.*

Use the `<ux-window>` *element to provide specifications for a window or viewport in which a user assistance topic or Web page can be displayed. The window or viewport can be referenced by the* `<resourceid>` *element associated with a topic or* `<topicref>` *element.*

2.3 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.

2.3.1 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 should 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.

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

Figure 10: Summary of requirements for the @id attribute

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.

2.3.2 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 should 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.

2.3.3 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)

2.3.4 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 *Subject scheme maps and their usage (40)*.

> **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.

2.3.4.1 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 *Using keys for addressing (69)* 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 *Example: How key scopes affect key precedence (87)* for more information.

2.3.4.2 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.

A key scope includes the following components:

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

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

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

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

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

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

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

Consider the following scenario:

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

Figure 11: Root map

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

Figure 12: installation.ditamap

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

2.3.4.3 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"`.

2.3.4.4 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.
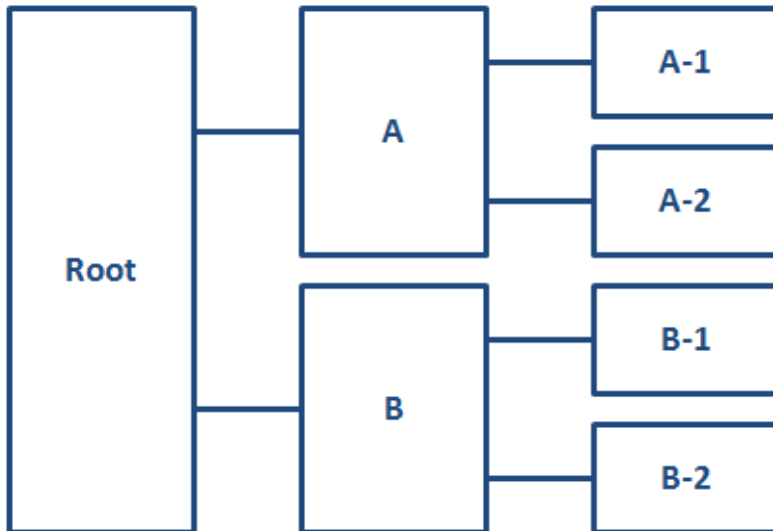
Figure 13: A key scope hierarchy

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 *Example: How key scopes affect key precedence (87)*.

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.

2.3.4.5 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.

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.

2.3.4.6 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. There is no special processing associated with either the `@locktitle` or the `@lockmeta` attributes when attributes are combined.

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.

2.3.4.7 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.

2.3.4.8 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.

For topic references that use the `@keyref` attribute, the effective value of the `<topicref>` element is determined in the following way:

Determining the effective resource

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 `@lockmeta` attribute is honored when metadata content is combined.

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.

2.3.4.9 Processing key references to generate text or link text

Key references can be used to pull text from the key definition. This topic explains how to generate text from a key definition, regardless of whether the key reference also results in a link.

Note: The processing described in this topic is unrelated to the `@conkeyref` attribute. In that case `@conkeyref` is used to determine the target of a `@conref` attribute, after which the normal `@conref` rules apply.

Empty elements that include a key reference with a defined key might get their effective content from the key definition. 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 (such as `@alt` on the `<image>` element)

When an empty element as defined above 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, effective content is used as alternate text, equivalent to creating an `<alt>` sub-element to hold that content.

- For empty `<link>` elements, effective content is used as link text, equivalent to creating a `<linktext>` sub-element to hold that content.

- For empty `<link>` and `<xref>` elements, a key definition can be used to provide a short description in addition to the normal effective content. If the key definition includes `<shortdesc>` inside of `<topicmeta>`, that `<shortdesc>` should be used to provide effective content for a `<desc>` sub-element.

- The `<longdescref>` and `<longquoteref>` elements are empty elements with no effective content. Key definitions are not used to set effective text for these elements.

- The `<param>` element does not have any effective content, so key definitions do not result in any effective content for `<param>` elements.

- The `<indextermref>` element is not completely defined, so determining effective content for this element is also left undefined.

Effective text content is determined using the following set of rules:

1. For elements that also exist as a child of `<topicmeta>` in the key definition, effective content is taken from the first matching direct child of `<topicmeta>`. For example, given the following key definition, an empty `<author>` element with the attribute `keyref="justMe"` would result in the matching content "Just M. Name":

```
<keydef keys="justMe" href="http://www.example.com/my-profile" format="html" scope="external">
  <topicmeta>
    <author>Just M. Name</author>
  </topicmeta>
</keydef>
```

2. For elements that do not allow the `@href` attribute, content is taken from the first `<keyword>` element inside of `<keywords>` inside of the `<topicmeta>`. For example, given the following key definition, empty `<keyword>`, `<term>`, and `<dt>` elements with the attribute `keyref="nohref"` would all result in the matching content "first":

```
<keydef keys="nohref">
  <topicmeta>
    <keywords><keyword>first</keyword><keyword>second</keyword><keyword>third</keyword></
keywords>
  </topicmeta>
</keydef>
```

3. For elements that do allow `@href`, elements from within `<topicmeta>` that are legal within the element using `@keyref` are considered matching text. For example, the `<xref>` element allows `@href`, and also allows `<keyword>` as a child. Using the code sample from the previous item, an empty `<xref>` with `keyref="nohref"` would use all three of these elements as text content; after processing, the result would be equivalent to:

```
<xref keyref="test"><keyword>first</keyword><keyword>second</keyword><keyword>third</keyword></
xref>
```

4. Otherwise, if `<linktext>` is specified inside of `<topicmeta>`, the contents of `<linktext>` are used as the effective content.

   Note: Because all elements that get effective content will eventually look for content in the `<linktext>` element, using `<linktext>` for effective content is a best practice for cases where all elements getting text from a key definition should result in the same value.

5. Otherwise, if the element with the key reference results in a link, normal link text determination rules apply as they would for `<xref>` (for example, using the `<navtitle>` or falling back to the URI of the link target).

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, `<linktext>` 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 should be generalized to `<keyword>`. If the generalized content is also not valid, a text equivalent should be used instead. For example, `<linktext>` 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>` should be used.

2.3.4.10 Examples of keys

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

2.3.4.10.1 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".

2.3.4.10.2 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.

2.3.4.10.3 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. 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.

1. The authoring team references the model information in the `changing-the-oil.dita` topic by using the following mark-up:

```
<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>
        <linktext>X</linktext>
      </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>
        <linktext>Y</linktext>
```

```
          </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.

2.3.4.10.4 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 "windows" 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 "windows" values for the `@platform` attribute excluded, then the effective key definition for "file-chooser-dialog" is the `file-chooser-generic.dita` resource.

2.3.4.10.5 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.

2.3.4.10.6 Example: Key definition with key reference

When a key definition also specifies a key reference, the key reference must also be resolved in order to determine the effective resources 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 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. Along with the navigation title, the `@href`, `@scope`, and `@format` attributes on the "widget" key definition override those on "mainProduct". Thus after key references are resolved, the original reference from `<topicref>` is equivalent to:

```
<topicref id="example"
        href="widgetInfo.dita" scope="local" format="dita" rev="v1r2"
        product="prodCode" audience="sysadmin">
    <topicmeta><navtitle>Information about Widget</navtitle></topicmeta>
</topicref>
```

2.3.4.10.7 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.

Table 2: 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`

Table 3: Effective key definitions for scope-2

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

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

Table 4: 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`

2.3.4.10.8 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.
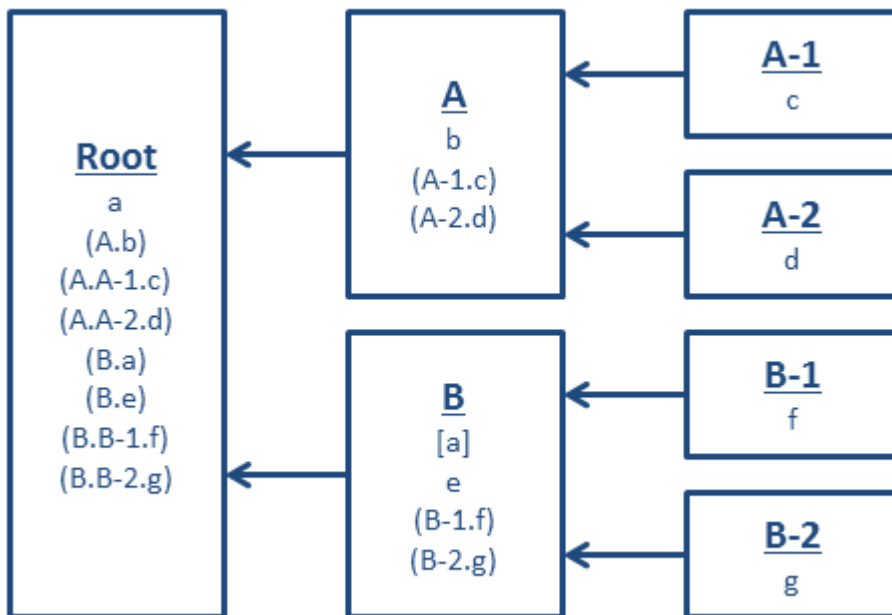
Figure 14: Graphical representation of the key scopes

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 5: 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

Key reference	Resource to which the key resolves
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 6: 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`.

2.3.4.10.9 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.

2.3.4.10.10 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 should include 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>
      <linktext>This product does not offer extended warranties.</linktext>
    </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.

2.3.4.10.11 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.

2.3.4.10.12 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:

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

Figure 15: DITA map for Acme

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

Figure 16: DITA map for OEM partner

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.

2.3.4.10.13 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.

2.3.4.10.14 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".

```
<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 17: Root map

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

Figure 18: Submap

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.

```
<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>
```

Figure 19: Complex map with multiple submaps and scopes

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`).

2.3.4.10.15 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.

2.4 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, and the `<index-sort-as>` element.

2.4.1 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.

2.4.1.1 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 should 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.

- The `@print` attribute is specified on the `<topicref>` element or an ancestor element, and the current processing does not match the value set by the `@print` attribute. For example, `print="printonly"` and the output format is XHTML-based, or `print="no"` and the output format is PDF. (Note that the `@print` attribute is deprecated in DITA 1.3; it is replaced by the `@deliveryTarget` attribute.)

- 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").

2.4.1.2 Indexes

An index can be generated from index entries that occur in topic bodies, topic prologs, or DITA maps.

The specialized indexing domain also provides elements to enable additional indexing function, such as "See" and "See also".

For more information, see *Indexing group elements (272)*.

2.4.2 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

2.4.2.1 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* and *referencing element* as defined in *DITA terminology and notation (20)*.

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 Links

3.10.13.6 The conaction attribute (363)

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.

3.10.13.8 The conkeyref attribute (369)

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.

3.10.13.5 The conref attribute (361)

The *@conref* attribute is used to reference content that can be reused. It allows reuse of DITA elements, including topic or map level elements.

3.10.13.7 The conrefend attribute (366)

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.

2.4.2.2 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.

Except where allowed by weak constraints, 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.

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 pulling content with the conref mechanism, if the referenced element is the same type as the referencing element, and the set of domains declared on the `@domains` attribute in the referenced topic or map instance is the same as or a subset of the domains declared in the referencing document, the element set allowed in the referenced element is guaranteed to be the same as, or a subset of, the element set allowed in the referencing element.

When pushing content with the conref mechanism, the domain checking algorithm is reversed. In this case, if the set of domains declared on the `@domains` attribute in the referencing topic or map instance is the same as or a subset of the domains declared in the referenced document, the element set allowed in the pushed content is guaranteed to be the same as, or a subset of, the element set allowed in the new location.

In both cases, processors resolving conrefs *SHOULD* tolerate specializations of valid elements and generalize elements in the pushed or pulled content fragment as needed for the resolving context.

Related Links

2.5.3.7 domains attribute rules and syntax (131)

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

2.4.2.3 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 *The xml:lang attribute (119)*.

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, as explained below) 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.

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 Links

3.10.13.5.1 Using the -dita-use-conref-target value (362)
> 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.

2.4.2.4 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-topic fragment identifier. It can be resolved only to paragraph p5 in `paras-01.dita`, which contains the content "Paragraph 5 in paras-01".

Paragraph	Value of `@id` attribute on conrefed paragraph	`<xref>` within conrefed paragraph	Resolution
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`.

2.4.3 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. The `@audience`, `@deliveryTarget`, `@otherprops`, `@platform`, and `@props` attributes (along with specializations of `@props`) 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 attributes listed above. 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 Links

3.10.1.2 Metadata attribute group (348)
> The metadata attribute group includes common metadata attributes, several of which support conditional processing (filtering and flagging) or the creation of new attribute domain specializations.

3.9 DITAVAL elements (339)
> 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`.

2.2.4.2.1 Conditional processing attributes (50)
> The metadata attributes specify properties of the content that can be used to determine how the content should be processed. Specialized metadata attributes can be defined to enable specific business-processing needs, such as semantic processing and data mining.

2.4.3.1 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 organize classification metadata into subcategories. This is intended to support situations where a predefined metadata attribute applies to multiple specialized subcategories. 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 `@audience`, `@product`, `@platform`, or `@otherprops`. The following rules apply:

- Groups consist of a name immediately followed by a parenthetical group of one 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 should be 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 "sampleGroup" is specified within both `@platform` and `@otherprops`, a DITAVAL rule for `sampleGroup="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 *DITAVAL elements (339)* for information on how to change default behaviors in DITAVAL provile.

> **Note:** While the grouped values reuse the generalized attribute syntax found in *Attribute generalization (138)*, the `@audience`, `@product`, `@platform`, and `@otherprops` attributes cannot be specialized or generalized.

Related Links

3.10.1.2 Metadata attribute group (348)
> *The metadata attribute group includes common metadata attributes, several of which support conditional processing (filtering and flagging) or the creation of new attribute domain specializations.*

3.9 DITAVAL elements (339)
> *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`.*

2.4.3.2 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 *DITAVAL elements (339)* 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 should evaluate 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:

     1. 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".

     2. 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 should 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 should be 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 should be 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 should be 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 Links

3.10.1.2 Metadata attribute group (348)
> *The metadata attribute group includes common metadata attributes, several of which support conditional processing (filtering and flagging) or the creation of new attribute domain specializations.*

DITAVAL markup with extended filtering example (339)
> *The `<val>` element is the root element of a DITAVAL file.*

2.4.3.3 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 should evaluate each value. Wherever an attribute value that has been set as flagged appears (for example, `audience="administrator"`), the processor should add the flag. When multiple flags apply to a single element, multiple flags should be rendered, typically in the order that they are encountered.

When the same element evaluates as both flagged and included, the element should be 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 should be filtered.

Related Links

DITAVAL markup with extended flagging example (339)
> *The `<val>` element is the root element of a DITAVAL file.*

DITAVAL markup for flagging revisions (344)
> *Identifies a value in the `@rev` attribute that should be flagged in some manner. Unlike the other conditional processing attributes, which can be used for both filtering and flagging, the `@rev` attribute can only be used for flagging.*

2.4.3.4 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. The map or topic references can still use the deprecated `@print` attribute to indicate whether they are applicable to print deliverables.

Within topics, most elements can use the `@deliveryTarget` attribute to indicate the delivery targets to which they apply.

If a referenced topic should be excluded from all output formats, set the `@processing-role` attribute to "resource-only" instead of using the `@deliveryTarget` or `@print` attribute. 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". This attribute is a replacement for the now deprecated `@print` attribute.

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.

@print attribute

Note: Beginning with DITA 1.3, the `@print` attribute is deprecated. Its functionality is superseded by that of the `@deliveryTarget` conditional processing attribute described above.

The `@print` attribute supports the following enumerated values, each of which control the way that print-oriented processors handle the inclusion or exclusion of topics or groups of topics.

Value	Example	Print-oriented processing	Non-print-oriented processing
Unspecified (default)	`<topicref href="foo.dita">`	Topics are **included** in output.	Topics are **included** in output.
yes	`<topicref href="foo.dita" print="yes">`	Topics are **included** in output.	Topics are **included** in output.
printonly	`<topicref href="foo.dita" print="printonly">`	Topics are **included** in output.	Topics are **excluded** from the output.
no	`<topicref href="foo.dita" print="no">`	Topics are **excluded** from the output.	Topics are **included** in output.
-dita-use-conref-target	`<topicref conref="#foo-topic" print="-dita-use-conref-target">`	Topics derive a value for the `@print` attribute from the `@print` attribute of the referenced element. See *Using the -dita-use-conref-target value (362)* for more information.	Topics derive a value for the `@print` attribute from the `@print` attribute of the referenced element. See *Using the -dita-use-conref-target value (362)* for more information.

If a value for the `@print` is not specified explicitly in a map element, but is specified in a map that references the map element, the `@print` value cascades to the referenced map.

2.4.3.5 Examples of conditional processing

This section provides examples that illustrate the ways that conditional processing attributes can be set and used.

Related Links

2.4.3.5.1 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>
```

2.4.3.5.2 Example: Filtering and flagging topic content

A publisher might want to flag information that applies to administrators, and to exclude information that applies to the extended product.

Consider the following DITA source fragment and conditional processing profile:

```
<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 20: DITA source fragment

```
<val>
  <prop att="audience" val="administrator" action="flag">
    <startflag><alt-text>ADMIN</alt-text></startflag>
  </prop>
```

```
    <prop att="product" val="extendedprod" action="exclude"/>
</val>
```

Figure 21: DITAVAL profile

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 should look something like the following:

ADMIN Set the configuration options:

- Set your blink rate

- Do some other stuff

- Do a special thing for Linux

2.4.4 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.

2.4.4.1 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.

2.4.4.2 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.

2.4.4.3 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 must 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 *Branch filtering: Impact on resource and key names (102)* for details on how to manage resource names and key names.

2.4.4.4 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 *Branch filtering: Implications of processing order (105)*.

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.

2.4.4.5 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.

2.4.4.6 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.

2.4.4.6.1 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:

```
<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 22: input.ditamap:

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

Figure 23: Contents of novice.ditaval

When this content is published, the following processing occurs:

- The first topic (`intro.dita`) does not use any of the conditions that are specified in `novice.ditaval`. It is published normally, potentially using other DITAVAL conditions that are specified externally.

- The second topic (`install.dita`) is filtered using any external conditions as well as the conditions that are specified in `novice.ditaval`.

- The third topic (`do-stuff.dita`) is filtered using any external conditions as well as the conditions that are specified in `novice.ditaval`.

- The fourth topic (`advanced-stuff.dita`) is removed from the map entirely, because it is filtered out with the conditions that are specified for the branch.

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

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

2.4.4.6.2 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.

```
<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">
```

```
        <topicref href="mac-specific-stuff.dita" platform="mac"/>
      </topicref>
      <!-- End of installing branch -->
      <topicref href="cleanup.dita"/>
    </topicref>
</map>
```

Figure 24: input.ditamap

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

Figure 25: Contents of win.ditaval

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

Figure 26: Contents of mac.ditaval

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

Figure 27: Contents of linux.ditaval

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

• The first topic (intro.dita) is published normally, potentially using any other DITAVAL conditions that are specified externally.

• The installing branch appears three times, once for each DITAVAL document. The branches are created as follows:

  – The first branch uses the first DITAVAL document (win.ditaval). Resources use their original names as specified in the map. The mac-specific-stuff.dita topic is removed. The resulting branch, with indenting to show the hierarchy, matches the original without the mac topic:

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

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

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

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

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

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

```xml
<map>
  <topicref href="intro.dita"/>
  <!-- Windows installing branch -->
  <topicref href="install.dita">
    <ditavalref href="win.ditaval"/>
    <topicref href="do-stuff.dita">
      <!-- more topics and nested branches -->
    </topicref>
    <topicref href="cleanup.dita"/>
  </topicref>
  <!-- Mac installing branch -->
  <topicref href="install.dita">
    <ditavalref href="mac.ditaval">
      <ditavalmeta><dvrResourceSuffix>-apple</dvrResourceSuffix></ditavalmeta>
    </ditavalref>
    <topicref href="do-stuff.dita">
      <topicref href="mac-specific-stuff.dita" platform="mac"/>
      <!-- more topics and nested branches -->
    </topicref>
    <topicref href="cleanup.dita"/>
  </topicref>
  <!-- Linux installing branch -->
  <topicref href="install.dita">
    <ditavalref href="linux.ditaval">
      <ditavalmeta><dvrResourceSuffix>-linux</dvrResourceSuffix></ditavalmeta>
    </ditavalref>
    <topicref href="do-stuff.dita">
      <!-- more topics and nested branches -->
    </topicref>
    <topicref href="cleanup.dita"/>
  </topicref>
  <!-- Several chapters worth of other material -->
</map>
```

Figure 28: input.ditamap

2.4.4.6.3 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.

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

2.4.4.6.4 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` should be filtered using the conditions specified in the `some.ditaval` document.

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

Figure 29: Map fragment

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

Figure 30: Contents of other.ditamap

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

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

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

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

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

2.4.4.6.5 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.

```
<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 31: input.ditamap

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

Figure 32: Contents of mac.ditaval

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

Figure 33: Contents of linux.ditaval

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

```
<!-- The following wrapperElement is not a real DITA element.
     It is used here purely as an example to illustrate one possible
     way of picturing the conditions. -->
<wrapperElement>
  <map>
    <title>Setting up my product on <keyword platform="mac">Mac</keyword></title>
    <topicmeta>
      <othermeta platform="mac"   name="ProductID" content="1234M"/>
    </topicmeta>
    <ditavalref href="mac.ditaval"/>
    <!-- lots of content, including relationship tables -->
  </map>
  <map>
    <title>Setting up my product on <keyword platform="linux">Linux</keyword></title>
    <topicmeta>
      <othermeta platform="linux" name="ProductID" content="1234L"/>
    </topicmeta>
    <ditavalref href="linux.ditaval"/>
    <!-- lots of content, including relationship tables -->
  </map>
</wrapperElement>
```

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

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

2.4.4.6.6 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.

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

Figure 34: Map fragment

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

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

Figure 35: Map fragment

2.4.4.6.7 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`.

2.4.4.6.8 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 should be reported by processors.

- In situations where resource names map directly to output file names, such as an HTML5 rendering that creates files based on the original resource name, the following name conflicts also occur. In this case a processor would need to report an error, use an alternate naming scheme, or both:

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

  2. `b.dita` generates `b.html` using two alternate set of conditions. One version uses `one.ditaval`, and the other version uses `two.ditaval`.

2.4.5 Chunking

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

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

Reuse of a nested topic

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

Identification of a set of topics as a unit

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

2.4.5.1 Using the @chunk attribute

The specification defines tokens for the `@chunk` attribute that cover the most common chunking scenarios. These tokens can be used to override whatever default chunking behavior is set by a processor. Chunking is necessarily format specific, with chunked output required for some and not supported for other rendered formats. Chunking is also implementation specific with some implementations supporting some, but not all, chunking methods, or adding new methods to the standard ones described in this specification.

The value of the `@chunk` attribute consists of one or more space delimited tokens. Tokens are defined in three categories: for selecting topics, for setting chunking policies, and for defining how the chunk values impact rendering. It is an error to use two tokens from the same category on a single `<topicref>` element.

Selecting topics

When addressing a document that contains multiple topics, these values determine which topics are selected. These values are ignored when the element on which they are specified does not reference a topic. The defined values are:

- **select-topic**: Selects an individual topic without any ancestors, descendents, or peers from within the same document.

- **select-document**: Selects all topics in the target document.

- **select-branch**: Selects the target topic together with its descendent topics.

Policies for splitting or combining documents

The chunking policy tokens determine how source topics are chunked to produce different output chunks, for output formats where that makes sense. When specified on a `<map>` element, the policy becomes the default policy for all

topic references. When specified on a topic reference, the policy applies only to that `<topicref>` and not to any descendant `<topicref>` elements.

- **`by-topic`**: A separate output chunk is produced for each of the selected topics. In particular, topics that are part of multi-topic documents are processed as though they are the root topics within a separate XML document.

- **`by-document`**: A single output chunk is produced for the referenced topic or topics, as though the selected topics were all children of the same document.

Rendering the selection

The following tokens affect how the chunk values impact rendering of the map or topics.

- **`to-content`**: The selection should be rendered as a new chunk of content.

    – When specified on a `<topicref>`, this means that the topics selected by this `<topicref>` and its children will be rendered as a single chunk of content.

    – When specified on the `<map>` element, this indicates that the contents of all topics referenced by the map are to be rendered as a single document.

    – When specified on a `<topicref>` that contains a title but no target, this indicates that processors *MUST* generate a title-only topic in the rendered result, along with any topics referenced by child `<topicref>` elements of this `<topicref>`. The rendition address of the generated topic is determined as defined for the `@copy-to` attribute. If the `@copy-to` attribute is not specified and the `<topicref>` has no `@id` attribute, the address of the generated topic is not required to be predictable or consistent across rendition instances.

    For cross references to `<topicref>` elements, if the value of the `@chunk` attribute is "to-content" or is unspecified, the cross reference is treated as a reference to the target topic. If the reference is to a `<topicref>` with no target, it is treated as a reference to the generated title-only topic.

- **`to-navigation` (DEPRECATED)**: The "to-navigation" token is deprecated in DITA 1.3. In earlier releases, the "to-navigation" token indicates that a new chunk of navigation should be used to render the current selection (such as an individual Table of Contents or related links). When specified on the `<map>` element, this token indicates that the map should be presented as a single navigation chunk. If a cross reference is made to a `<topicref>` that has a title but no target, and the `@chunk` value of that `<topicref>` is set to "to-navigation", the resulting cross reference is treated as a reference to the rendered navigation document (such as an entry in the table of contents).

Some tokens or combinations of tokens might not be appropriate for all output types. When unsupported or conflicting tokens are encountered during output processing, processors *SHOULD* produce warning or error messages. Recovery from such conflicts or other errors is implementation dependent.

There is no default value for the `@chunk` attribute on most elements and the `@chunk` attribute does not cascade from container elements, meaning that the `@chunk` value on one `<topicref>` is not passed to its children. A default by-xxx policy for an entire map can be established by setting the `@chunk` attribute on the `<map>` element, which will apply to any `<topicref>` that does not specify its own by-xxx policy.

When no `@chunk` attribute values are specified or defaulted, chunking behavior is implementation dependent. When variations of this sort are not desired, a default for a specific map can be established by including a `@chunk` attribute value on the `<map>` element.

When chunk processing results in new documents, the resource name or identifier for the new document (if relevant) is determined as follows:

1. If an entire map is used to generate a single chunk (by placing to-content on the `<map>` element), the resource name *SHOULD* be taken from the resource name of the map.

2. If the `@copy-to` attribute is specified, the resource name *MUST* taken from the `@copy-to` attribute.

3. If the `@copy-to` attribute is not specified and one or more keys are specified on the `<topicref>`, the resource name *SHOULD* be constructed using one of the keys.

4. If `@copy-to` and `@keys` are not specified and the by-topic policy is in effect, the resource name *SHOULD* be taken from the `@id` attribute of the topic.

5. If `@copy-to` and `@keys` are not specified and the by-document policy is in effect, the resource name *SHOULD* be taken from the resource name of the referenced document.

When following these steps results in resource name clashes, processors *MAY* recover by generating alternate resource identifiers. For example, when two chunked topics use the same `@id` attribute, a processor could recover by combining the original resource name with the `@id` value instead of using only the `@id` value.

Implementation-specific tokens and future considerations

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

For example: "acme:level2" could be a token for the Acme DITA Toolkit that requests the "level2" chunking method.

2.4.5.2 Chunking examples

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

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

The examples below assume the existence of the following files:

- `parent1.dita`, `parent2.dita`, etc., each containing a single topic with id P1, P2, etc.

- `child1.dita`, `child2.dita`, etc., each containing a single topic with id C1, C2, etc.

- `grandchild1.dita`, `grandchild2.dita`, etc., each containing a single topic with id GC1, GC2, etc.

- `nested1.dita`, `nested2.dita`, etc., each containing two topics: parent topics with id N1, N2, etc., and child topics with ids N1a, N2a, etc.

- `ditabase.dita`, with the following contents:

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

```
      </topic>
  </dita>
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
<map chunk="by-document">
   <topicref href="parent1.dita"
      chunk="to-content" copy-to="parentchunk.dita">
      <topicref href="child1.dita" chunk="select-branch"/>
      <topicref href="child2.dita"
         chunk="to-content select-branch"
         copy-to="child2chunk.dita">
         <topicref href="grandchild2.dita"/>
      </topicref>
      <topicref href="child3.dita">
         <topicref href="grandchild3.dita"
            chunk="select-branch"/>
      </topicref>
   </topicref>
</map>
```

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

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

11. In DITA 1.3, the "to-navigation" chunk is deprecated. In earlier releases, the following map produced two navigation chunks, one for P1, C1, and the other topic references nested under `parent1.dita`, and a second for P2, C2, and the other topic references nested under `parent2.dita`.

```
<map>
   <topicref href="parent1.dita"
         navtitle="How to set up a web server"
         chunk="to-navigation">
     <topicref href="child1.dita"
         chunk="select-branch"/>
     <!-- ... -->
   </topicref>
   <topicref href="parent2.dita"
         navtitle="How to ensure database security"
         chunk="to-navigation">
     <topicref href="child2.dita"
         chunk="select-branch"/>
     <!-- ... -->
   </topicref>
   <!-- ... -->
```

```
        </map>
```

2.4.6 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>` and `<index-sort-as>` elements provide 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.

2.4.6.1 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.

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".

```
<?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 36: installingAcme.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>
```

Figure 37: warningsAcme.dita

2.4.6.2 The @dir attribute

The `@dir` attribute provides instructions to processors about how *bi-directional text* should be 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 ditamaps) or assumed by the processing application. If used, the `@dir` attribute *SHOULD* be specified on the highest level element in the topic or document element of the map.

• 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 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 Links

What you need to know about the BIDI algorithm and inline markup (http://www.w3.org/International/articles/inline-bidi-markup/)
XHTML Bi-directional Text Attribute Module (http://www.w3.org/TR/2004/WD-xhtml2-20040722/mod-bidi.html)
Specifying the direction of text and tables: the dir attribute (http://www.w3.org/TR/html4/struct/dirlang.html#adef-dir)
HTML 4.0 Common Attributes (http://www.htmlhelp.com/reference/html40/attrs.html)

2.4.7 Processing documents with different values of the @domains attribute

When DITA elements are copied from one document to another, processors need to determine the validity of the copied elements. This copying might occur as the result of a content reference (conref) or key reference (keyref), or it might occur in the context of an author editing a DITA document.

A processor can examine the value of the `@domains` attribute and compare the set of modules listed to the set of modules for which it provides direct support. It then can take appropriate action if it does not provide support for a given module, for example, issuing a warning before applying fallback processing.

Documents might have incompatible constraints applied; see *Weak and strong constraints (141)* for more information about constraint compatibility checking.

When copying content from one DITA document to another, processors *SHOULD* determine if the data being copied (the copy source) requires modules that are not required by the document into which the data is to be copied (the copy target). Such a copy operation is always safe if the copy source requires a subset of the modules that are required by the copy target. Such a copy is unsafe if the copy source requires modules that are not required by the copy target.

When a copy operation is unsafe, processors *MAY* compare the copy source to the copy target to determine if the copy source satisfies the constraints of the copy target. If the copy source meets the copy target constraints, the copy operation can proceed. Processors *SHOULD* issue a warning that the copy was allowed but the constraints are not compatible. If the copy source does not meet the constraints of the copy target, processors *MAY* apply generalization until the generalized result either satisfies the copy target constraints or no further generalization can be performed. If the copy operation can be performed following generalization, the processor *SHOULD* issue a warning that the constraints are not compatible and generalization had to be performed in order to complete the copy operation.

Related Links

2.5.3.7 domains attribute rules and syntax (131)

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

2.2.4.2.3 Architectural attributes (51)

> 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.

2.4.8 Sorting

Processors can be configured to sort elements. Typical processing includes sorting glossary 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 `<index-sort-as>` element can be used to specify 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 Links

3.5.3.5 sort-as *(301)*
> *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. The 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 is useful for elements where the base sort phrase is inadequate or non-existent, for example, a glossary entry for a Japanese Kanji phrase.*

3.4.2.5 index-sort-as *(279)*
> *The `<index-sort-as>` element specifies a sort phrase under which an index entry would be sorted.*

2.5 Configuration, specialization, generalization, and constraints

The extension facilities of DITA allow existing vocabulary and constraint modules 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.

2.5.1 Overview of DITA extension facilities

DITA provides three extension facilities: configuration, constraint, and specialization. In addition, generalization augments specialization.

Configuration

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 implemented as sets of vocabulary modules, each of which declares the markup 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.

Generalization

Generalization is the process of reversing a specialization. It converts specialized elements or attributes into the original types from which they were derived.

Constraint

Constraint enables the restriction of content models and attribute lists for individual elements. There is no need to modify the vocabulary modules. Constraints are implemented as constraint modules, which are integrated into document-type shells.

2.5.2 Configuration

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.

2.5.2.1 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 constraint 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 `@domains` and `@class` attributes.

The following figure illustrates the relationship between a DTD-based DITA document, its document-type shell, and the various vocabulary modules that it uses. A similar structure applies to DITA documents that use other XML grammars.
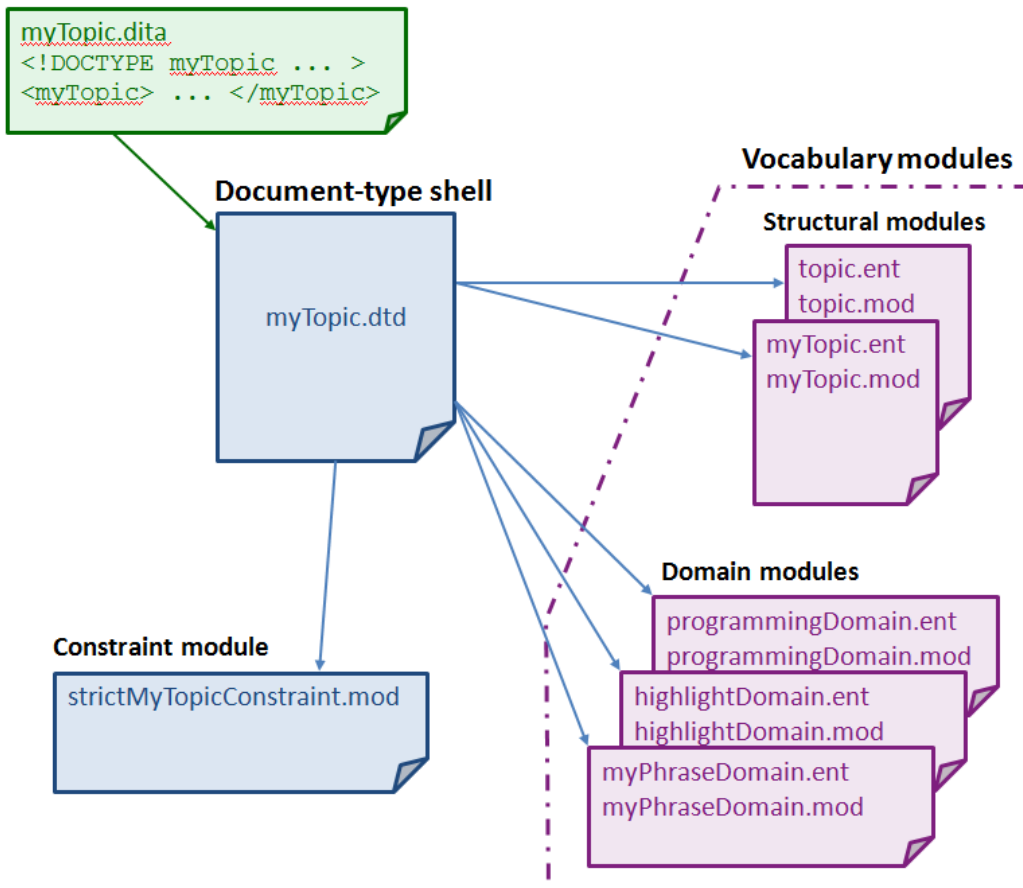
Figure 38: Document type shell

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.

2.5.2.2 Rules for document-type shells

This topic collects the rules that concern DITA document-type shells.

- 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.

- With two exceptions, a document-type shell *MUST NOT* directly define element or attribute types; it only includes and configures vocabulary and constraint modules. The exceptions to this rule are the following:

  – The ditabase document-type shell directly defines the `<dita>` element.

  – RNG- and XML Schema-based shells directly specify values for the `@domains` attribute; these values reflect the details of the domains and structural types that are integrated by the document-type shell.

- 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.com//DTD DITA Topic//EN", where "example.com" is the owner identifier component of the public identifier. An appropriate URN would be "urn:example.com:names:dita:rng:topic.rng".

2.5.2.3 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 modules that are declared in the `@domains` attribute on the root element of the document

- 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, constraint modules, 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.

2.5.2.4 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.

2.5.3 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.

2.5.3.1 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 specialization 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.

2.5.3.2 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.

2.5.3.3 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` and `@domains` 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 should be 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.

2.5.3.4 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.

2.5.3.5 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 is declared as a global attribute. Attribute specializations cannot be limited to specific element types.
- 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.

2.5.3.6 @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 taoken separated by one or more spaces, where *modulename* is the short name of the vocabulary module and *typename* is the element type name. Tokens are ordered left to right from most general to most specialized.

  These tokens provide a mapping for every structural type or domain in the ancestry of the specialized element. The specialization hierarchy for a given element type must reflect any intermediate modules between the base type and the specialization type, even those in which no element renaming occurs.

- At least one trailing space character (" "). The trailing space ensures that string matches on the tokens can always include a leading and trailing space in order to reliably match full tokens.

Rules

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

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

Authors *SHOULD NOT* modify the `@class` attribute.

Example: DTD declaration for @class attribute for the <step> element

The following code sample lists the DTD declaration for the `@class` attribute for the `<step>` element:

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

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

Example: Element with @class attribute made explicit

The following code sample shows the value of the `@class` attribute for the `<wintitle>` element:

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

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

Example: @class attribute with intermediate value

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

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

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

2.5.3.7 @domains attribute rules and syntax

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

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

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

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

Syntax and rules

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

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

structural modules

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

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

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

structural modules with dependencies

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

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

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

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

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

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

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

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

element domains

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

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

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

structural constraint modules

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

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

where:

- *inheritance-hierarchy* is the specialization hierarchy, for example, `topic task`.

- *qualifier* is a string that is specific to the constraints module and characterizes it, for example, "strict" or "requiredTitle" or "myCompany-".

- *Tagname* is the element type name with an initial capital, for example, "Taskbody" or "Topic".

- The literal "-c" indicates that the name is the name of a constraint.

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

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

domain constraint modules

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

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

where:

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

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

attribute domains

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

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

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

Example: Task with multiple domains

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

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

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

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

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

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

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

Related Links

When DITA elements are copied from one document to another, processors need to determine the validity of the copied elements. This copying might occur as the result of a content reference (conref) or key reference (keyref), or it might occur in the context of an author editing a DITA document.

When processing content references, DITA processors compare the restrictions of each context to ensure that the conrefed content is valid in its new context.

2.5.3.8 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.

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

Example: Creating an element domain specialization for SVG

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

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

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

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

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

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

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

2.5.3.9 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.

2.5.4 Generalization

Generalization is the process of reversing a specialization. It converts specialized elements or attributes into the original types from which they were derived.

2.5.4.1 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 enviroment 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 `@domains` attribute should 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 `@domains` 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 (which would also have a different `@domains` attribute declaration). No renaming of elements is needed to remove constraints.

2.5.4.2 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 *Generalization with cross-specialization dependencies* (139).

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.

Generalization and conref

To determine compatibility between a document instance and a target document type when resolving a conref reference, a generalization processor can use the `@domains` and `@class` attributes for the document instance and the `@domains` attribute for the target document type to determine how to rename elements in the resolved instance. For each element type, a generalization processor:

- Iterates over the `@class` attribute from specific to general, inspecting the vocabulary modules.
- Identifies the first vocabulary module that is both present in each document type, with a compatible set of constraints for that vocabulary module. If such a module is not found, the instance can only be generalized to a less constrained document type.

2.5.4.3 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 ui-d 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.

The `@domains` attribute for the root element of the structural type is checked before generalizing domains:

	Source module unspecified	**Source module specified**
Target module unspecified	Do not generalize domain specializations in this structural type.	Check whether the `@domains` attribute lists the specified domain; proceed with generalization if it does, otherwise ignore the structural type instance unless it is itself a candidate for generalization.
Target module specified	Check whether the `@domains` attribute contains the target module. If it does, generalize to the target module. Otherwise, skip the structural type instance unless it is itself a candidate for generalization.	It is an error if the `@domains` attribute matches a specified source but the domain value string does not contain the target. If the `@domains` attribute matches a specified source module and the domain value string does contain the target module, generalize to the target module. Otherwise, ignore the structural type instance unless it is itself a candidate for generalization.

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` and `@domains` attribute, both of which should be supplied by the target document type.

2.5.4.4 Attribute generalization

DITA provides a syntax to generalize attributes that have been specialized from the `@props` or `@base` attribute. Specialization-aware processors *SHOULD* 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 `@domains` 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.

Note: The `@audience`, `@platform`, `@product`, and `@otherprops` attributes allow grouped values that reuse the generalized syntax described here; however, these attributes are not specialized or specializeable. For these attributes, it can be typical to author or edit the grouped values directly.

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.

2.5.4.5 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 should 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.

2.5.5 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.

2.5.5.1 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.

2.5.5.2 Constraint rules

There are certain rules that apply to the design and implementation of constraints.

Contribution to the @domains attribute

Each constraint that is integrated into a DITA document type *MUST* be declared in the `@domains` attribute for each structural type that is integrated into the document type. For DTDs, the contribution for the `@domains` attribute is specified in the constraint module file; for XSD and RELAX NG, the contribution to the `@domains` attribute is specified directly in the document type shell.

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.

2.5.5.3 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.

2.5.5.4 Weak and strong constraints

Constraints can be classified into two categories: Weak and strong. This classification determines whether processors enforce strict compatibility during `@conref` or `@conkeyref` resolution.

Strong constraints

Constraints for which processors enforce strict compatibility during `@conref` or `@conkeyref` resolution.

Weak constraints

Constraints for which a processor does not enforce strict compatibility during `@conref` or `@conkeyref` resolution.

By default, constraints are weak unless they are explicitly designated as strong.

Any constraint declaration can designate a constraint as strong. A constraint can be designated as strong by prefixing the letter "s" to the domains attribute contribution, for example, "`s(topic task strictTaskbody-c)`". Processors also can be configured to treat all constraints as strong.

The following behavior is expected of processors:

- Processors *MAY* perform constraint compatibility checking.

- If processors perform constraint compatibility checking, they *SHOULD* enforce strict compatibility for strong constraints.

- Processors *MAY* have an option for configuring whether all constraints are treated as strong constraints.

2.5.5.5 Conref compatibility with constraints

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

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

Conref pull

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

Conref push

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

Example: Conref pull and constraint compatibility

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

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

Values of `@domains` attribute in document type that contains the referencing element	Values of `@domains` attribute in document type that contains the referenced element	Resolution	Comments
			the referencing topic is less constrained than the referenced topic.
`(topic hi-d) (topic simpleSection-c) s(topic simpleP-c)`	`(topic simpleSection-c) (topic task) (topic hi-d) (topic hi-d basicHighlightingDomain-c)`	Prevented	The referencing document has constraints that are not present in the referenced document, including a strong constraint applied to the `<p>` element.

Example: Conref push and constraint compatibility

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

Values of `@domains` attribute in document type that contains the referencing element	Values of `@domains` attribute in document type that contains the referenced element	Resolution	Comments
`(topic)`	`(topic shortdescReq-c)`	Allowed	Although the content model of the referenced topic is more constrained than the referencing topic, this is a weak constraint and so permitted.
`(topic)`	`s(topic shortdescReq-c)`	Prevented	The constraint is specified as a strong constraint, and the content model of the referenced topic is more constrained than the referencing topic.
`(topic shortdescReq-c)`	`(topic)`	Allowed	The content model of the referencing topic is more constrained than the referenced topic.
`(topic task) (topic hi-d) (topic hi-d basicHighlightingDomain-c)`	`(topic simpleSection-c) (topic task) (topic task simpleStep-c)`	Allowed	The referenced topic has a subset of the vocabulary modules that are integrated into the document-type shell for

Values of `@domains` attribute in document type that contains the referencing element	Values of `@domains` attribute in document type that contains the referenced element	Resolution	Comments
			the referencing topic. For modules used in both document types, the referenced topic is more constrained than the referencing topic, but this is a weak constraint and so permitted.
(topic simpleSection-c) (topic task) (topic hi-d) (topic hi-d basicHighlightingDomain-c)	(topic hi-d) (topic simpleSection-c) s(topic simpleP-c)	Prevented	For the common topic module, the referenced document has more constraints than the referencing document, including a strong constraint applied to the `<p>` element.

2.5.5.6 Examples: Constraints

This section of the specification contains examples and scenarios. They illustrate a variety of ways that constraints can be used; they also provide examples of the DTD coding requirements for constraints and how constraints are integrated into document-type shells.

2.5.5.6.1 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 contraint module is named `acme-TopicConstraint.mod`.

2. She adds the following content to `acme-TopicConstraint.mod`:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.5.5.6.2 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               -->
<!-- ============================================================= -->

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

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

```
                  '%filter-atts;
                  base      CDATA #IMPLIED
                  %base-attribute-extensions;
                  importance (default|deprecated|high|low|normal|obsolete|optional|
                              recommended|required|urgent|-dita-use-conref-target) #IMPLIED
                  rev       CDATA #IMPLIED
                  status    (changed|deleted|unchanged|-dita-use-conref-target) #IMPLIED' >
<!ENTITY % localization-atts
                  'translate (no|yes|-dita-use-conref-target) #IMPLIED
                  xml:lang CDATA #IMPLIED
                  dir       (lro|ltr|rlo|rtl|-dita-use-conref-target) #IMPLIED' >

<!-- Declares the constrained content model. Original definition   -->
<!-- included %univ-atts;, spectitle, and outputclass; 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.

2.5.5.6.3 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-constraints
  "(topic hi-d basic-HighlightingDomain-c)"
>

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

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

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

<!ENTITY % HighlightingDomain-c-dec
  PUBLIC "-//ACME//ENTITIES DITA Highlighting Domain Constraint//EN"
```

```
                  "acme-HighlightingDomainConstraint.mod"
>%basic-HighlightingDomain-c-dec;
```

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

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

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

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

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

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

2.5.5.6.4 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. The information architect creates an entities file named noPhConstraint.ent, where "no" is a qualifier string that characterizes the constraint.

2. The information architect adds the following content to noPhConstraint.ent:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.6 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.

2.6.1 Recognized XML-document grammar mechanisms

The DITA standard recognizes three 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.

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 three 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 Links

Tools for generating DTD or XSD from RELAX NG

2.6.2 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 `<div>` 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.

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, the `@domains` and `@class` attributes) are not explicitly present must implement the DTD compatibility specification in order to get default attribute values.

Related Links
Tools for generating DTD or XSD from RELAX NG

2.6.3 DTD coding requirements

This section explains how to implement DTD based document-type shells, specializations, and constraints.

2.6.3.1 DTD: Overview of coding requirements

DITA coding practices for DTDs rely heavily on entities to implement specialization and constrraints. As such, an understanding of entities is critical when working with DTD document-type shells, vocabulary modules, or constraint modules.

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

In DITA DTDs, every element 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 can be (pre)defined in a document-type shell to mean "ph plus several elements from the highlighting domain". Because the document-type shell places that entity definition before the usual definition, every element that included `%ph;` in its content model now includes `<ph>` plus every phrase specialization in the highlighting domain.

Content models defined as entities

Every element in a DITA DTD defines its content model using an entity. 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. This is done to enable constraints; a constraint module can (pre)define the `%ph.content;` model to remove selected elements.

Attribute sets defined as entities

Every element in a DITA DTD defines its attribute using an entity. 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. As above, this is done to enable constraints; a constraint module can (pre)define the `%ph.attributes;` model to remove selected attributes.

Note: When constructing a constraint 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, which is why they are

described here as (pre)definitions. Most topics about DITA DTDs, including others in this specification, will describe these overrides as redefinitions to ease understanding.

2.6.3.2 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 constraints 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*

2. *Domain constraint integration*

3. *Domain entity declarations*

4. *Domain attributes declarations*

5. *Domain extensions*

6. *Domain attribute extensions*

7. *Topic nesting override*

8. *Domains attribute override*

9. *Content constraint integration*

10. *Topic [or map] element integration*

11. *Domain element integration*

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. By convention, an `.ent` file extension is used to indicate files that define only parameter entities, while a `.mod` file extension is used to indicate files that define elements or constraints.

Topic [or map] entity declarations

This section declares and references an external parameter entity for each of the following:

• 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

Each parameter entity (`.ent`) file contributes a domain token for structural topics or maps. The parameter entity is named *type-name*-dec.

For example, a document-type shell that integrates the `<concept>` specialization would include:

```
<!ENTITY % concept-dec
  PUBLIC "-//OASIS//ENTITIES DITA 1.3 Concept//EN"
         "concept.ent"
>%concept-dec;
```

Domain constraint integration

For each domain constraint module that is integrated into the document type shell, this section declares a parameter entity and references the constraint module file where the constraint is defined. 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 included 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

For each element domain that is integrated into the document-type shell, this section declares a parameter entity and references the external entities file where the element domain is defined. 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:

```
<!ENTITY % hi-d-dec PUBLIC
    "-//OASIS//ENTITIES DITA Highlight Domain//EN"
    "highlightDomain.ent"
>%hi-d-dec;
```

Domain attributes declarations

For each attribute domain that is integrated into the document-type shell, this section declares a parameter entity and references the external entities file where the attribute domain is defined. 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:

```
<!ENTITY % deliveryTargetAtt-d-dec
  PUBLIC "-//OASIS//ENTITIES DITA 1.3 Delivery Target Attribute Domain//EN"
        "deliveryTargetAttDomain.ent"
>%deliveryTargetAtt-d-dec;
```

Domain extensions

For each element that is extended by one or more domains, this section redefines the parameter entity for the element. 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:

```
<!ENTITY % pre
    "pre         |
     %pr-d-pre; |
     %sw-d-pre; |
     %ui-d-pre;">
```

Domain attribute extensions

For each attribute domain that is integrated into the document-type shell, this section redefines the parameter entities for the attribute. It adds an extension to the parameter entity for the relevant attribute.

In the following example, the `@props` attribute is specialized to create the `@new` and `@othernew` attributes, while the `@base` attribute is specialized to create `@newfrombase` and `@othernewfrombase` attributes:

```
<!ENTITY % props-attribute-extensions
        "%newAtt-d-attribute;
```

```
            %othernewAtt-d-attribute;">
<!ENTITY % base-attribute-extensions
        "%newfrombaseAtt-d-attribute;
         %othernewfrombaseAtt-d-attribute;">
```

Topic nesting override

For each topic type that is integrated into the document-type shell, this section specifies whether and how subtopics nest by redefining the `topictype`-info-types entity. The definition is usually 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:

```
<!ENTITY % concept-info-types "concept | myTopicType">
```

Domains attribute override

This section sets the effective value of the `@domains` attribute for the top-level document type that is configured by the document type shell. It redefines the `included-domains` entity to include the text entity for each domain, constraint, and structural specialization that is either included or reused in the document type shell.

In the following example, entities are included for both the troubleshooting specialization and the task specialization on which the troubleshooting specialization depends; for the highlighting and utilities element domains; for the `newAtt-d` attribute domain, and for the `noBasePre-c` constraint module:

```
<!ENTITY included-domains
    "&troubleshooting-att;
     &task-att;
     &hi-d-att;
     &ut-d-att;
     &newAtt-d-att;
     &noBasePre-c-ph;
    "
>
```

Note: Although parameter entities (entities that begin with "%") must be defined before they are referenced, text entities (entities that begin with "&") can be referenced before they are defined. This allows the `included-domains` entity to include the constraint entity, which is not defined until the constraint module is referenced later in the document type shell.

Content constraint integration

For each constraint module that is integrated into the document-type shell, this section declares and references the external module file where the constraint is defined. The parameter entity is named `constraintName`-c-def.

In the following example, the constraint 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 1.3 Strict Taskbody Constraint//EN"
  "strictTaskbodyConstraint.mod"
>%strictTaskbody-c-def;
```

Topic [or map] element integration

For each structural module that is integrated into the document-type shell, this section declares a parameter entity and references the external module file where the structural module is defined. The parameter entity is named *structuralType*-type. The modules must be 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) should be included before the module with a dependency.

The following example declares and references the structural modules that are integrated into the document-type shell for troubleshooting:

```
<!ENTITY % topic-type
   PUBLIC "-//OASIS//ELEMENTS DITA 1.3 Topic//EN"
        "../../base/dtd/topic.mod"
>%topic-type;

<!ENTITY % task-type
   PUBLIC "-//OASIS//ELEMENTS DITA 1.3 Task//EN"
        "task.mod"
>%task-type;

<!ENTITY % troubleshooting-type
   PUBLIC "-//OASIS//ELEMENTS DITA 1.3 Troubleshooting//EN"
        "troubleshooting.mod"
>%troubleshooting-type;
```

Domain element integration

For each element domain that is integrated into the document-type shell, this section declares a parameter entity and references the external module file where the element domain is defined. The parameter entity is named *domainName*-def.

For example, the following code declares and references the parameter entity used for the highlighting domain:

```
<!ENTITY % hi-d-def PUBLIC
    "-//OASIS//ELEMENTS DITA Highlight Domain//EN"
    "highlightDomain.mod"
>%hi-d-def;
```

Note: If a structural module depends on a domain, the domain module should be included before the structural module. This erases the boundary between the final two sections, 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.

2.6.3.3 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. In addition, it covers how to create the @domains attribute contribution for these modules.

A vocabulary module that defines a structural or element domain specialization is composed of two files:

- An entity declaration (.ent) file, which declares the text entities that are used to integrate the vocabulary module into a document-type shell
- 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

@domains attribute contribution

A domain declaration entity is used to construct the effective value of the @domains attribute for a map or topic type.

Text entity name

The name of the text entity is the structural type name or the domain abbreviation, followed by a hyphen ("-") and the literal `att`.

Text entity values

The value of the text entity is the `@domains` attribute contribution for the current module. See *domains attribute rules and syntax (131)* for details on how to construct this value.

For example, the `@domains` attribute contributions for the concept structural module and the highlighting domain module are are constructed as follows.

- `<!ENTITY concept-att "(topic concept)">`

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

Element definitions

A structural or domain vocabulary module must contain 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 must be 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 must be a parameter entity with a name that matches the element type name. The default value is the element type name. This 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 constraints. Element name entities for a single vocabulary module are typically grouped together at the top of the vocabulary module.

For example: `<!ENTITY % topichead "topichead">`

Content-model parameter entity

For each element type, there must be 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. Consistent use and naming of the *tagname*`.content` parameter entity enables the use of constraint modules to restrict the content model.

For example:

```
<!ENTITY % topichead.content
  "((%topicmeta;)?,
    (%anchor; |
     %data.elements.incl; |
     %navref; |
     %topicref;)*)
">
```

Attribute-list parameter entity

For each element type, there must be 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` and the attributes provided by the `global-atts` parameter entity). Consistent use and naming of the *tagname*`.attributes` parameter entity enables the use of constraint modules to restrict attributes.

For example:

```
<!ENTITY % topichead.attributes
 "navtitle CDATA #IMPLIED
  outputclass CDATA #IMPLIED
  keys CDATA #IMPLIED
  copy-to CDATA #IMPLIED
  %topicref-atts;
  %univ-atts;"
>
```

Element declaration

For each element type, there must be 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 must be 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 must define a @class attribute for every element that is declared in the module. The value of the attribute is constructed according to the rules in *class attribute rules and syntax (129)*. The ATTLIST declaration for the @class attribute should also include a reference to the global-atts parameter entity.

For example, the ATTLIST definition for the <topichead> element (a specialization of the <topicref> element in the base map type) includes global attributes with an entity, then the definition of the @class attribute, as follows:

```
<!ATTLIST topichead  %global-atts;  class CDATA "+ map/topicref  mapgroup-d/topichead ">
```

2.6.3.4 DTD: 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 must set the @DITAArchVersion attribute to the version of DITA in use, typically by referencing the arch-atts parameter entity. It must also set the @domains attribute to the included-domains entity. These attributes give processors a reliable way to check the architecture version and look up the list of domains available in the document type.

The following example shows how these attributes are defined for the <concept> element in DITA 1.3:

```
<!ATTLIST concept
  %concept.attributes;
  %arch-atts;
  domains  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.

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 will be 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:

```
<!ENTITY % concept-info-types "%concept;">
```

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 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 rare cases, it might not be desirable to control topic nesting with a parameter entity. For example:

- If a specialized topic type should never allow any nested topics, regardless of context, it can be defined without any entity or any nested topics.

- If a specialized topic type should only ever 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.

2.6.3.5 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.

Parameter entity value

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

Example

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

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

2.6.3.6 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 `@domains` attribute for the attribute domain.

An attribute domain's name 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 `@domains` attribute contribution for the module; see *domains attribute rules and syntax (131)* 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 "a(props deliveryTarget)" >
```

2.6.3.7 DTD: 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.

Structural constraint modules

Structural constraint modules have the following requirements:

@domains contribution entity name and value

The constraint module should contain a declaration for a text entity with the name `tagname`-constraints, where *tagname* is the name of the element type to which the constraints apply. The value of the text entity is the `@domains` attribute contribution for the module; see *domains attribute rules and syntax (131)* for details on how to construct this value.

For example, the following text entity provides the declaration for the strict task constraint that is shipped with the DITA standard.

```
<!ENTITY taskbody-constraints
   "(topic task strictTaskbody-c)"
>
```

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`, and also removes `@spectitle` and `@othertype`:

```
<!ENTITY % note.attributes
     "type  (attention | caution | note ) #IMPLIED
      %univ-atts;
      outputclass  CDATA  #IMPLIED">
```

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:

@domains contribution entity name and value

The constraint module should contain a declaration for a text entity with the name *domain*`Domain-constraints`, where *domain* is the name of the domain to which the constraints apply, for example, "Highlighting" or "Programming". The value of the text entity is the `@domains` attribute contribution for the module; see *domains attribute rules and syntax (131)* for details on how to construct this value.

For example, the following text entity provides the declaration for a constraint module that restricts the highlighting domain:

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

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* 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. Omitting a base type constitutes a form of constraint; as with any other constraint, this form of constraint must contribute a token to the `@domains` attribute. That token can be defined in a module file (which does not define any other entities or values), or the token can be placed directly into the document-type shell definition for the `included-domains` entity.

In the following example, the `<pre>` base type is removed from the entity declaration, effectively allowing only specializations of `<pre>` but not `<pre>` itself. This omission would require the use of a `@domains` contribution token within the `included-domains` entity.

```
<!ENTITY % pre
    "%pr-d-pre; |
     %sw-d-pre; |
     %ui-d-pre;">
```

2.6.4 RELAX NG coding requirements

This section explains how to implement RELAX NG based document-type shells, specializations, and constraints.

2.6.4.1 RELAX NG: Overview of coding requirements

RELAX NG modules are self-integrating, which means that they automatically add to the content models and attribute sets they extend. This means that information architects do not have much work to do when assembling vocabulary modules and constraints 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 specialized vocabulary 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.

RELAX NG grammars for DITA document-type shells, vocabulary modules, and constraint modules *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 `<div>` 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 constraint modules that use the RELAX NG compact syntax can use the same organization requirements as those defined for the RELAX NG XML syntax.

2.6.4.2 RELAX NG: Coding requirements for document-type shells

A document-type shell integrates one or more topic type or map type modules, zero or more domain modules, and zero or more constraint modules.

Because RELAX NG modules are self-integrating, document-type shells only need to include vocabulary modules. Unlike DTDs, there is no separate specification required in order to integrate domain and nested topic elements into the base content models.

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 one 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>
```

DITA domains attribute

The document-type shell must list the domain or structural modules that are named as dependencies in the `@domains` attribute value. Unlike DTDs, a default value for `@domains` cannot automatically be constructed using RELAX NG facilities. Instead, the values used to construct `@domains` are taken from each vocabulary and constraint module, in addition to any domains contributions based on constraints implemented within the shell.

For example:

```
<div>
  <a:documentation>DOMAINS ATTRIBUTE</a:documentation>
  <define name="domains-att">
    <optional>
      <attribute name="domains"
                 a:defaultValue="(topic hazard-d)
```

```
                               (topic hi-d)
                               (topic indexing-d)
                               (topic ut-d)
                               a(props deliveryTarget)"
      />
    </optional>
  </define>
</div>
```

Content constraint integration

The document-type shell must include any constraint modules. Constraint modules include the module they override and override the patterns that they constrain directly in the constraint module itself. Any module that is constrained in this section (including the base topic or map modules) will be left out of the following section.

For example, when the following constraint is included for the task module, the task module itself will *not* be included in the shell; the task module itself is included within `strictTaskbodyConstraintMod.rng`:

```
<div>
<a:documentation>CONTENT CONSTRAINT INTEGRATION</a:documentation>
  <include href="strictTaskbodyConstraintMod.rng">
    <define name="task-info-types">
      <ref name="task.element"/>
    </define>
  </include>
</div>
```

Module inclusions

The document-type shell must include any unconstrained domain or structural module. If the top-level map or topic type is unconstrained, it is also included in this section.

For example:

```
<div>
  <a:documentation>MODULE INCLUSIONS</a:documentation>
  <include href="topicMod.rng"/>
  <include href="highlightDomainMod.rng"/>
  <include href="utilitiesDomainMod.rng"/>
  <include href="indexingDomainMod.rng"/>
  <include href="hazardstatementDomainMod.rng"/>
</div>
```

Constraining domains in the shell

Domains can be constrained to disallow some extension elements without the use of a separate module file. This is done by overriding the base type pattern within the reference to the domain module. In this case, the constraint represented by the pattern redefinition still must be declared in the `@domains` attribute; the `@domains` contribution should be documented in the document-type shell with the constraint. There is not a designated section of the document-type shell for this type of constraint; it can be placed either in *Content constraint integration* or *Module inclusions*.

The following example demonstrates the portion of a document type shell that includes the highlight domain module while directly constraining that module to remove the `<u>` element. The `<ditaarch:domainsContribution>` element is used to document the `@domains` contribution for this constraint.

```
  <div>
    <a:documentation>MODULE INCLUSIONS</a:documentation>
    <include href="topicMod.rng"/>
    <include href="hazardstatementDomainMod.rng"/>
    <include href="highlightDomainMod.rng">
      <ditaarch:domainsContribution
        >(topic hi-d-noUnderline-c)</ditaarch:domainsContribution>
```

```
        <define name="u">
          <notAllowed></notAllowed>
        </define>
      </include>
      <include href="indexingDomainMod.rng"/>
      <include href="utilitiesDomainMod.rng"/>
    </div>
```

ID-defining element overrides

This section must declare 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 avoid errors from RELAX NG parsers that would otherwise complain about different uses of the `@id` attribute.

Typically, this section lists only a few elements, such as topic types or the `<anchor>` element, but it could include specializations that constrain `@id`. In addition, foreign vocabularies require you to include exclusions for the namespaces used by those domains.

For example, this section declares that `<topic>` and `<task>` use an `@id` attribute with an XML data type of ID, along with any elements in the SVG or MathML namespaces. Each of these is excluded from the "any" pattern by placing them within the `<except>` rule as shown below:

```
<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>
```

2.6.4.3 RELAX NG: Coding requirements for element type declarations

Structural and domain vocabulary modules have the same coding requirements for element type declarations. Each RELAX NG vocabulary module consists of a single module file.

Element definitions

A structural or element-domain vocabulary module must contain a declaration for each specialized element type that is named in the module. While the XML standard allows content models to refer to undeclared element types, all element types that are named in content models or attribute list declarations within a vocabulary module must have a RELAX NG element declaration. The RELAX NG element declaration can occur in one of the following:

• 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 must be 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 must be used to define the content model and attributes for the element type. Each set of patterns is typically grouped within a `<div>` element for clarity.

- *tagname*.content defines the complete content model for the element *tagname*. The content model pattern can be overridden in constraint modules to further constrain the content model for the element type.

- *tagname*.attributes defines the complete attribute list for the element *tagname*, except for @class and the attributes provided by the `global-atts` pattern. The attribute list declaration can be overridden in constraint modules to further constrain the attribute list for the element type.

- *tagname*.attlist is 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.

- *tagname*.element is 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.

The following example shows the declaration for the `<topichead>` element, including the definition for each pattern described above.

```
<div>
  <a:documentation>LONG NAME: 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="navtitle"/>
    </optional>
    <optional>
      <attribute name="outputclass"/>
    </optional>
    <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" ditaarch:longName="Topic head">
      <a:documentation>The &lt;topichead> element provides a title-only entry in a navigation map,
        as an alternative to the fully-linked title provided by the &lt;topicref> 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>
```

idElements pattern contribution

Element types that declare the `@id` attribute as type "ID", including all topic and map element types, must provide a declaration for the `idElements` pattern. This is needed 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, and also includes a reference to the `global-atts` attribute list pattern. `@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 *class attribute rules and syntax (129)*.

For example:

```
<define name="anchorref.attlist" combine="interleave">
  <ref name="global-atts"/>
  <optional>
    <attribute name="class"
        a:defaultValue="+ map/topicref mapgroup-d/anchorref "
    />
  </optional>
</define>
```

2.6.4.4 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.

All vocabulary and constraint modules must document their `@domains` attribute contribution. The value of the contribution is constructed according to the rules found in *domains attribute rules and syntax (131)*. 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 can also be used.

Required topic and map element attributes

The topic or map element type must reference the `arch-atts` pattern, which defines the `@DITAArchVersion` attribute in the DITA architecture namespace and sets the attribute to the version of DITA in use. In addition, the topic or map element type must reference the `domains-att` pattern, which will pull in a definition for the `@domains` attribute. These attributes give processors a reliable way to check the architecture version and list of available domains.

For example, the following definition references the `arch-atts` and `domains-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="domains-att"/>
  </define>
  <!-- ... -->
</div>
```

Controlling nesting in topic types

Specialized topics typically define an `info-types` style pattern to specify default topic nesting. Document type shells can then 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.

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 will be 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>
```

If the topic does not permit nested topics by default, this pattern uses the `<empty>` element. For example:

```
<define name="learningAssessment-info-types">
  <empty/>
</define>
```

The `info-types` pattern can also be used to refer to common nesting rules across the document-type shell. 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 *SHOULD* be 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 rare cases, it might not be desirable to control topic nesting with the `info-types` pattern. For example:

- If a specialized topic type should never allow any nested topics, regardless of context, it can be defined without any pattern or any nested topics.

- If a specialized topic type should only ever 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.

2.6.4.5 RELAX NG: Coding requirements for element domain modules

Vocabulary modules that define element domains must define an extension pattern for each element that is extended by the domain. These patterns are used when including the domain module in a document-type shell.

All vocabulary and constraint modules must document their `@domains` attribute contribution. The value of the contribution is constructed according to the rules found in *domains attribute rules and syntax (131)*. 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 can also be used.

For each element type that is extended by the element domain module, the module must define a domain extension pattern. The pattern consists of a choice group of references to element-type name patterns, with one reference to each extension of the base element type.

The name of the pattern uses the following format, where *shortName* is the short name for the domain, and *elementName* is the name of the element that is extended:

```
shortName-d-elementName
```

For example, the following pattern extends the `<ph>` element type by adding the specializations of `<ph>` that are defined in the highlighting domain:

```
<define name="hi-d-ph">
  <choice>
```

```
        <ref name="b.element"/>
        <ref name="i.element"/>
        <ref name="sup.element"/>
        <ref name="sub.element"/>
        <ref name="tt.element"/>
        <ref name="u.element"/>
    </choice>
</define>
```

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 extension pattern. Because
the pattern uses a `@combine` value of "choice", the effect is that the domain-provided elements are automatically
added to the effective content model of the extended element in any grammar that includes the domain module.

For example, the following pattern adds the highlighting 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>
```

2.6.4.6 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. An attribute domain module defines exactly one new attribute type.

All vocabulary and constraint modules must document their `@domains` attribute contribution. The value of the
contribution is constructed according to the rules found in *domains attribute rules and syntax (131)*. 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 can
also be used.

An attribute domain's name 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 consists of one file, which has three sections:

Domains attribute contribution

The `@domains` contribution must be documented in the module. The value is constructed according to the
rules found in *domains attribute rules and syntax (131)*.

Attribute extension pattern

The attribute extension pattern extends either the `@props` or `@base` attribute set pattern to include the attribute
specialization.

For 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="deliveryTargetAtt-d-attribute"/>
</define>
```

For 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>
```

Attribute declaration pattern

The specialized attribute is declared in a pattern named *domainName*-d-attribute. The attribute must be defined as optional. For example, the `@deliveryTarget` specialization of `@props` is defined as follows:

```
<define name="deliveryTargetAtt-d-attribute">
  <optional>
    <attribute name="deliveryTarget"/>
  </optional>
</define>
```

2.6.4.7 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.

All vocabulary and constraint modules must document their `@domains` attribute contribution. The value of the contribution is constructed according to the rules found in *domains attribute rules and syntax (131)*. 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 can also be used.

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 patterns as needed to implement the constraint.

For example, a constraint module that modifies the `<section>` element needs to import the base module `topicMod.rng`. Within that import, it will constrain 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.

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:

• 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>
```

- 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`.

Example: contribution to the @domains attribute for structural constraint module

The following code fragment specifies the contribution to the `@domains` attribute as `(topic task strictTaskbody-c)`:

```
<moduleDesc>
   <!-- ... -->
   <moduleMetadata>
      <!-- ... -->
      <domainsContribution>(topic task strictTaskbody-c)</domainsContribution>
   </moduleMetadata>
</moduleDesc>
```

Example: contribution to the @domains attribute for domain constraint module

The following code fragment illustrates the domains contribution for a constraint module that restricts the task requirements domain:

```
<moduleDesc>
   <!-- ... -->
   <moduleMetadata>
      <!-- ... -->
      <domainsContribution>(topic task taskreq-d requiredReqcondsTaskreq-c)</domainsContribution>
   </moduleMetadata>
</moduleDesc>
```

2.6.5 XML Schema coding requirements

This section explains how to implement XML Schema (XSD) based document-type shells, specializations, and constraints.

2.6.5.1 XML Schema: Overview and limitations of coding requirements

DITA coding practices for XML Schema rely on the XSD redefine facility in order to implement specializations or constraints. However, limitations in the redefine facility can present problems for some DITA modules implemented in XML Schema.

Specializations and constraints in XML Schema are implemented using the XSD `<xs:redefine>` facility. However, this facility does not allow sequence groups to be directly constrained. Thus, to support both specialization and constraints, it might be necessary to refactor content models into named groups that can be redefined. In order to keep the XSD, RELAX NG, and DTD implementations as consistent as possible, the DITA Technical Committee only refactored those content models that were required for OASIS-provided grammars, the strict task body and machinery-industry task. The other DITA content models distributed by OASIS have not been refactored.

You *MAY* modify OASIS-provided XSD modules to refactor content models if required by your constraint. You *SHOULD* notify the DITA Technical Committee (TC) of your constraint requirements, so the TC can consider adding the required refactoring to the OASIS-provided XSDs.

2.6.5.2 XML Schema: Coding requirements for document-type shells

XSD-based document-type shells are organized into sections; each section contains a specific type of declaration.

XSD-based document-type shells use the XML Schema redefine feature (`<xs:redefine>`) to override base group definitions for content models and attribute lists. This facility is analogous to the parameter entities that are used for DTD-based document-type shells. Unlike DTD parameter entities, an `<xs:redefine>` both includes the XSD file that it redefines and holds the redefinition that is applied to the groups in the included XSD file. Thus, for XSD files that define groups, the file can be included using `<xs:include>` if it is used without modification or using `<xs:redefine>` if any of its groups are redefined.

XSD-based document-type shells contain the following sections.

Topic or map domains

For each element or attribute domain that is integrated into the document-type shell, this section uses an `<xs:include>` element to include the XSD module for that domain.

For example:

```
<xs:include schemaLocation="urn:oasis:names:tc:dita:xsd:deliveryTargetAttDomain.xsd:1.3"/>
<xs:include schemaLocation="urn:oasis:names:tc:dita:xsd:highlightDomain.xsd:1.3"/>
<xs:include schemaLocation="urn:oasis:names:tc:dita:xsd:indexingDomain.xsd:1.3"/>
```

Group definitions

The group inclusion section contains `<xs:include>` or `<xs:redefine>` references for element groups. The group files define named groups that are used to integrate domain-provided element and attribute types into base content models. There is one group file for each structural type; domain files can also have group files.

For both map and topic shells, this section also must include or redefine the following groups; it must also include the module file for each group:

- Common element group (`commonElementGrp.xsd` and `commonElementMod.xsd`)

- Metadata declaration group (`metaDeclGrp.xsd` and `metaDeclMod.xsd`)

- Table model group (`tblDeclGrp.xsd` and `tblDeclMod.xsd`)

The group files and the module files for base groups can be specified in any order.

For each element extended by one or more domains, the document-type shell must redefine the model group for the element to a list of alternatives including the literal name of the element and the element extension model group from each domain that is providing specializations. To integrate a new domain in the document-type shell, use the schema `<xs:redefine>` mechanism to import a group definition file while redefining and extending an element from that group. The model group requires a reference to itself to extend the base model group.

For each attribute extended by one or more domains, the document-type shell must redefine the attribute extension model group for the attribute to a list of alternatives including the literal name of the attribute and the attribute extension model group from each domain that is providing specializations. To integrate a new attribute domain in the document-type shell, use the schema `<xs:redefine>` mechanism to import the `commonElementGrp.xsd` group file while redefining and extending the base attribute.

For example, the following portion of a document-type shell includes the common metadata module and then adds a domain extension of the `<metadata>` element from the metadata group. It also extends the `@props` attribute from the common element module to add the specialized attribute `@deliveryTarget`.

```
<xs:include schemaLocation="urn:oasis:names:tc:dita:xsd:metaDeclMod.xsd:1.3"/>
<!-- ... -->
<xs:redefine schemaLocation="urn:oasis:names:tc:dita:xsd:commonElementGrp.xsd:1.3">
  <!-- ...Redefinition of any elements in common module -->
  <xs:attributeGroup name="props-attribute-extensions">
    <xs:attributeGroup ref="props-attribute-extensions"/>
    <xs:attributeGroup ref="deliveryTargetAtt-d-attribute"/>
  </xs:attributeGroup>
</xs:redefine>
<xs:redefine schemaLocation="urn:oasis:names:tc:dita:xsd:metaDeclGrp.xsd:1.3">
  <xs:group name="metadata">
    <xs:choice>
      <xs:group ref="metadata"/>
      <xs:group ref="relmgmt-d-metadata"/>
    </xs:choice>
  </xs:group>
</xs:redefine>>
```

Module inclusions

The module inclusion section includes the module XSD files for structural types used in the shell. These must be placed after the group and files and redefinitions.

This section must also include any other module XSD files required by the topic or map types. For example, if a troubleshooting specialization is specialized from topic but includes elements from task, then the task structural module must be included in the document shell.

If a structural type is constrained, that constraint will be included rather than the module itself; for example, in a document-type shell that constrains the task specialization, the task constraint module will be included rather than the task module.

For example, the following portion of a document-type shell includes the structural modules for topic and concept:.

```
<xs:include schemaLocation="urn:oasis:names:tc:dita:xsd:topicMod.xsd:1.3"/>
<xs:include schemaLocation="urn:oasis:names:tc:dita:xsd:conceptMod.xsd:1.3"/>
```

Domains attribute declaration

The `@domains` attribute declaration section declares the `@domains` attribute for the shell. It does this by redefining the `domains-att` group, adding one token for each vocabulary and constraint module integrated by the shell. See *domains attribute rules and syntax (131)* for details on the syntax for domains tokens.

For example, the following sample defines the `domains-att` to include the OASIS domains for map group, indexing, and `@deliveryTarget`:

```
<xs:attributeGroup name="domains-att">
  <xs:attribute name="domains" type="xs:string"
    default="(map mapgroup-d) (topic indexing-d) a(props deliveryTarget)"/>
</xs:attributeGroup>
```

Info-types definition

This section defines whether and how topics can nest by redefining the `info-types` group. That group is referenced but undefined in the module files, so it must be defined in the shell. Topic testing can be disallowed by setting the `info-types` group to reference the `<no-topic-nesting>` element, with the `@maxOccurs` and `@minOccurs` attributes each set to "0".

Optionally, *topictype*-info-types groups can be redefined to provide more fine grained control of nesting with specialized topic types. As with domain extensions, this is done by redefining the group while importing the module that defines the group.

For example, in the concept vocabulary module delivered by OASIS, the `concept-info-types` group controls which topics can nest inside the `<concept>` element. That group is defined as including `<concept>` plus the `info-types` group. The following examples demonstrate how to control topic nesting within `<concept>` using a document-type shell.

- To have `<concept>` only nest itself, the `info-types` group must be defined so that it does not add any additional topics:

```
<xs:group name="info-types">
  <xs:choice>
    <xs:element ref="no-topic-nesting" maxOccurs="0" minOccurs="0"/>
  </xs:choice>
</xs:group>
```

- In order to turn off topic nesting entirely within `<concept>`, the `concept-info-types` group must be redefined to remove `<concept>`, and the `info-types` group must be defined as above:

```
<xs:group name="info-types">
  <xs:choice>
    <xs:element ref="no-topic-nesting" maxOccurs="0" minOccurs="0"/>
  </xs:choice>
</xs:group>

<xs:redefine schemaLocation="urn:oasis:names:tc:dita:xsd:conceptMod.xsd:1.3" >
  <xs:group name="concept-info-types">
    <xs:choice>
      <xs:group ref="info-types"/>
    </xs:choice>
  </xs:group>
</xs:redefine>
```

- In order to add `<topic>` as a nesting topic within `<concept>`, define `info-types` to allow any number of `<topic>` elements:

```
<xs:group name="info-types">
  <xs:choice>
    <xs:element ref="topic" maxOccurs="unbounded" minOccurs="0"/>
  </xs:choice>
</xs:group>
```

- With the preceding example, `<concept>` is allowed to nest either `<concept>` or `<topic>`. In order to make `<topic>` the *only* valid child topic, the `concept-info-types` must be redefined as follows:

```
<xs:redefine schemaLocation="urn:oasis:names:tc:dita:xsd:conceptMod.xsd:1.3" >
  <xs:group name="concept-info-types">
    <xs:choice>
      <xs:group ref="info-types"/>
    </xs:choice>
  </xs:group>
</xs:redefine>
```

2.6.5.3 XML Schema: Coding requirements for element type declarations

Structural and domain vocabulary modules have the same XSD coding requirements for element type declarations.

Element definitions

A structural or domain vocabulary module must contain a declaration for each specialized element type named by the module. While the XSD standard allows content models to refer to undeclared element types, all element

types named in content models within a vocabulary module must have an `<xs:element>` declaration, either in the vocabulary module, in a base module from which the vocabulary module is specialized, or in a required domain module.

Domain modules consist of a single XSD document. Structural modules consist of two modules; one module contains all element name groups, and the other contains all other declarations for the module.

For each element type that is declared in the vocabulary module, the following set of groups and declarations must be used to define the content model and attributes for the element type. These groups are typically placed together within the module for clarity.

- For each element type declared in the vocabulary module there must be an `<xs:group>` element whose name is the element type name, and whose one member is a reference to the element type. This element name group provides a layer of abstraction that facilitates redefinition. A document-type shell can redefine an element group to add domain-specialized elements or to replace a base element type with one or more specializations of that type.

- Each element type must have a corresponding content model group named *tagname*`.content`. The value of the group is the complete content model definition; the content model group can be overridden in constraint modules to further constrain the content model.

- Each element type must have a corresponding attribute group named *tagname*`.attributes`. The value of the group is the complete attribute set for the element type, except for the `@class` attribute. Like the content model, this group can be overridden in a constraint module to constrain the attribute set.

- Each element type must have a complex type definition named *tagname*`.class`, which references the *tagname*`.content` and *tagname*`.attributes` groups.

- Each element type must have an `<xs:element>` declaration named *tagname*, that uses as its type the *tagname*`.class` complex type and extends that complex type to add the `@class` attribute for the element.

For example, the following set of declarations shows each of the required groups and definitions for the specialized `<codeph>` element.

```
<xs:group name="codeph">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="codeph"/>
    </xs:choice>
  </xs:sequence>
</xs:group>

<xs:group name="codeph.content">
  <xs:sequence>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="basic.ph.notm"/>
      <xs:group ref="data.elements.incl"/>
      <xs:group ref="draft-comment"/>
      <xs:group ref="foreign.unknown.incl"/>
      <xs:group ref="required-cleanup"/>
    </xs:choice>
  </xs:sequence>
</xs:group>

<xs:attributeGroup name="codeph.attributes">
  <xs:attributeGroup ref="univ-atts"/>
  <xs:attribute name="outputclass" type="xs:string"/>
  <xs:attributeGroup ref="global-atts"/>
</xs:attributeGroup>

<xs:complexType name="codeph.class" mixed="true">
  <xs:sequence>
    <xs:group ref="codeph.content"/>
  </xs:sequence>
  <xs:attributeGroup ref="codeph.attributes"/>
</xs:complexType>

<xs:element name="codeph">
  <xs:annotation>
```

```
      <xs:documentation> <!-- documentation for codeph --> </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="codeph.class">
          <xs:attribute ref="class" default="+ topic/ph pr-d/codeph "/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
```

2.6.5.4 XML Schema: Coding requirements for structural modules

An XSD structural module declares a top-level map or topic type. It is implemented as a pair of XSD documents, one that defines groups used to integrate and override the type and one that defines the element types specific to the type.

All vocabulary and constraint modules must document their `@domains` attribute contribution. The OASIS grammar files use a `<dita:domainsModule>` element to document the contribution; this element is used consistently to make it easy to find values when creating a document type shell. An XML comment or `<xs:appinfo>` element can also be used.

Module files

A structural vocabulary module has two files:

- A module schema document.
- A module group definition schema document.

Required topic and map element attributes

The root element for a structural type must reference the `@DITAArchVersion` attribute and the `@domains` attribute. These attributes give processors a reliable way to check the architecture version and look up the list of domains available in the document type. The attributes are referenced as in the following example:

```
<xs:attributeGroup name="concept.attributes">
  <!-- Various other attributes -->
  <xs:attribute ref="ditaarch:DITAArchVersion"/>
  <xs:attributeGroup ref="domains-att"/>
  <xs:attributeGroup ref="global-atts"/>
</xs:attributeGroup>
```

Controlling nesting in topic types

For topic modules, the last position in the content model is typically a reference to the *topictype*-info-types group. Document types shells can control how topics are allowed to nest by redefining the group. If topic nesting is hard coded in the structural module, and cannot be modified from the document-type shell, the *topictype*-info-types group is not needed.

For example, the vocabulary module for the `<concept>` structural type uses the group `concept-info-types`; this group is given a default value, and then referenced from the content model for the `<concept>` element type:

```
<xs:group name="concept-info-types">
  <xs:choice>
    <xs:group ref="concept"/>
    <xs:group ref="info-types"/>
  </xs:choice>
</xs:group>

<xs:group name="concept.content">
  <xs:sequence>
    <xs:group ref="title"/>
    <!-- ...other elements, such as shortdesc and body, and then... -->
    <xs:group ref="related-links" minOccurs="0"/>
    <xs:group ref="concept-info-types" minOccurs="0" maxOccurs="unbounded"/>
```

```
    </xs:sequence>
  </xs:group>
```

2.6.5.5 XML Schema: Coding requirements for attribute domain modules

An attribute domain vocabulary module declares one new attribute specialized from either the `@props` or `@base` attribute.

All vocabulary and constraint modules must document their `@domains` attribute contribution. The OASIS grammar files use a `<dita:domainsModule>` element to document the contribution; this element is used consistently to make it easy to find values when creating a document type shell. An XML comment or `<xs:appinfo>` element can also be used.

An attribute domain consists of one file. The file must have a single attribute group definition that contains the definition of the attribute itself, where the attribute group is named *attname*Att-d-attribute.

For example, the `@deliveryTarget` attribute is defined with the following attribute group:

```
<xs:attributeGroup name="deliveryTargetAtt-d-attribute">
  <xs:attribute name="deliveryTarget" type="xs:string">
    <!-- Documentation for attribute -->
  </xs:attribute>
</xs:attributeGroup>
```

2.6.5.6 XML Schema: 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.

All vocabulary and constraint modules must document their `@domains` attribute contribution. The OASIS grammar files use a `<dita:domainsModule>` element to document the contribution; this element is used consistently to make it easy to find values when creating a document type shell. An XML comment or `<xs:appinfo>` element can also be used.

For each vocabulary module with a content model or attributes to be constrained, there must be an `<xs:redefine>` element that references the vocabulary module. Within the `<xs:redefine>` element, for each element type content model to be constrained, an `<xs:group>` element is needed with the name *element*.content. Also within the `<xs:redefine>` element, for each attribute set to be constrained, an `<xs:attributeGroup>` element is needed with the name *element*.attributes. The constrained model is defined inside of the `<xs:group>` or `<xs:attributeGroup>`.

> **Note:** This means that when adding a constraint module to an existing document-type shell, you must remove any `<xs:include>` elements that refer to the XSD module to which the redefine is applied. For example, to redefine groups defined in `commonElementsMod.xsd`, you must remove any `<xs:include>` reference to `commonElementsMod.xsd`.

Because the constraint module includes the module that it modifies, only one constraint module can be used per vocabulary module (otherwise the module being constrained would be included multiple times). If multiple constraint modules are needed for a single vocabulary module, they must be combined into a single XSD module. For example, when combining existing constraint modules for `<p>` and `<div>`, a single module must be created that combines the `<xs:group>` and `<xs:attributeGroup>` constraints from existing modules inside a single `<xs:redefine>` reference to `commonElementsMod.xsd`.

When constraining a list of elements provided by a domain, there must be a group that lists the subset of domain elements in a constraints module. The group name *SHOULD* be named "*qualifierdomain*-c-*tagname*" where *qualifier* is a description for the constraint module, *domain* is the name of the domain, map, or topic being constrained, and *tagname* is the name of the extension element being restricted.

Example: Structural constraint module

The following code fragment shows how the `<topic>` element can be constrained to disallow the `<body>` element. This `<xs:redefine>` element is located in a constraint module that references the file `topicMod.xsd`, which means that a document-type shell using this constraint would reference this module *instead of* referencing `topicMod.xsd` (it would not reference both).

```
<xs:redefine schemaLocation="urn:oasis:names:tc:dita:xsd:topicMod.xsd:1.2">
  <xs:group name="topic.content">
    <xs:sequence>
      <xs:sequence>
        <xs:group ref="title"/>
        <xs:group ref="titlealts" minOccurs="0"/>
        <xs:choice minOccurs="1" >
          <xs:group ref="shortdesc" />
          <xs:group ref="abstract" />
        </xs:choice>
        <xs:group ref="prolog" minOccurs="0"/>
        <!--<xs:group ref="body" minOccurs="0"/>-->
        <xs:group ref="related-links" minOccurs="0"/>
        <xs:group ref="topic-info-types" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:sequence>
  </xs:group>
</xs:redefine>
```

For a more complete example, see `strictTaskbodyConstraintMod.xsd`, delivered with the DITA 1.3 grammar files.

Example: Domain constraint module

The following code fragment shows how the highlighting domain can be constrained to limit the elements that are available in the domain to only `<b>` and `<i>`.

```
<xs:group name="basicHighlight-c-ph">
  <xs:choice>
    <xs:element ref="b"/>
    <xs:element ref="i"/>
  </xs:choice>
</xs:group>
```

3 Language Reference: Base edition

The language reference portion of the DITA specification contains a topic for each DITA element. The topic defines the element, its inheritance hierarchy, and provides examples of usage. This portion of the DITA specification also includes information about DITA attributes.

3.1 Base DITA elements, A to Z

This section provides links to all of the base DITA elements in alphabetical order.

3.2 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.

3.2.1 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 *DITA topics (26)*.

3.2.1.1 <topic>

The `<topic>` element is the top-level DITA element for a single-subject topic or article.

Content models

See *appendix (450)* for information about this element in OASIS document type shells.

Inheritance

- topic/topic

Example

```
<topic id="topic">
 <title>Some little topic</title>
 <body>
  <p>Here's a little topic.</p>
  <ul>
   <li>Some item</li>
   <li>Another item</li>
  </ul>
 </body>
</topic>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (with a narrowed definition of `@id`, given below), *Architectural attribute group (350)*, and *outputclass*.

@id (REQUIRED)

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.

3.2.1.2 <title>

The `<title>` element contains a heading or label for the main parts of a topic: the topic as a whole, sections, examples, and labelled content such as figures and tables. The element also can be used to provide a title for a map or a relationship table; when used in a relationship table, the title typically is used as a authoring convenience and is not rendered for display to an end user.

Content models

See *appendix (449)* for information about this element in OASIS document type shells.

Inheritance

- topic/title

Example

```
<topic id="topicid">
  <title rev="121212">Monitoring the patient following the procedure
    <draft-comment>deleted the word "during" and added the word "following"</draft-comment>
</title>
  <body>
    <!-- ... -->
    <fig id="fig-01">
      <title rev="121212">Patient-monitoring procedure flow following procedures
        <draft-comment>deleted the word "during" and added
             the word "following"</draft-comment></title>
    <!-- ... -->
    </fig>
    <!-- ... -->
  </body>
</topic>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (without the Metadata attribute group), `@base` and `@rev` from the *Metadata attribute group (348)*, and *outputclass*.

3.2.1.3 <titlealts>

The `<titlealts>` element allows the insertion of alternate titles, such as titles that should be used in creating a table of contents for navigation or a title specific to search results. When the `<titlealts>` element is absent, the `<title>` element is used for all title purposes.

Content models

See *appendix (449)* for information about this element in OASIS document type shells.

Inheritance

- topic/titlealts

Example

```
<task id="progexample">
   <title>Example of Required Programming</title>
      <titlealts><navtitle>Programming Example</navtitle></titlealts>
   <taskbody> . . . </taskbody>
</task>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.2.1.4 <searchtitle>

The `<searchtitle>` element is used to specify a title that is displayed by search tools that locate the topic. This is most 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.

When a topic is rendered as XHTML, the contents of the `<searchtitle>` will typically appear in the XHTML's title element, which used in the result summary for many search engines. This element might not be supported for output formats that do not support distinct search titles for topics.

Content models

See *appendix (441)* for information about this element in OASIS document type shells.

Inheritance

"- topic/searchtitle " when used in topics, and "- map/searchtitle " when used in maps.

Example

In the following example, the general title "Programming Example" is likely very 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 that case, "Example of basic programming in XSLT" will be much more helpful.

```
<task id="progexample">
 <title>Programming Example</title>
  <titlealts><searchtitle>Example of basic
            programming in XSLT</searchtitle></titlealts>
 <taskbody> . . . </taskbody>
</task>
```

When `<searchtitle>` is used in maps, the element provides a new search title for the topic when used in a specific context. For example, the if the following map 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="progexample.dita">
  <topicmeta>
    <navtitle>Programming example</navtitle>
    <searchtitle>Example of programming in XSLT</searchtitle>
  </topicmeta>
</topicref>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.2.1.5 <navtitle>

The navigation title (`<navtitle>`) element is one of a set of alternate titles that can be included inside the `<titlealts>` element. This navigation title might differ from the first level heading that shows in the main browser window. Use `<navtitle>` when the actual title of the topic isn't appropriate for use in a table of contents, navigation pane, or online content (for example, because the actual title is too long). Beginning with DITA 1.2, the `<navtitle>` element is also available in the `<topicmeta>` element in a `<topicref>` in a map, and its use is preferred over the `<topicref>` element's `@navtitle` attribute.

When `<navtitle>` is used in a map, it functions in the same way as the `@navtitle` attribute; both are used to specify a navigation title for the target of the `<topicref>` element. That is, the title itself will only be used as an actual navigation title when the title is locked; the title is locked when the `<topicref>` element sets the attribute `locktitle="yes"`. If the title is not locked, processing systems will typically retrieve the current title from the target topic, looking first for a `<navtitle>` element and second for the general `<title>`.

When both a `<navtitle>` element and a `@navtitle` attribute are specified, the `<navtitle>` element *SHOULD* be used.

Because the `<navtitle>` element is available within `<topicmeta>`, and `<topicmeta>` is used in many different contexts, it is possible that `<navtitle>` can be specified in contexts where a navigation title does not make sense (for example, on the `<topicgroup>` element). In those situations, the `<navtitle>` element has no defined purpose.

Content models

See *appendix (433)* for information about this element in OASIS document type shells.

Inheritance

- topic/navtitle

Example

```
<task id="progexample">
 <title>Publishing a DITA information set in PDF</title>
  <titlealts><navtitle>Publishing in PDF</navtitle></titlealts>
 <taskbody><context>Some text</context></taskbody>
</task>
```

Figure 39: <navtitle> sample in a topic

In this sample, the first title is not locked, and will generally be replaced with a title retrieved from `a.dita`. The second title is locked, and will be displayed when this map is used as a basis for navigation.

```
<map xml:lang="en">
  <title>This is a sample map</title>
  <topicref href="a.dita">
    <topicmeta>
      <navtitle>Title of A</navtitle>
    </topicmeta>
  </topicref>
  <topicref href="b.dita" locktitle="yes">
    <topicmeta>
      <navtitle>Short Title for B</navtitle>
    </topicmeta>
  </topicref>
</map>
```

Figure 40: <navtitle> samples in a map

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.2.1.6 <shortdesc>

The `<shortdesc>` element provides a short description of the topic. The short description, which represents the purpose or theme of the topic, is also intended to be used as a link preview and for search results. The element can occur both in topics and maps.

The short description should be a single, concise paragraph containing one or two sentences of no more than 50 words.

Short descriptions in topics

The content of the `<shortdesc>` element *SHOULD* be rendered as the initial paragraph of the topic.

Use the `<shortdesc>` element when the first paragraph of topic content is simple enough to be suitable for use as a link preview or for summaries. Otherwise use the `<abstract>` element to provide richer content around the `<shortdesc>`. See the *abstract (187)* description for more information about the behavior of `<shortdesc>` in an abstract.

While the `<shortdesc>` element is optional, it is recommended that topics contain this element. When a topic contains only one paragraph, it is preferable to include the text in the `<shortdesc>` and omit the topic body or leave it empty.

Short descriptions in maps

The `<shortdesc>` element can be used in DITA maps and 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 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, processors *SHOULD* use the content of the `<shortdesc>` element in the DITA topic when they render the topic itself, unless the `@copy-to` attribute is specified on the topic reference to the element.

When a `<shortdesc>` element applies to an entire DITA map, it serves as description only.

Content models

See *appendix (443)* for information about this element in OASIS document type shells.

Inheritance

"- topic/shortdesc " when used in topics, and "- map/shortdesc " when used in maps.

Examples

The following example demonstrates the use of a stand-alone `<shortdesc>` inside of a concept topic.

```
<concept id="concept">
 <title>Introduction to Bird Calling</title>
 <shortdesc>If you wish 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>
 <conbody>
   <p>Bird calling requires learning:</p>
   <ul>
    <li>Popular and classical bird songs</li>
    <li>How to whistle like a bird</li>
   </ul>
 </conbody>
</concept>
```

Example: short description within a map

```
<topicref href="myThing.dita">
  <topicmeta>
    <navtitle>Navigation title for my topic</navtitle>
    <shortdesc>A description of myThing that is specific to this context.</shortdesc>
  </topicmeta>
</topicref>
<topicref href="http://www.example.org" scope="external">
  <topicmeta>
    <navtitle>Example website</navtitle>
    <shortdesc>The example.org address is often used in examples</shortdesc>
  </topicmeta>
</topicref>
```

Example: with phrase-level short description

```
<abstract>The abstract is being used to provide more complex content.
    <shortdesc>The shortdesc 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 shortdesc must be directly contained by the abstract. The abstract can put text around the shortdesc.

Preview/summary output

The shortdesc must be directly contained by the abstract.

Example: with block-level short description

```
<abstract><p>The abstract is being used to provide more complex content.</p>
    <shortdesc>The shortdesc must be directly contained by the abstract.</shortdesc>
<p>The abstract can put text around the shortdesc.</p>
</abstract>
```

Topic output

The abstract is being used to provide more complex content.

The shortdesc must be directly contained by the abstract.

The abstract can put text around the shortdesc.

Preview/summary output

The shortdesc must be directly contained by the abstract.

Example: with multiple short descriptions

```
<abstract>The abstract is being used to provide more complex content.
    <shortdesc>The shortdesc must be directly contained by the abstract.</shortdesc>
    <p>The abstract can put text around the shortdesc.</p>
    <shortdesc>There can be more than one shortdesc.</shortdesc>
</abstract>
```

Topic output

The abstract is being used to provide more complex content. The shortdesc must be directly contained by the abstract.

The abstract can put text around the shortdesc.

There can be more than one shortdesc.

Preview/summary output

The shortdesc must be directly contained by the abstract. There can be more than one shortdesc.

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)* and *outputclass*.

3.2.1.7

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.

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>`.

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.

Content models

See *appendix* *(409)* for information about this element in OASIS document type shells.

Inheritance

- topic/abstract

Example: with phrase-level short description

```
<abstract>The abstract is being used to provide more complex content.
    <shortdesc>The shortdesc 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 shortdesc must be directly contained by the abstract. The abstract can put text around the shortdesc.

Preview/summary output

The shortdesc must be directly contained by the abstract.

Example: with block-level short description

```
<abstract><p>The abstract is being used to provide more complex content.</p>
    <shortdesc>The shortdesc must be directly contained by the abstract.</shortdesc>
<p>The abstract can put text around the shortdesc.</p>
</abstract>
```

Topic output

The abstract is being used to provide more complex content.

The shortdesc must be directly contained by the abstract.

The abstract can put text around the shortdesc.

Preview/summary output

The shortdesc must be directly contained by the abstract.

Example: with multiple short descriptions

```
<abstract>The abstract is being used to provide more complex content.
    <shortdesc>The shortdesc must be directly contained by the abstract.</shortdesc>
```

```
    <p>The abstract can put text around the shortdesc.</p>
    <shortdesc>There can be more than one shortdesc.</shortdesc>
</abstract>
```

Topic output

The abstract is being used to provide more complex content. The shortdesc must be directly contained by the abstract.

The abstract can put text around the shortdesc.

There can be more than one shortdesc.

Preview/summary output

The shortdesc must be directly contained by the abstract. There can be more than one shortdesc.

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)* and *outputclass*.

3.2.1.8 <body>

The `<body>` element is the container for the main content of a `<topic>`.

Content models

See *appendix* *(411)* for information about this element in OASIS document type shells.

Inheritance

- topic/body

Example

```
<topic>
<title>Sample title</title>
<prolog><!-- metadata here --></prolog>
<body> <!-- Body content goes here --> </body>
</topic>
```

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)* and *outputclass*.

3.2.1.9 <bodydiv>

The `<bodydiv>` element is used to contain informal blocks of information within the body of a topic. The `<bodydiv>` element is specifically designed to be a grouping element, without any explicit semantics, other than to organize subsets of content into logical groups that are not intended or should not be contained as a topic. As such, it does not contain an explicit title to avoid enabling the creation of deeply nested content that would otherwise be written as separate topics. For content that requires a title, use a `<section>` element or a nested topic.

The `<bodydiv>` element can nest itself, which means that it can be used as a specialization base to create structured information within a body. 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` 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.

Content models

See *appendix* *(412)* for information about this element in OASIS document type shells.

Inheritance

- topic/bodydiv

Example

```
<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>
```

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)* and *outputclass*.

3.2.1.10 <related-links>

The related information links of a topic (`<related-links>` element) are stored in a special section following the body of the topic. After a topic is processed into its final output form, the related links are usually 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 are typically displayed together with links generated based on a map context.

The following lists contains processing expectations for 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.

Content models

See *appendix* *(439)* for information about this element in OASIS document type shells.

Inheritance

- topic/related-links

Example

The following indicates that the two external links are always applicable to this topic, regardless of how the topic is used.

```
<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>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (apart from `@href`), *The role and otherrole attributes (373)*, and *outputclass*.

3.2.1.11 <dita>

The `<dita>` element provides a top-level container for multiple topics when you create documents using the ditabase document type. The `<dita>` element lets you create any sequence of topics, and the ditabase document type lets you nest these topic types inside each other. The `<dita>` element has no particular output implications; it simply allows you to create multiple topics of different types at the same level in a single document.

Content models

See *appendix (417)* for information about this element in OASIS document type shells.

Inheritance

Not a specializable DITA element.

Example

```
<dita>
  <concept id="batintro"><title>Intro to bats</title><conbody><!-- ... --></conbody></concept>
  <reference id="batparts"><title>Parts of bats</title><refbody><!-- ... --></refbody></reference>
  <task id="batfeeding"><title>Feeding a bat</title><taskbody><!-- ... --></taskbody></task>
  <task id="battraining"><title>Training a bat</title><taskbody><!-- ... --></taskbody></task>
  <task id="batcleanup"><title>Cleaning a bat</title><taskbody><!-- ... --></taskbody></task>
</dita>
```

Attributes

The following attributes are available on this element: `@xmlns:ditaarch` and `@DITAArchVersion` from *Architectural attribute group (350)*, *Localization attribute group (349)* and *Debug attribute group (350)*.

3.2.2 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.

3.2.2.1 <alt>

The `<alt>` element provides alternate text for an image. It is equivalent to the `@alt` attribute on the `<image>` element; since the `@alt` attribute is deprecated, use the `<alt>` element instead. The `<alt>` element can be more easily edited or translated than the `@alt` attribute.

Content models

See *appendix (409)* for information about this element in OASIS document type shells.

Inheritance

- topic/alt

Example

The markup for alternate text within an image looks like this:

```
<image href="tip-ing.jpg">
  <alt>Here's a Tip!</alt>
</image>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.2.2.2 <cite>

The `<cite>` element is used when you need a bibliographic citation. It specifically identifies the title of the resource.

Though citations are often set apart from surrounding text, such as through italics, rendering of the `<cite>` element is left up to implementations.

Content models

See *appendix (413)* for information about this element in OASIS document type shells.

Inheritance

- topic/cite

Example

```
<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>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, and *@keyref (360)*.

3.2.2.3 <dd>

The `<dd>` element contains the description of a term in a definition list entry (`<dlentry>`).

Content models

See *appendix (416)* for information about this element in OASIS document type shells.

Inheritance

- topic/dd

Example

```
<dl>
<dlentry>
<dt>Bytes returned</dt>
<dd>The number of bytes of data returned.</dd>
</dlentry>
<dlentry>
<dt>Bytes available</dt>
<dd>The number of bytes of data available to be returned.</dd>
</dlentry>
<dlentry><dt>Handle</dt>
<dd>The returned handle value</dd>
</dlentry>
</dl>
```

Figure 41: Simple definition list example

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.2.2.4 `<desc>`

The `<desc>` element contains the description of the current element.

The following list outlines some usages 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; processors *MAY* choose to display this text as hover help for a link.

`<object>`
  Provides alternate content to use when the context does not permit displaying the object.

Content models

See *appendix (417)* for information about this element in OASIS document type shells.

Inheritance

- topic/desc

Example

```
<fig><title>The Handshake</title>
<desc>This image shows two hands clasped in a formal,
business-like handshake.</desc>
<image href="handshake.jpg">
 <alt>The handshake</alt>
</image>
</fig>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.2.2.5 `<ddhd>`

The `<ddhd>` element provides an optional heading or title for a column of descriptions or definitions in a definition list (`<dl>`).

Content models

See *appendix (417)* for information about this element in OASIS document type shells.

Inheritance

- topic/ddhd

Example

```
<dl>
 <dlhead>
  <dthd>Image File View Selection</dthd>
  <ddhd>Resulting Information</ddhd>
 </dlhead>
 <dlentry>
  <dt>File Type</dt>
  <dd>Image's file extension</dd>
 </dlentry>
 <dlentry>
  <dt>Image Class</dt>
  <dd>Image is raster, vector, metafile 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>
```

Figure 42: Definition list with a heading

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.2.2.6 <div>

The `<div>` element is used to organize subsets of content into logical groups that are not intended to be or should not be contained as a topic.

The `<div>` element is designed to be a grouping element; it does not imply any explicit semantics or contain an explicit title. This avoids enabling the creation of deeply-nested content that would otherwise be written as separate topics. If the content requires a title, use a `<section>` element, a nested `<topic>`, or possibly a `<fig>` element.

Content models

See *appendix (418)* for information about this element in OASIS document type shells.

Inheritance

- topic/div

Example

In the following example, a `<div>` 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 `<div>` element, the content could not be grouped for content referencing since the start and end elements are of different types.

Figure 43: Using <div> for grouping

In the following example, `<div>` is used as the basis for specializing a new domain element, `<pullquote>`:

```
<!ENTITY % pullquote.content
  "(%div.cnt;)*"
>
<!ENTITY % pullquote.attributes
            "%univ-atts;
             outputclass
                       CDATA
                                 #IMPLIED"
>
<!ELEMENT pullquote     %pullquote.content;>
<!ATTLIST pullquote     %pullquote.attributes;>

<!ATTLIST pullquote       %global-atts;  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>
```

Figure 44: Using <div> for specialization

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.2.2.7 <dl>

A `<dl>` element is a list of terms and corresponding definitions. The term (`<dt>`) is usually flush left. The description or definition (`<dd>`) is usually either indented and on the next line, or on the same line to the right of the term. However, actual rendering is up to the rendering engine.

An optional heading for the terms and definitions can be provided using the `<dlhead>` element, which contains header elements for those columns. The default formatting for the `<dlhead>` generally looks like a table with a heading row, but this is also up to the rendering engine.

Content models

See *appendix (418)* for information about this element in OASIS document type shells.

Inheritance

- topic/dl

Examples

```
<dl>
<dlentry>
<dt>Bytes returned</dt>
<dd>The number of bytes of data returned.</dd>
</dlentry>
<dlentry>
<dt>Bytes available</dt>
<dd>The number of bytes of data available to be returned.</dd>
</dlentry>
<dlentry><dt>Handle</dt>
<dd>The returned handle value</dd>
</dlentry>
</dl>
```

Figure 45: Simple definition list example

```
<dl>
 <dlhead>
  <dthd>Image File View Selection</dthd>
  <ddhd>Resulting Information</ddhd>
 </dlhead>
 <dlentry>
  <dt>File Type</dt>
  <dd>Image's file extension</dd>
 </dlentry>
 <dlentry>
```

```
 <dt>Image Class</dt>
 <dd>Image is raster, vector, metafile 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>
```

Figure 46: Definition list with a heading

Rendering of definition lists will vary by application and by display format. Processors *MIGHT* render the second example as follows.

Image File View Selection	Resulting Information
File Type	Image's file extension
Image Class	Image is raster, vector, metafile or 3D
Number of pages	Number of pages in the image
Fonts	Names of the fonts contained within a vector image

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, *compact*, and *spectitle*.

3.2.2.8 <dlentry>

The <dlentry> element groups a single entry in a definition list (<dl>). The <dlentry> element includes a term (<dt>) and one or more definitions or descriptions (<dd>) of that term.

Content models

See *appendix (419)* for information about this element in OASIS document type shells.

Inheritance

- topic/dlentry

Example

```
<dl>
<dlentry>
<dt>Bytes returned</dt>
<dd>The number of bytes of data returned.</dd>
</dlentry>
<dlentry>
<dt>Bytes available</dt>
<dd>The number of bytes of data available to be returned.</dd>
</dlentry>
<dlentry><dt>Handle</dt>
<dd>The returned handle value</dd>
</dlentry>
</dl>
```

Figure 47: Simple definition list example

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.2.2.9 <dlhead>

The `<dlhead>` element contains optional headings for the term and description columns in a definition list (`<dl>`). The definition list heading might contain a heading for the column of terms (`<dthd>`) and a heading for the column of descriptions (`<ddhd>`).

Content models

See *appendix (419)* for information about this element in OASIS document type shells.

Inheritance

- topic/dlhead

Example

```
<dl>
 <dlhead>
  <dthd>Image File View Selection</dthd>
  <ddhd>Resulting Information</ddhd>
 </dlhead>
 <dlentry>
  <dt>File Type</dt>
  <dd>Image's file extension</dd>
 </dlentry>
 <dlentry>
  <dt>Image Class</dt>
  <dd>Image is raster, vector, metafile 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>
```

Figure 48: Definition list with a heading

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.2.2.10 <dt>

The `<dt>` element contains a term in a definition list entry (`<dlentry>`).

Content models

See *appendix (419)* for information about this element in OASIS document type shells.

Inheritance

- topic/dt

Example

```
<dl>
<dlentry>
<dt>Bytes returned</dt>
<dd>The number of bytes of data returned.</dd>
</dlentry>
<dlentry>
<dt>Bytes available</dt>
<dd>The number of bytes of data available to be returned.</dd>
</dlentry>
```

```
<dlentry><dt>Handle</dt>
<dd>The returned handle value</dd>
</dlentry>
</dl>
```
Figure 49: Simple definition list example

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, and *@keyref (360)*.

3.2.2.11 <draft-comment>

The `<draft-comment>` element is designed to facilitate review and discussion of topic contents within the marked-up content. Use the `<draft-comment>` element to ask a question or to make a comment that you want others to review. To indicate the source of the draft comment or the status of the comment, use the `@author`, `@time`, or `@disposition` attributes.

Processing systems *SHOULD* provide a mechanism that causes the content of this element to be rendered in draft output only. By default, processors *SHOULD* strip them out to prevent publishing internal comments by mistake.

Content models

See *appendix (419)* for information about this element in OASIS document type shells.

Inheritance

- topic/draft-comment

Example

```
<draft-comment author="EBP">Where's the usage information for this
section?</draft-comment>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (with a narrowed definition for `@translate`, given below), *outputclass*, and the attributes defined below.

@translate

Indicates whether the content of the element should be translated or not. For this element the default value is "no". Setting to "yes" will override the default. 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. Available values are:

no

The content of this element is not translateable.

yes

The content of this element is translateable.

-dita-use-conref-target

See *Using the -dita-use-conref-target value (362)* for more information.

@author

Designates the originator of the draft comment.

@time

Describes when the draft comment was created.

@disposition

> Status of the draft comment. Prior to DITA 1.2, this attribute was limited to the following values: issue, open, accepted, rejected, deferred, duplicate, reopened, unassigned, or completed.

3.2.2.12 <dthd>

The `<dthd>` element provides an optional heading for the column of terms in a definition list (`<dl>`).

Content models

See *appendix (420)* for information about this element in OASIS document type shells.

Inheritance

- topic/dthd

Example

```
<dl>
 <dlhead>
  <dthd>Image File View Selection</dthd>
  <ddhd>Resulting Information</ddhd>
 </dlhead>
 <dlentry>
  <dt>File Type</dt>
  <dd>Image's file extension</dd>
 </dlentry>
 <dlentry>
  <dt>Image Class</dt>
  <dd>Image is raster, vector, metafile 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>
```

Figure 50: Definition list with a heading

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.2.2.13 <example>

The `<example>` element is a section that contains examples that illustrate or support the current topic.

Use `<example>` to contain both sample code (or similar artifacts) and the discussion that illustrates the sample. For example, a DITA topic about programming code could use the `<example>` element to contain both the sample code and the text that describes the code. Similarly, a sample that included preformatted text might use the `<pre>` element along with a paragraph or list to elaborate upon sections of that text.

Content models

See *appendix (421)* for information about this element in OASIS document type shells.

Inheritance

- topic/example

Example

```
<example id="example">
  <title>Example</title>
  <codeblock>&lt;p&gt;Example of the p element&lt;/p&gt;</codeblock>
</example>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, and *spectitle*.

3.2.2.14 <fig>

The `<fig>` element is a figure (sometimes called an "exhibit") with an optional title for a wide variety of content. A figure commonly contains an image or artwork, but it can contain several kinds of text objects as well. A title is placed inside the figure to provide a caption that describes the content.

Use the `<desc>` element to provide a longer description of the figure following the title.

Content models

See *appendix (422)* for information about this element in OASIS document type shells.

Inheritance

- topic/fig

Example

```
<fig expanse="column">
  <title>The Handshake</title>
  <image href="handshake.jpg" placement="break">
    <alt>The Handshake</alt>
  </image>
</fig>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Display attribute group (355)*, *outputclass*, and *spectitle*.

3.2.2.15 <figgroup>

The `<figgroup>` element is used primarily for specialization, in order to create segments within a figure. The element can nest itself, which allows it to create complex specialized structures (such as the nestable groups of syntax within a syntax diagram). Figure groups can be used to contain multiple cross-references, footnotes or keywords.

Content models

See *appendix (422)* for information about this element in OASIS document type shells.

Inheritance

- topic/figgroup

Example

```
<fig>
  <title>Sample complex figure</title>
  <figgroup>
    <title>First group</title>
    <ph>These elements</ph>
    <ph>are grouped together</ph>
```

```
      <ph>for some purpose</ph>
  </figgroup>
  <figgroup>
     <title>Second group</title>
     <data name="MetaItem" value="13"/>
     <data name="MetaThing" value="31"/>
     <ph>These elements</ph>
     <ph>are grouped with associated metadata</ph>
  </figgroup>
</fig>
```

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)* and *outputclass*.

3.2.2.16 <fn>

The `<fn>` element is a footnote used to annotate text with notes that are inappropriate for inline inclusion. It is also used to indicate the source for facts or other material used in the text.

Footnote content is skipped at the place where it was entered and rendered elsewhere, according to these rules:

- A footnote with no specified `@id` attribute is a *single-use footnote*. Upon output, it generates a number as a superscript "callout" that is linked to the placement of the footnote, such as at the bottom of the immediate printed page or at the end of an online article. If a character is specified in the `@callout` attribute for the footnote, that character is used as the superscript "callout" that is linked to the placement of the footnote.

- A footnote that specifies an `@id` attribute is a *use-by-reference footnote*. Upon output, it does not appear anywhere unless it has been referenced using an `<xref>` with the `@type` attribute set to "fn".

- Ordinarily, a footnote in one topic can't be referenced in another topic. The previous behaviors are local to each topic. But by using the conref mechanism, you can create a new copy of another topic's footnote within the local topic where it will then follow these behaviors:

  - If you use `<fn conref="file.dita#topic/thatid"></fn>` all by itself, the result will be the same as the single-use footnote entered literally in the same location. That is, it creates a local copy of the footnote with no local `@id` attribute, so it uses the behavior from the first bullet above.

  - If you use `<fn conref="file.dita#topic/thatid" id="thisid"></fn>`, followed by `<xref href="#thistopic/thisid" type="fn"/>`, the result will be the same as the use-by-reference model described in the second bullet. That is, the `<fn>` element creates a local copy of the footnote with an id of "thisid"; that local copy is then referenced by the `<xref>` element.

The details of footnote processing and styling are implementation and/or stylesheet dependent. For example, a tool that renders DITA as PDF might lack support for the `@callout` attribute, or footnotes might be collected as endnotes for certain types of publications.

Content models

See *appendix* *(422)* for information about this element in OASIS document type shells.

Inheritance

- topic/fn

Example

The first example is of a **single-use** footnote. It uses a simple `<fn>` element, with no `@id` and no `@callout` attribute. In that case, markup such as the following:

```
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.
```

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] ---

The second example is a **single-use** footnote that uses a `@callout` attribute. It is marked up as follows:

```
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.
```

That DITA markup might produce output similar to the following:

> The memory storage capacity of the computer is 2 GB[#] with error correcting support.
>
> . . . . . .
>
> [#] A GB (gigabyte) is equal to 1000 million bytes
>
> ----- [bottom of page] ---

The third example is a **use-by-reference** footnote. It uses an `@id` on a footnote, and then references that `@id` multiple times. The DITA markup looks like this:

```
I like pets. <fn id="reuse-fn">This is the name of an animal.</fn>
At my house, I have a dog<xref href="#topic/reuse-fn" type="fn"/>, a
cat<xref href="#topic/reuse-fn" type="fn"/>, and a
llama<xref href="#topic/reuse-fn" type="fn"/>.
```

and might produce output similar to the following:

> I like pets. At my house, I have a dog[1], a cat[1], and a llama[1].
>
> . . . . . .
>
> [1]This is the name of an animal.
>
> ----- [bottom of page] ---

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, and the attribute defined below.

@callout

Specifies what character is used for the footnote link, for example a number or an alpha character. The attribute can also specify a short string of characters. When no `@callout` value is specified, footnotes are typically numbered.

@datatype

Available for describing the type of data contained in the value attribute for this metadata element. The default value is the empty string "".

3.2.2.17 <image>

Use the `<image>` element to include artwork or images in a DITA topic.

The `<image>` element has an optional `@placement` attribute that indicates whether the included graphic or artwork is rendered inline (a button or icon) or on a separate line for a larger image. There also are attributes that indicate the size to which the included graphic or artwork is scaled.

An image element specifies an `@href` attribute, a `@keyref` attribute, or both. When both `@keyref` and `@href` are specified, the `@href` is used as a fallback when the key reference cannot be resolved. The image addressed by the `@keyref` or `@href` is brought into the main flow of the content as rendered.

To make the intent of the image more accessible for users using screen readers or text-only readers, authors should include a description of the image content in the `<alt>` element.

Content models

See *appendix (425)* for information about this element in OASIS document type shells.

Inheritance

- topic/image

Example

```
<image href="bike.gif" placement="break">
  <alt>Two-wheeled bicycle</alt>
</image>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, *`@keyref` (360)*, and the attributes defined below.

@href

Provides a reference to the image. See *The href attribute (359)* for detailed information on supported values and processing implications.

@scope

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 (362)*. See *The scope attribute (373)* for more information on values.

@height

Indicates the vertical dimension for the resulting image display. If necessary, the image *SHOULD* be scaled to the specified size. 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". If a height value is specified and no width value is specified, the width *SHOULD* be scaled by the same factor as the height. 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.

@width

Indicates the horizontal dimension for the resulting image display. If necessary, the image *SHOULD* be scaled to the specified size. 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". If a width value is specified and no height value is specified, the height *SHOULD* be scaled 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.

@align

Controls 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 this image's `@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

Allow an image to be scaled up or down to fit within available space. Allowable values are yes, no, and *-dita-use-conref-target (362)*. If, for a given image, 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 (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

Indicates whether an image should be displayed inline or separated from the surrounding text. The default is inline. Allowable values are: inline, break, or and *-dita-use-conref-target (362)*.

@alt (DEPRECATED)

Alternative text that describes the image to provide accessibility to page readers or provides a text description when an image cannot be displayed by the user's software. The `@alt` attribute is deprecated; use the *alt (191)* element instead.

@longdescref (DEPRECATED)

A reference to a textual description of the graphic or object. This attribute supports creating accessible content. See *The href attribute (359)* for detailed information on supported values and processing implications. For examples of how this attribute is used in output, see this topic on *long descriptions*. NOTE: This attribute is deprecated in favor of the *longdescref (206)* subelement to this element.

3.2.2.18 <keyword>

The `<keyword>` element identifies a keyword or token, such as a single value from an enumerated list, the name of a command or parameter, product name, or a lookup key for a message.

A keyword contains any text that has a unique or key-like value, such as a product name. Where there is an element that has a better meaning for what you are describing, use that element. The `<keyword>` element is a generic element; use it when no other element applies. The `<keyword>` element can also be used to contain reusable text.

Specific markup recommendations:

- Use `<apiname>` for API names and `<cmdname>` for command names.
- Use *`<term>` (218)* to indicate what you are defining with inline paragraph definitions.
- Use *`<ph>` (214)* for general phrases when `<keyword>` is not appropriate.
- Use `<kwd>` to indicate programming keywords in syntax diagrams and syntax phrases.

Specialized elements derived from `<keyword>` might also have extended processing, such as different formatting or automatic indexing.

All `<keyword>` or `<indexterm>` elements in the `<keywords>` metadata element are considered part of the topic's metadata and should be processed as appropriate for the given output medium.

While the `<keyword>` element can be used inline, the *`<keywords>`* *(262)* element is not an inline element. The `<keywords>` element only appears in the `<topicmeta>` or `<prolog>` elements; it is used to specify keywords that apply to the topic.

Content models

See *appendix* *(428)* for information about this element in OASIS document type shells.

Inheritance

- topic/keyword

```
<p>The <keyword>assert</keyword> pragma statement allows messages to be passed
to the emulator, pre-compiler, etc..</p>
<p>The <keyword id="myProduct">AmazingProduct</keyword> can make use of
this feature to do really neat stuff.</p>
```

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)*, *outputclass*, and *@keyref* *(360)*.

3.2.2.19

A `<li>` element is a single item in an ordered (`<ol>`) or unordered (`<ul>`) list. When a DITA topic is rendered, numbers and alpha characters are typically displayed with list items in ordered lists, while bullets and dashes are typically displayed with list items in unordered lists.

Content models

See *appendix* *(429)* for information about this element in OASIS document type shells.

Inheritance

- topic/li

Example

```
<ul>
  <li>This is an item in an unordered list.</li>
  <li>This is another item in an unordered list.</li>
</ul>
```

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)* and *outputclass*.

3.2.2.20 <lines>

The `<lines>` element contains text where line breaks are significant but white space is not. It can be used to represent dialogs, poetry, or other text fragments where line breaks are significant. The `<lines>` element is similar to `<pre>` in that line breaks are preserved, but the font style is not typically set to monospace, and extra spaces inside the lines are not preserved.

Content models

See *appendix* *(430)* for information about this element in OASIS document type shells.

Inheritance

- topic/lines

Example

```
This is a sample of my favorite sonnet.
<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>
```

Though exact formatting will vary, the previous sample will typically be rendered as follows.

This is a sample of my favorite sonnet

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:
...

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Display attribute group (355)*, *outputclass*, *xml:space*, and *spectitle*.

3.2.2.21 <longdescref>

The `<longdescref>` element supports a reference to a text description of the graphic or object. This element replaces the deprecated `@longdescref` attribute on `<image>` and `<object>` elements.

Content models

See *appendix (431)* for information about this element in OASIS document type shells.

Inheritance

- topic/longdescref

Example

```
<image href="llama.jpg">
  <alt>Llama picture</alt>
  <longdescref href="my-pet-llama.dita"/>
</image>
```

Figure 51: <longdescref> which references a local DITA description

```
<image href="puffin.jpg">
  <alt>Puffin pigure</alt>
  <longdescref href="http://www.example.org/birds/puffin.html"
               scope="external"
               format="html"/>
</image>
```

Figure 52: <longdescref> which references an external description

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)*, *outputclass*, and *@keyref (360)*.

3.2.2.22 <longquoteref>

The `<longquoteref>` element provides a reference to the source of a long quote.

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 quote into a link, or do something else.

Content models

See *appendix (431)* for information about this element in OASIS document type shells.

Inheritance

- topic/longquoteref

Example

```
<p>A great person once said the following thing.
<lq>Examples are the key to any
specification.<longquoteref href="http://www.example.org/quotes" scope="external"/></lq></p>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)*, *outputclass*, and *@keyref (360)*.

3.2.2.23 <lq>

The `<lq>` is used to provide extended content quoted from another source. Use the quote element `<q>` for short, inline quotations, and long quote `<lq>` for quotations that are too long for inline use, following normal guidelines for quoting other sources. The `@href` and `@keyref` attributes are available to specify the source of the quotation. The `<longquoteref>` element is available for more complex references to the source of a quote.

Although rendering is left up to implementations, processors generally render `<lq>` as an indented block.

Content models

See *appendix (431)* for information about this element in OASIS document type shells.

Inheritance

- topic/lq

Example

```
<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>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>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with a narrowed definition for `@type`, given below), *outputclass*, and *@keyref (360)*, and the attributes defined below.

@reftitle

  The title of the document or topic being 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 *The type attribute (370)* for detailed information on the usual supported values and processing implications. The following attribute values are allowed (but deprecated) for backward compatibility:

external

> The `@href` is to a Web site. This value is deprecated in favor of use of the `@scope` and `@format` attributes.

internal

> The `@href` is to a DITA topic. This value is deprecated in favor of use of the `@scope` and `@format` attributes.

3.2.2.24 <note>

A `<note>` element contains information that expands on or calls attention to a particular point. This information is typically differentiated from the main text.

> **Tip:** Variant types of `<note>` (tip, caution, danger, restriction, etc.) can be indicated through values selected on the `@type` attribute. This `<note>` uses the attribute `type="tip"`

Content models

See *appendix (434)* for information about this element in OASIS document type shells.

Inheritance

- topic/note

Example

The following code sample demonstrates the use of 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>
```

The sample markup could produce output such as the following:

Tip: Thinking of a seashore, green meadow, or cool mountain overlook can help you to relax and be more patient.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, *spectitle*, and the attributes defined below.

@type

> Defines the type of a note. For example, if the note is a tip, the word **Tip** might be used to draw the reader's attention to it. Note that this differs from the `@type` attribute on many other DITA elements. See *The type attribute (370)* 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 (362)*.

@othertype

> Indicates an alternate note type, when the type is not available in the `@type` attribute value list. This value is used as the user-provided note title when the `@type` attribute value is set to "other".

3.2.2.25 <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.

The `<object>` element enables authors to include animated images, applets, plug-ins, ActiveX controls, video clips, and other multimedia objects in a topic.

Content models

See *appendix (434)* for information about this element in OASIS document type shells.

Inheritance

- topic/object

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>
 <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>
 <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>
```

```
<object
  id="E5123_026.mp4"
  width="300"
  height="300">
  <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 53: Object with reference to video using key reference on the <param> elements

```
<object
 classidkeyref="video_classid"
 codebasekeyref="video_codebase"
 datakeyref="cutkey370"
 height="280"
 width="370"
 id="cutkey370">
 <desc>A description of the task</desc>
 <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>
```

Figure 54: Object with indirect reference to a flash file and fallback @data value

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, 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 loadtimes 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

Key reference to the location of an object's data, as for `@data`. 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 for the purpose of the `@datakeyref` attribute. 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

Indicates the vertical dimension for the resulting object display. If necessary, the object is scaled to the specified size. 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". If a height value is specified and no width value is specified, the width will be scaled by the same factor as the height. 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.

@width

Indicates the horizontal dimension for the resulting object display. If necessary, the object is scaled to the specified size. 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". If a width value is specified and no height value is specified, the height will be scaled 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.

@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.

@longdescref (DEPRECATED)

A reference to a textual description of the graphic or object. This attribute supports creating accessible content. See *The href attribute (359)* for detailed information on supported values and processing implications. For examples of how this attribute is used in output, see this topic on *long descriptions*. NOTE: This attribute is deprecated in favor of the *longdescref (206)* subelement to this element.

3.2.2.26

The `<ol>` element includes a list of items sorted by sequence or order of importance.

Content models

See *appendix (434)* for information about this element in OASIS document type shells.

Inheritance

- topic/ol

Example

```
Here are the colors of the rainbow in order of appearance from top to bottom:
<ol>
<li>Red</li>
<li>Orange</li>
<li>Yellow</li>
<li>Green</li>
<li>Blue</li>
<li>Indigo</li>
<li>Violet</li>
</ol>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, *compact*, and *spectitle*.

3.2.2.27 <p>

The `<p>` element is a single paragraph containing a single main idea.

Content models

See *appendix (435)* for information about this element in OASIS document type shells.

Inheritance

- topic/p

Example

```
<p>This paragraph contains text that should be of
interest. It might give details about my company, or an explanation
of how my product is used, or it might be a short review of the book
I read last week.</p>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.2.2.28 <param>

The `<param>` (parameter) element specifies a set of values that might be required by an `<object>` at runtime. 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.

Content models

See *appendix (436)* for information about this element in OASIS document type shells.

Inheritance

- topic/param

Example

See *object (209)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* 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 *Using the -dita-use-conref-target value (362)* 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.

3.2.2.29 <ph>

The `<ph>` element is used to enclose a phrase for reuse or conditional processing (for example, when part of a paragraph applies to a particular audience). It can be used by specializations of DITA to create semantic markup for content at the phrase level, which then allows (but does not require) specific processing or formatting.

Content models

See *appendix (436)* for information about this element in OASIS document type shells.

Inheritance

- topic/ph

Example

```
<p>This was not changed. <ph rev="v5r2">This was updated.</ph> This was not.</p>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, and *@keyref (360)*.

3.2.2.30 <pre>

The `<pre>` element contains text for which all line breaks and spaces are preserved. It is typically presented in a monospaced font. Do not use `<pre>` when a more semantically specific element is appropriate, such as `<codeblock>`.

Content models

See *appendix (437)* for information about this element in OASIS document type shells.

Inheritance

- topic/pre

Example

The following example will preserve all line breaks and white space.

```
<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>
```

The rendered result will differ depending on the processor that is rendering your DITA content. It will generally look something like this:

```
        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.
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Display attribute group (355)*, *outputclass*, *xml:space*, and *spectitle*.

3.2.2.31 <q>

The `<q>` element includes content quoted from another source. This element is used for short quotes that are displayed inline. Use the long quote element (<lq>) for quotations that should be set off from the surrounding text or that contain multiple paragraphs.

Authors should not add quote punctuation manually when using the `<q>` element. Processors that render the `<q>` element *SHOULD* add appropriate styling, such as locale-specific quotation marks.

Content models

See *appendix (438)* for information about this element in OASIS document type shells.

Inheritance

- topic/q

Example

```
George said, <q>Disengage the power supply before servicing the unit.</q>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.2.2.32 <section>

The `<section>` element represents an organizational division in a topic. Sections are used to organize subsets of information that are directly related to the topic. Multiple `<section>` elements within a single topic do not represent a hierarchy, but rather peer divisions of that topic. Sections cannot be nested. A `<section>` can have an optional title.

For example, the titles **Reference Syntax** and **Properties** might represent section-level discourse within a topic about a command-line process—the content in each `<section>` relates uniquely to the subject of that topic.

> **Note:** For maximum flexibility, 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.

Content models

See *appendix (441)* for information about this element in OASIS document type shells.

Inheritance

- topic/section

Example

```
<reference id="reference">
 <title>Copy Command</title>
 <refbody>
  <section>
   <title>Purpose</title>
   <p>This little command copies
   things.</p>
  </section>
 </refbody>
</reference>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, and *spectitle*.

3.2.2.33 <sectiondiv>

The `<sectiondiv>` element allows logical grouping of content within a section. There is no additional meaning associated with the `<sectiondiv>` element. The `<sectiondiv>` element does not contain a title; the lowest level of titled content within a topic is the section itself. If additional hierarchy is required, use nested topics instead of the section.

The `<sectiondiv>` element nests itself, so it can be specialized to create structured information within sections. 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` attribute.

Because 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.

Content models

See *appendix (442)* for information about this element in OASIS document type shells.

Inheritance

- topic/sectiondiv

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>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.2.2.34 <sl>

The `<sl>` element contains a simple list of items of short, phrase-like content, such as a list of materials in a kit or package.

On output, the list should have no bullets, on the assumption that each item is short enough to fit on one line, and needs no additional differentiation from its neighbors.

Content models

See *appendix (443)* for information about this element in OASIS document type shells.

Inheritance

- topic/sl

Example

In a reference topic discussing related modules, the following sample 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>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, *compact*, and *spectitle*.

3.2.2.35 <sli>

The `<sli>` element is an item in a simple list (<sl>). Simple list items have phrase or text content, adequate for describing package contents, for example. When a DITA topic is formatted for output, the items of a simple list should be placed each on its own line, with no other prefix such as a number (as in an ordered list) or bullet (as in an unordered list).

Content models

See *appendix (443)* for information about this element in OASIS document type shells.

Inheritance

- topic/sli

Example

See *sl* *(217)*.

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)* and *outputclass*.

3.2.2.36 <term>

The `<term>` element identifies words that might have or require extended definitions or explanations.

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.

Content models

See *appendix* *(448)* for information about this element in OASIS document type shells.

Inheritance

- topic/term

Example

```
<p>A <term>reference implementation</term> of DITA implements the standard,
fallback behaviors intended for DITA elements.</p>
```

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)*, *outputclass*, and *@keyref* *(360)*.

3.2.2.37 <text>

The `<text>` element associates no semantics with its content. It exists to serve as a container for text where a container is needed (for example, as a target for content references, or for use within restricted content models in specializations).

Unlike `<ph>`, `<text>` cannot contain images. Unlike `<keyword>`, `<text>` does not imply keyword-like semantics. The `<text>` element contains only text data, or nested `<text>` elements.

Content models

See *appendix* *(448)* for information about this element in OASIS document type shells.

Inheritance

- topic/text

Example

```
<p>This an example of <keyword><text id="reuse">Text
   that is reusable</text></keyword>, with no extra
   semantics attached to the text when it is reused.</p>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.2.2.38 <tm>

The `<tm>` element identifies a term or phrase that is trademarked. Trademarks include registered trademarks, service marks, slogans, and logos.

The business rules for indicating and displaying trademarks might differ from company to company. These business rules can be enforced by either authoring policy or processing.

Content models

See *appendix (450)* for information about this element in OASIS document type shells.

Inheritance

- topic/tm

Example

```
<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>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and the attributes defined below.

@tmtype (REQUIRED)

  Specifies the trademark type. Allowable values are:

tm

   trademark (tm)

reg

   registered trademark (regtm)

service

   service mark (service)

-dita-use-conref-target

   See *Using the -dita-use-conref-target value (362)* for more information.

@trademark

  The trademarked term.

@tmowner

  The trademark owner, for example "OASIS".

@tmclass

  Classification of the trademark. This can be used to differentiate different groupings of trademarks.

3.2.2.39

The `<ul>` element is a list of items in which the order of list items is not significant. List items are typically styled on output with a "bullet" character, depending on nesting level.

Content models

See *appendix (453)* for information about this element in OASIS document type shells.

Inheritance

- topic/ul

Example

```
<ul>
 <li>This is an item in an unordered list.</li>
 <li>To separate it from other items in the list, the
 formatter puts a bullet beside it.</li>
 <li>The following paragraph, contained in the list item
 element, is part of the list item which contains it.
 <p>This is the contained paragraph.</p></li>
 <li>This is the last list item in our unordered list.</li>
</ul>
```

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)*, *outputclass*, *compact*, and *spectitle*.

3.2.2.40 <xref>

Use the `<xref>` element to provide an inline cross reference. It is commonly used to link to a different location within the current topic, a different topic, a specific location in another topic, or an external resource. The target of the cross-reference is specified using the `@href` or `@keyref` attributes.

When creating a cross-reference, link to the element structure, not the `<title>` element of the object. For example, to create a cross-reference to a figure, link to the `<fig>` element, not the `<title>` element within the `<fig>` element. Output processing determines whether the text of the `<title>` element is used to render the cross-reference.

Cross references that link to elements in other topics can employ key-based addressing by using the `@keyref` attribute. This enables the cross-reference to point to different topics in the context of different top-level maps. Cross references that employ direct URI-based addressing using the `@href` attribute create dependencies; if the topic with the cross-reference is included in a map, the target topic also might need to be included for the cross-reference to be resolvable in the context of that map. While you can use conditional processing to have different cross-references for different contexts, it is usually easier and more effective to use keys. By using keys, the cross-reference can be independent of the contexts it might be used in, because it is up to each different map to bind the key used by the cross-reference to the appropriate target.

In many cases it is best to avoid inline cross references in favor of links of links that are generated by the *related-links* *(190)* element or relationship tables. Such links typically are rendered at the end of the topic. This prevents readers from leaving a topic and missing important information. Using relationship tables also enables the links to be managed at a DITA map level, making it possible for topics to be quickly integrated into new contexts without breaking links.

Content models

See *appendix* *(455)* for information about this element in OASIS document type shells.

Inheritance

- topic/xref

Examples

- This example shows a cross-reference to another topic; that topic's title will typically be used as the link text.

```
<p>Background information about DITA is provided in the topic entitled
<xref href="whatsdita.dita"></xref>.</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 used in different map contexts.

```
<p>Background information about DITA is provided in the topic entitled
<xref keyref="whatisdita"></xref>.</p>
```

- This example shows cross-references to another topic; the supplied text will be used as the link text:

```
<p><xref href="whatsdita.dita#tmmdita">Background information about
DITA</xref> is provided free of charge. For a context free cross reference,
the keyref attribute can also link to a topic for <xref keyref="whatisdita">information
 about DITA</xref>.</p>
```

- If you are linking to an element inside of a topic, use the following format in the `@href` attribute:

```
filename.dita#topicid/elementid
```

- The `@keyref` attribute can also be used to link to an element inside of a topic; in this case the key represents the topic, but the element ID remains the same:

```
keyname/elementid
```

- If you are linking within the same file, you can leave off the file name portion `filename.dita`. So, for a section with the ID "mysection", use:

```
#topicid/mysection
```

- Alternatively, if the link is to an element within the same topic you can use:

```
#./mysection
```

- For a list item within that section, assuming the item has an ID of "mylist", use:

```
#topicid/mylist
```

- Alternatively, because the link is to an element within the same topic, you can use:

```
#./mylist
```

See *DITA addressing* (62) for details on using URI references and *Indirect key-based addressing* (66) for details on using key references.

If your URL has an ampersand (&) in it, you need to code that using a entity reference. For example, this URL includes an & character:

```
http://www.example.com/docview.wss?rs=757&context=SSVNX5
```

When used in an `@href` attribute, the ampersand must be entered as **&** as shown here:

```
<xref href="http://www.example.com/docview.wss?rs=757&amp;context=SSVNX5"
scope="external">Part number SSVNX5</xref>
```

Attributes

The following attributes are available on this element: *Universal attribute group* (347), *Link relationship attribute group* (357), *outputclass*, and *@keyref* (360).

Related Links

2.3 DITA addressing (62)

> 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.

The `<link>` element defines a relationship to another topic or non-DITA resource. When displayed, links are typically sorted based on their attributes, which define the type or role of the link's target in relation to the current topic.

3.2.3 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 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.

3.2.3.1 <table>

The `<table>` element organizes arbitrarily complex relationships of tabular information. This standard table markup allows column or row spanning and table captions or descriptions. An optional title allowed inside the `<table>` element provides a caption to describe the table.

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. In addition, the table includes a `<desc>` element, which enables table description that is parallel with figure description. See *simpletable (230)* for a simplified table model that 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.

Content models

See *appendix (447)* for information about this element in OASIS document type shells.

Inheritance

- topic/table

Example

```
<table>
<tgroup cols="2">
<colspec colname="COLSPEC0" colwidth="121*"/>
<colspec colname="COLSPEC1" 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>
```

Figure 55: DITA source

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

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.

```
<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>
```

Figure 56: DITA source

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 7: 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 8: 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.

```
<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>
```

Figure 57: DITA source

The formatted output might be displayed as follows:

Table 9: 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

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, `@display` and `@scale` from *Display attribute group (355)*, and the attributes defined below. This element also uses `@colsep`, `@rowsep`, and `@rowheader` from *Complex-table attribute group (353)*.

@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 *Using the -dita-use-conref-target value (362)* 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

  Determines the horizontal placement of the element. Supported values are 1 and 0, although these are not mandated by the DTD, Schema, or RNG.
  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.

3.2.3.2 <tgroup>

The `<tgroup>` element in a table contains the header and body rows of a table.

Content models

See *appendix (449)* for information about this element in OASIS document type shells.

Inheritance

- topic/tgroup

Example

See *table (222)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, and the attribute defined below. This element also uses `@colsep`, `@rowsep`, and `@align` from *Complex-table attribute group (353)*.

@cols (REQUIRED)

    Indicates the number of columns in a `<tgroup>`.

3.2.3.3 <colspec>

The `<colspec>` element contains a column specification for a table, including assigning a column name and number, cell content alignment, and column width.

Content models

See *appendix (413)* for information about this element in OASIS document type shells.

Inheritance

- topic/colspec

Example

See *table (222)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (without the Metadata attribute group), `@base` from the *Metadata attribute group (348)*, *outputclass*, and the attributes defined below. This element also uses `@align`, `@char`, `@charoff`, `@colsep`, `@rowsep`, and `@rowheader` from *Complex-table attribute group (353)*.

@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.

3.2.3.4 <thead>

The `<thead>` element is a table header that precedes the table body (`<tbody>`) element in a complex table.

Content models

See *appendix (449)* for information about this element in OASIS document type shells.

Inheritance

- topic/thead

Example

See *table (222)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, and `@valign` from *Complex-table attribute group (353)*.

3.2.3.5 <tbody>

The `<tbody>` element contains the rows in a table.

Content models

See *appendix (447)* for information about this element in OASIS document type shells.

Inheritance

- topic/tbody

Example

See *table (222)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, and `@valign` from *Complex-table attribute group (353)*.

3.2.3.6 <row>

The `<row>` element contains a single row in a table.

Content models

See *appendix (441)* for information about this element in OASIS document type shells.

Inheritance

- topic/row

Example

See *table (222)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*. This element also uses `@rowsep` and `@valign` from *Complex-table attribute group (353)*.

3.2.3.7 <entry>

The `<entry>` element defines a single cell in a table.

Content models

See *appendix (421)* for information about this element in OASIS document type shells.

Inheritance

- topic/entry

Example

See *table (222)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (without the Metadata attribute group), `@base` and `@rev` from the *Metadata attribute group (348)*, *outputclass*, and the attributes defined below. This element also uses `@align`, `@char`, `@charoff`, `@colsep`, `@rowsep`, and `@valign` from the *Complex-table attribute group (353)*.

@rotate

  Indicates whether the contents of the entry is rotated. While the attribute is declared with the XML data type CDATA, the only predefined values are:

1

  The contents of the cell are rotated 90 degrees counterclockwise.

0

  No rotation occurs.

-dita-use-conref-target

  See *Using the -dita-use-conref-target value (362)* 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

  The presence of the `@scope` attribute indicates that the current entry is a header for the specified scope. 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.

@headers

   Identifies one or more entry element headers that apply to its 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.

3.2.3.8 <simpletable>

The `<simpletable>` element is used for tables that are regular in structure and do not need a caption. Choose the `<simpletable>` element when you want to show information in regular rows and columns. 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 your structure.

This 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).

Content models

See *appendix (443)* for information about this element in OASIS document type shells.

Inheritance

- topic/simpletable

Example

Source:

```
<simpletable>
 <sthead>
  <stentry>Type style</stentry>
  <stentry>Elements used</stentry>
 </sthead>
 <strow>
  <stentry>Bold</stentry>
  <stentry>b</stentry>
 </strow>
 <strow>
  <stentry>Italic</stentry>
  <stentry>i</stentry>
 </strow>
 <strow>
  <stentry>Underlined</stentry>
  <stentry>u</stentry>
 </strow>
</simpletable>
```

Formatted output:

Type style	Elements used
Bold	b
Italic	i

Type style	Elements used
Underlined	u

Example using @keycol

In this sample, the first column is identified as a header column through the use of `keycol="1"` on the `<simpletable>` element. This indicates that items in the first column should be treated as headers for the row that follows. Rendering of the header column is left up to the implementation.

Source:

```
<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>
```

Formatted output:

Term	Categorization	Definition
Widget	noun	Thing that is used for something
Frustration	noun	What you feel when you drop the widget

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Display attribute group (355)*, *Simpletable attribute group (358)*, *outputclass*, and *spectitle*.

3.2.3.9 <sthead>

The `<sthead>` element contains an optional header row for a `<simpletable>` element.

Content models

See *appendix (445)* for information about this element in OASIS document type shells.

Inheritance

- topic/sthead

Example

See *simpletable (230)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.2.3.10 <strow>

The `<strow>` element contains a single row inside of a `<simpletable>` element.

Content models

See *appendix (445)* for information about this element in OASIS document type shells.

Inheritance

- topic/strow

Example

See *simpletable (230)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.2.3.11 <stentry>

The `<stentry>` element represents a single cell within a `<simpletable>` element. You can place any number of `<stentry>` cells in either an `<sthead>` element (for headings) or `<strow>` element (for rows of data).

Content models

See *appendix (445)* for information about this element in OASIS document type shells.

Inheritance

- topic/stentry

Example

See *simpletable (230)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, and *specentry*.

3.2.4 Related links elements

The related-links section of DITA topics is a special structure that contains links. Links support navigation from a topic to other related topics or resources.

Links are different from cross-references. While cross-references occur only within the body of a topic and can target any element in this or other topics, links only represent topic-to-topic connections, or connections to non-DITA resources. Links occur after the body of a topic, in the *related-links (190)* element.

Links can also be managed indirectly using DITA maps, which provide a more efficient way to manage links. Using maps to define links avoids embedded pointers in each topic. This helps keep topics free from specific contexts, and makes it easier to reuse those topics in new locations.

3.2.4.1 <link>

The `<link>` element defines a relationship to another topic or non-DITA resource. When displayed, links are typically sorted based on their attributes, which define the type or role of the link's target in relation to the current topic.

The optional container elements for link (`<linkpool>` and `<linklist>`) allow authors to define groups with common attributes or to preserve the authored sequence of links on output. Links placed in a `<linkpool>` might be rearranged or removed for display purposes (combined with other local or map-based links); links in a

`<linklist>` should be displayed in the order they are defined. Refer to those elements for additional explanation.

Content models

See *appendix (430)* for information about this element in OASIS document type shells.

Inheritance

- topic/link

Example

```
<related-links>
  <linkpool type="concept">
    <link href="czez.dita#czez" role="next"></link>
    <link href="czunder.dita"></link>
    <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>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)*, *outputclass*, `@keyref` *(360)*, *The role and otherrole attributes (373)*, and *query (DEPRECATED)*.

Related Links

2.3 DITA addressing (62)

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.

3.2.2.40 xref (220)

Use the `<xref>` element to provide an inline cross reference. It is commonly used to link to a different location within the current topic, a different topic, a specific location in another topic, or an external resource. The target of the cross-reference is specified using the `@href` or `@keyref` attributes.

3.2.4.2 <linklist>

The `<linklist>` element defines an author-arranged group of links. When rendering the links, processors *SHOULD* preserve the order of links specified within a `<linklist>` element.

There are two ways to organize related information links within a topic. First, you can add them all in no particular order, either by using `<linkpool>` elements or by placing `<link>` elements directly within `<related-links>`, in which case the rendering is implementation dependent. For example, a tool could sort all links based on the role or type; a tool could also move or remove links to fit the context (for example, moving a prerequisite link to the top of a browser window, or removing links to the next topic if it is rendered on the same page in a PDF). These behaviors are examples only and are not required.

Second, links can be grouped using one or more `<linklist>` elements. When you group them using `<linklist>`, then the order of the links within each `<linklist>` is preserved when rendered. You can also use a combination of the two approaches, which will allow some links to be automatically sorted while the others are left as-is.

Attributes set on the `<linkpool>` and `<linklist>` elements are inherited by their descendants. For example, if you have a `<linklist>` element that contains all external links, you can set `scope="external"` on that outer `<linklist>` element and leave it off the `<link>` elements within that `<linklist>`.

Content models

See *appendix (430)* for information about this element in OASIS document type shells.

Inheritance

- topic/linklist

Example

```
<related-links>
  <linklist scope="external">
    <title>Example links</title>
    <desc>These links will always appear in this order.</desc>
    <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>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, *collection-type*, *The role and otherrole attributes (373)*, *spectitle*, *mapkeyref*, and the attributes defined below. This element also uses `@format`, `@scope`, and `@type` from *Link relationship attribute group (357)*.

@duplicates

Specifies whether or not duplicate links will be filtered out of a group of links. Allowable values are:

yes

Allow duplicate links

no

Filter out duplicate links

-dita-use-conref-target

See *Using the -dita-use-conref-target value (362)* for more information.

Conceptually, two links are duplicates if they address the same resource using the same properties, such as link text and link role. The details of determining duplicate links is processor specific.

The suggested processing default is "yes" within `<linklist>` elements and "no" for other links.

@collection-type

See *collection-type* for a full definition and list of supported values.

In the initial DTD implementation of DITA, this attribute was defined with an additional value of "tree"; that value was only defined for `@collection-type` on the `<linkpool>` and `<linklist>` elements. The "tree" value is not allowed on `@collection-type` when used in maps, and is not defined in the XSD or RELAX NG versions of `<linkpool>` or `<linklist>`. The extra value in the DTD implementation is retained for backwards compatibility, but is deprecated.

3.2.4.3 <linkpool>

The `<linkpool>` element defines a group of links that have common characteristics, such as type or audience or source. When links are in `<related-links>` or `<linkpool>` elements, the organization of links on final output is determined by the output process, not by the order that the links actually occur in the DITA topic.

There are two ways to organize related information links within a topic. First, you can add them all in no particular order, either by using `<linkpool>` elements or by placing `<link>` elements directly within `<related-links>`, in which case the rendering is implementation dependent. For example, a tool could sort all links based on the role or type; a tool could also move or remove links to fit the context (for example, moving a prerequisite link to the top of a browser window, or removing links to the next topic if it is rendered on the same page in a PDF). These behaviors are examples only and are not required.

Second, links can be grouped using one or more `<linklist>` elements. When you group them using `<linklist>`, then the order of the links within each `<linklist>` is preserved when rendered. You can also use a combination of the two approaches, which will allow some links to be automatically sorted while the others are left as-is.

Attributes set on the `<linkpool>` and `<linklist>` elements are inherited by their descendants. For example, if you have a `<linklist>` element that contains all external links, you can set `scope="external"` on that outer `<linklist>` element and leave it off the `<link>` elements within that `<linklist>`.

Content models

See *appendix (431)* for information about this element in OASIS document type shells.

Inheritance

- topic/linkpool

Example

```
<related-links>
  <linkpool type="concept">
    <link href="czez.dita#czez" role="next"></link>
    <link href="czunder.dita"></link>
    <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>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, *collection-type*, *The role and otherrole attributes (373)*, *mapkeyref*, and the attributes defined below. This element also uses `@format`, `@scope`, and `@type` from *Link relationship attribute group (357)*.

@duplicates

  Specifies whether or not duplicate links will be filtered out of a group of links. Allowable values are:

yes

    Allow duplicate links

no

    Filter out duplicate links

-dita-use-conref-target

    See *Using the -dita-use-conref-target value (362)* for more information.

Conceptually, two links are duplicates if they address the same resource using the same properties, such as link text and link role. The details of determining duplicate links is processor specific.

The suggested processing default is "yes" within `<linklist>` elements and "no" for other links.

@collection-type

See *collection-type* for a full definition and list of supported values.

In the initial DTD implementation of DITA, this attribute was defined with an additional value of "tree"; that value was only defined for `@collection-type` on the `<linkpool>` and `<linklist>` elements. The "tree" value is not allowed on `@collection-type` when used in maps, and is not defined in the XSD or RELAX NG versions of `<linkpool>` or `<linklist>`. The extra value in the DTD implementation is retained for backwards compatibility, but is deprecated.

3.2.4.4 <linktext>

The `<linktext>` element provides the literal label or line of text for a link. For links to local DITA topics, the text of a link can typically be resolved during processing. Use the `<linktext>` element only when the target cannot be reached, such as when it is a peer or external link, or when the target is local but not in DITA format. When used inside a topic, `<linktext>` is used as the text for the specified link; when used within a map, `<linktext>` is used as the text for generated links that point to the specified topic.

Content models

See *appendix (431)* for information about this element in OASIS document type shells.

Inheritance

"- topic/linktext " when used in topics, and "- map/linktext " when used in maps.

Example

```
<link href="tzover.htm#accsqlj">
 <linktext>Accessing relational data with SQLJ</linktext>
</link>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.2.4.5 <linkinfo>

The `<linkinfo>` element allows you to place a descriptive paragraph after the links that are contained in a `<linklist>` element.

Content models

See *appendix (430)* for information about this element in OASIS document type shells.

Inheritance

- topic/linkinfo

Example

```
<linklist>
  <title>Repairing widgets</title>
  <link href="debug.dita" type="task"></link>
  <link href="repair.dita" type="task"></link>
  <link href="test.dita" type="task"></link>
  <linkinfo>To repair a reciprocating widget,
you must follow the instructions very carefully. Note
```

```
the sequence to follow. Do it.</linkinfo>
</linklist>
```

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)*.

3.3 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.

3.3.1 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.

3.3.1.1 <map>

The `<map>` element describes the relationships among a set of resources, such as DITA topics. Maps consist of references to topics, maps, and other resources organized into hierarchies, groups, and tables. Maps express these relationships in a single common format that can be used for different outputs.

The containing element for a map is the `<map>` element. Within the map, use the `<topicref>` element to add and organize references to the topics, and the `<topicgroup>` and `<reltable>` elements to provide non-hierarchical relationships. You can use the `<map>` element to set default attribute values for all `<topicref>` elements in the map.

A map describes the relationships among a set of DITA topics. The following are some examples of relationships that can be described in a map:

- Hierarchical (Parent/Child). 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.

When rendering a map, processors might make use of these relationships, such as to create a Table of Contents (TOC), aggregate topics into a PDF document, or create links between topics in output.

The `<title>` element can optionally be used to provide a title for the map (the `<title>` element is preferred over the `@title` attribute). In some scenarios the title is purely informational, and is present as an aid to the author. In other scenarios it might be useful or even required. For example, if a map is converted to Eclipse Help, the Eclipse system will require a title for the resulting table of contents. In the `<bookmap>` specialization of `<map>`, the `<title>` element provides a title for the book represented by that map. In a map referenced by another map, the `<title>` might be discarded as topics from the submap are aggregated into a larger publication.

Content models

See *appendix* *(432)* for information about this element in OASIS document type shells.

Inheritance

- map/map

Example

In this example, there are six `<topicref>` elements. They 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" type="topic">
    <topicref href="batcaring.dita" type="task"></topicref>
    <topicref href="batfeeding.dita" type="task"></topicref>
    <topicref href="batsonar.dita" type="concept"></topicref>
    <topicref href="batguano.dita" type="reference"></topicref>
    <topicref href="bathistory.dita" type="reference"></topicref>
  </topicref>
</map>
```

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)* (with a narrowed definition of `@id`, given below), *Attributes common to many map elements* *(351)*, *Architectural attribute group* *(350)*, *outputclass*, and the attributes defined below. This element also uses `@type`, `@scope`, and `@format` from *Link relationship attribute group* *(357)*.

@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.

@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.

@title

An identifying title for the map. May be used or ignored, depending on the capabilities of the display mechanism. Note that beginning with DITA 1.1, the map can include a `<title>` element, which is preferred over the `@title` attribute.

3.3.1.2 <topicref>

The `<topicref>` element identifies a topic (such as a concept, task, or reference) or other resource. A `<topicref>` can contain other `<topicref>` elements, allowing you to express navigation or table-of-contents hierarchies, as well as implying relationships between a containing (parent) `<topicref>` and its children. You can set the collection type of a parent `<topicref>` to determine how its children are related to each other. You can also express relationships among `<topicref>` elements by using group and table structures (such as `<topicgroup>` and `<reltable>`). Relationships are expressed as links in the output; by default, each participant in a relationship has links to the other participants in that relationship.

You can fine tune the output from your map by setting different attributes on the `<topicref>` element. For example, the `@linking` attribute controls how a topic's relationships to other topics are expressed as links, and the `@toc` attribute controls whether the topic shows up in TOC or navigation output.

Content models

See *appendix* *(451)* for information about this element in OASIS document type shells.

Inheritance

- map/topicref

Example

In this example, there are six `<topicref>` elements. They are nested and have a hierarchical relationship. `bats.dita` is the parent topic and the other topics are its children.

```
<map title="Bats">
 <topicref href="bats.dita" type="topic">
  <topicref href="batcaring.dita" type="task"></topicref>
  <topicref href="batfeeding.dita" type="task"></topicref>
  <topicref href="batsonar.dita" type="concept"></topicref>
  <topicref href="batguano.dita" type="reference"></topicref>
  <topicref href="bathistory.dita" type="reference"></topicref>
 </topicref>
</map>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with a narrowed definition of `@href`, given below), *Attributes common to many map elements (351)*, *Topicref element attributes group (358)*, *outputclass*, *@keys (360)*, and *@keyref (360)*.

@href

A pointer to the resource represented by the `<topicref>`. See *The href attribute (359)* 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.

3.3.1.3 <topicmeta>

The `<topicmeta>` element defines the metadata that applies to a topic when it appears in a map. When appropriate, that metadata also applies to the other topics in the map that are contained by the same element that contains the `<topicmeta>` element. When creating links, `<topicmeta>` content can also be used to override the title and short description that are used for the link. In addition, it can be used to add index entries to referenced content using the `<keywords>` element.

The metadata given 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, the two references to a single resource might specify different navigation titles or search titles, each of which is specific to a single context.

Note: The topic *Cascading of metadata attributes in a DITA map (53)* in the DITA Architectural Specification provides more information about which metadata elements inside `<topicmeta>` cascade to other `<topicref>` elements. In addition, the topic *Reconciling topic and map metadata elements (55)* provides more information about how metadata specified in the map interacts with metadata specified in each topic.

Content models

See *appendix (451)* for information about this element in OASIS document type shells.

Inheritance

- map/topicmeta

Example

In this example, the metadata defined by the `<topicmeta>` element applies to the associated `<topicref>` (`bats.dita`) and all of its children. The `<topicmeta>` element contains an audience definition that indicates that `bats.dita` and its children are of interest to experienced programmers who are troubleshooting.

```
<map>
  <topicref href="bats.dita">
```

```
    <topicmeta>
      <audience type="programmer" job="troubleshooting" experiencelevel="expert"/>
    </topicmeta>
    <topicref href="batcaring.dita"></topicref>
    <topicref href="batfeeding.dita"></topicref>
  </topicref>
</map>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and the attribute defined below.

@lockmeta

By default, metadata in the map supplements or overrides metadata that is specified at the topic level, unless the `@lockmeta` attribute of the `<topicmeta>` element is set to "no". Allowable values are "yes", "no", and *Using the -dita-use-conref-target value (362)*.

3.3.1.4 <anchor>

The `<anchor>` element provides an integration point that another map can reference in order to insert its navigation into the referenced map's navigation tree. For those familiar with Eclipse help systems, this serves the same purpose as the `<anchor>` element in that system. It might not be supported for all output formats.

The mechanism by which map processors discover maps to be anchored is processor specific.

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.

Content models

See *appendix (409)* for information about this element in OASIS document type shells.

Inheritance

- map/anchor

Example

In this example, the `map1.ditamap` contains an `<anchor>` element with an `@id` attribute set to "a1".

```
<map>
  <title>MyComponent tasks</title>
  <topicref navtitle="Start here" href="start.dita" toc="yes">
    <navref mapref="othermap2.ditamap"/>
    <navref mapref="othermap3.ditamap"/>
    <anchor id="a1"/>
  </topicref>
</map>
```

Figure 58: DITA map that contains an anchor

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.

```
<map anchorref="map1.ditamap#a1">
  <title>This map is pulled into the MyComponent task map</title>
  <!-- ... -->
</map>
```

Figure 59: DITA map that references an anchor

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (with a narrowed definition of `@id`, given below) and *outputclass*.

@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 *ID attribute (62)* for more details.

3.3.1.5 <navref>

The `<navref>` element represents a pointer 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.

For example, if a map is converted to the Eclipse help system format, the DITA element `<navref mapref="other.ditamap"/>` is converted to the Eclipse element `<link toc="other.xml"/>`. When Eclipse loads the referencing map, it will replace this link element with the contents of the file `other.xml`, provided that the file `other.xml` is available.

Note that not all output formats support such linking. In order to include another map directly without depending on the output format, 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"/>
```

Content models

See *appendix (433)* for information about this element in OASIS document type shells.

Inheritance

- map/navref

Example

In this example, the map titled "MyComponent tasks" references the maps `othermap2.ditamap` and `othermap3.ditamap`.

```
<map title="MyComponent tasks">
 <navref mapref="../com.ibm.xml.doc/othermap1.ditamap"/>
 <navref mapref="../com.ibm.xml.doc/othermap2.ditamap"/>
</map>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, *@keyref (360)*, 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 your output format (such as XML TOC file for Eclipse output).

@keyref (DEPRECATED)

The `@keyref` attribute was unintentionally defined for `<navref>` in the original DITA grammar files. It is retained for backwards compatibility. The attribute will be removed in a future release, and processors are not expected to support it.

3.3.1.6 <reltable>

The `<reltable>` element is a relationship table that specifies relationships among topics, based on the familiar table model of rows (`<relrow>`), columns (`<relheader>`), and cells (`<relcell>`).

A frequently-used type of relationship table establishes relationships between task, concept, and reference topics. Each column in a relationship table typically represents a specific role in a set of relationships; for example, the first column often contains references to tasks, while the second and third columns often 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.

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).

Relationship tables can be used in conjunction with hierarchies and groups to manage all the related links in an information set.

Within a map tree, the effective relationship table is the union of all relationship tables in the map.

Content models

See *appendix* *(440)* for information about this element in OASIS document type shells.

Inheritance

- map/reltable

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: `batsonar.dita`. The second cell contains two task topics: `batcaring.dita` and `batfeeding.dita`. The third cell contains two reference topics: `batguano.dita` and `bathistory.dita`.

```
<map>
  <reltable>
    <relheader>
      <relcolspec type="concept"/>
      <relcolspec type="task"/>
      <relcolspec type="reference"/>
    </relheader>
    <relrow>
      <relcell>
        <topicref href="batsonar.dita"/>
      </relcell>
      <relcell>
        <topicref href="batcaring.dita"/>
        <topicref href="batfeeding.dita"/>
      </relcell>
      <relcell>
        <topicref href="batguano.dita"/>
        <topicref href="bathistory.dita"/>
      </relcell>
    </relrow>
  </reltable>
</map>
```

A DITA-aware tool might represent the `<reltable>` element graphically:

type="concept"	type="task"	type="reference"
`batsonar.dita`	`batcaring.dita` `batfeeding.dita`	`batguano.dita` `bathistory.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, `batcaring.dita` and `batfeeding.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:

batsonar.dita

`batcaring.dita`, `batfeeding.dita`, `batguano.dita`, `bathistory.dita`

batcaring.dita

`batsonar.dita`, `batguano.dita`, `bathistory.dita`

batfeeding.dita

`batsonar.dita`, `batguano.dita`, `bathistory.dita`

batguano.dita

`batsonar.dita`, `batcaring.dita`, `batfeeding.dita`

bathistory.dita

`batsonar.dita`, `batcaring.dita`, `batfeeding.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 `batfeeding.dita` and `batcaring.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.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Attributes common to many map elements (351)* (without the `@keyscope` attribute, and with a narrowed definition of `@toc`, given below), *outputclass*, and the attributes defined below. This element also uses `@type`, `@scope`, and `@format` from *Link relationship attribute group (357)*.

@toc

Specifies whether a topic appears in the table of contents (TOC). If the value is not specified locally, but is specified on an ancestor, the value will cascade from the closest ancestor. On this element the default value for `@toc` is "no". See *Attributes common to many map elements (351)* for a complete definition of `@toc`.

@title

An identifying title for this element.

3.3.1.7 <relrow>

The `<relrow>` element defines a row in the relationship table (`<reltable>`). It creates a relationship between the cells in the row, which is expressed in output as links between the topics or resources referenced in those cells.

Content models

See *appendix (440)* for information about this element in OASIS document type shells.

Inheritance

- map/relrow

Example

See *reltable (242)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.3.1.8 <relcell>

The `<relcell>` element defines a cell in the relationship table (`<reltable>`). The `<topicref>` elements that it contains are related to the `<topicref>` elements in other cells of the same row. By default, topics or resources that are referenced in the same cell are not related to each other, unless you change the `@collection-type` attribute of the `<relcell>` to indicate that they are related.

Content models

See *appendix (439)* for information about this element in OASIS document type shells.

Inheritance

- map/relcell

Example

See *reltable (242)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Attributes common to many map elements (351)* (without `@keyscope` attribute), and *outputclass*. This element also uses `@type`, `@scope`, and `@format` from *Link relationship attribute group (357)*.

3.3.1.9 <relheader>

The `<relheader>` element is a row in a relationship table that contains column definitions (`<relcolspec>` elements). Each table can have only one set of column definitions.

Content models

See *appendix (440)* for information about this element in OASIS document type shells.

Inheritance

- map/relheader

Example

See *reltable (242)*.

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)*.

3.3.1.10 <relcolspec>

The `<relcolspec>` element is used to define a column in a relationship table. The `<relcolspec>` element can be used to set default attribute values for the `<topicref>` elements in the column.

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. 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.

Beginning with DITA 1.2, you also can add `<topicref>` elements to the `<relcolspec>` element; this defines a relationship between the topics that are referenced in 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>`.

Also beginning with DITA 1.2, if you add a `<title>` element to the `<relcolspec>` element, the content of the `<title>` element is 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 references a non-DITA resource, the value of the `<topicref>` element's navigation title is used for the label.
- If the `<relcolspec>` element contains a `<topicref>` element that references a DITA resource and the `@locktitle` attribute is set to "yes," the value of the `<topicref>` element's navigation title is used for the label.
- If the `<relcolspec>` element contains a `<topicref>` element that references a DITA resource and the `@locktitle` attribute is missing or set to "no," the label is derived from the `<navtitle>` or `<title>` element specified within 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.

Content models

See *appendix* *(439)* for information about this element in OASIS document type shells.

Inheritance

- map/relcolspec

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: `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>
```

Example with column titles

Consider the following relationship table:

```
<reltable>
  <relheader>
    <relcolspec type="task">
      <topicref navtitle="Troubleshooting" href="tbs.dita" locktitle="yes"/>
    </relcolspec>
    <relcolspec type="reference">
      <topicref navtitle="Messages" href="msg.dita" locktitle="yes"/>
    </relcolspec>
  </relheader>
  <relrow>
    <relcell>
      <topicref navtitle="Debugging login errors" href="debug_login.dita"/>
    </relcell>
    <relcell>
      <topicref navtitle="Login not found" href="login_error_1.dita"/>
    </relcell>
  </relrow>
  <relrow>
    <relcell>
      <topicref navtitle="Checking access controls" href="checking_access.dita"/>
    </relcell>
    <relcell>
      <topicref navtitle="Login not allowed" href="login_error_2.dita"/>
    </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 *MAY* 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.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Attributes common to many map elements (351)* (without `@keyscope` attribute), and *outputclass*. This element also uses `@type`, `@scope`, and `@format` from *Link relationship attribute group (357)*.

3.3.1.11 <ux-window>

Use the `<ux-window>` element to provide specifications for a window or viewport in which a user assistance topic or Web page can be displayed. The window or viewport can be referenced by the `<resourceid>` element associated with a topic or `<topicref>` element.

The `<ux-window>` element can be used anywhere within a map. If more than one `<ux-window>` element in a map has the same `@name` attribute, the first window specification in document order with that `@name` attribute is used.

Content models

See *appendix (454)* for information about this element in OASIS document type shells.

Inheritance

- map/ux-window

Example

In this example, a window with a name of "csh" is defined in the map. The window name is later referenced by the `@ux-windowref` attribute on the `<resourceid>` element.

```
<map title="Widget Help">
 <topicmeta>
  <ux-window id="fg23" name="csh" top="10" left="20" height="400" width="500"
     features="status=yes,toolbar=no,menubar=no,location=no" relative="yes"
     full-screen="no" />
 </topicmeta>
 <topicref href="file_ops.dita" type="concept">
   <topicref href="saving.dita" type="task" />
   <topicref href="deleting.dita" type="task" />
   <topicref href="editing.dita" type="task">
     <topicmeta>
       <resourceid id="ab43" appname="ua"
          appid="5432" ux-context-string="idh_fileedit" ux-windowref="csh" />
     </topicmeta>
   </topicref>
 </topicref>
</map>
```

In this example, different window specifications are defined for tablet and desktop presentation:

```
<map title="Puggles Help">
 <topicmeta>
 <ux-window id="p76" name="ux-tablet" top="5%" left="5%" height="90%" width="90%"
    features="status=no,toolbar=no,menubar=no,location=no" relative="no"
    full-screen="no" />
 <ux-window id="p80" name="ux-desktop" top="5%" left="55%" height="60%" width="40%"
    features="status=yes,toolbar=no,menubar=no,location=yes" relative="no"
    full-screen="no" />
 </topicmeta>
 <topicref href="c_puggles_intro.dita" type="concept">
  <!-- ... -->
 </topicref>
</map>
```

Attributes

The following attributes are available on this element: *ID attribute group (347)*, *Metadata attribute group (348)*, *Debug attribute group (350)*, *class (Not for use by authors)*, and the attributes defined below.

@name (REQUIRED)

The value used to refer to this window definition.

@top

The top position of the target help window, whether relative to the calling window or to the entire display. The value of this attribute is a real number 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).

@left

The left position of the target help window, whether relative to the calling window or to the entire display. The value of this attribute is a real number 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).

@height

The height of the window. The value of this attribute is a real number 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).

@width

The width of the window. The value of this attribute is a real number 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).

@on-top

Indicates whether the initial z-order of the target help window is on top of all windows on the desktop. Allowable values are: "yes", "no", and *-dita-use-conref-target (362)*. The default value is "no".

@features

A list of other features (size, position, scrollbars, etc.) of the window. Each feature name and value can not contain any blank space, and each feature name and value is separated by a comma or other delimiter character.

@relative

Indicates whether the window dimensions are relative to the calling window or the entire target display. The default value is "no". Allowable values are:

no

The window dimensions specified on this element are absolute positions (not relative to the calling window).

yes

The window dimensions specified on this element are relative to the calling window.

-dita-use-conref-target

See *Using the -dita-use-conref-target value (362)* for more information.

@full-screen

Indicates whether the window is initially displayed in a maximized state. Allowable values are: "yes", "no", and *-dita-use-conref-target (362)*. The default value is "no".

Related Links

2.2.4.7 Context hooks and window metadata for user assistance (61)

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.

3.3.2 Map group elements

The map group domain elements define, group, or reference content. Many of the map group elements are convenience elements, which means that they simply make it easier for an author to make use of existing functions.

For example, the `<topichead>` element allows a map to specify a heading without allowing 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 simply makes the intent clearer and prevents the accidental inclusion of an `@href` attribute.

3.3.2.1 <anchorref>

The `<anchorref>` element is used to reference an `<anchor>` element in a map. The contents of an `<anchorref>` element are rendered both in the original authored location and at the location of the referenced `<anchor>` element. The referenced `<anchor>` element might be defined in the current map or another map. When possible, this integration is done when displaying the map with `<anchor>` to an end user.

This function 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 allows the author to attach only the contents of a single map branch. This enables architects to reuse a branch of content without reusing the entire map.

If the rendering platform does not support runtime integration of navigation based on the anchor point, a build system should treat the `<anchorref>` element similar to a "conref push" instruction by pushing the content to the spot that contains 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>` 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 target `<anchor>` element.

By default, the content of the `<anchorref>` element is rendered at both the anchor target and the original location. To prevent the content 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.

Content models

See *appendix* *(410)* for information about this element in OASIS document type shells.

Inheritance

+ map/topicref mapgroup-d/anchorref

Example

```
<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 60: Initial map contents

```
<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>
```

Figure 61: Effective result of evaluating the <anchorref> element

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with narrowed definitions of `@href`, `@type`, and `@format`, all given below), *Attributes common to many map elements (351)*, *Topicref element attributes group (358)*, *outputclass*, `@keys` *(360)*, and `@keyref` *(360)*.

@href

A pointer to an `<anchor>` element in this or another DITA map. When rendered, the contents of the current element will be copied to the location of the `<anchor>`. See *The href attribute (359)* for supported syntax when referencing a map element.

@type

Describes the target of a reference. For the `<anchorref>` element, this value defaults to "anchor", because the element is expected to point to an `<anchor>` element in this or another map.

@format

The `@format` attribute identifies the format of the resource being referenced. For the `<anchorref>` element, this value defaults to "ditamap", because the element references a point in a map.

3.3.2.2 <keydef>

The `<keydef>` element is a convenience element that is used to define keys without any of the other effects that occur when using a `<topicref>` element: no content is included in output, no title is included in the table of contents, and no linking or other relationships are defined. The `<keydef>` element is not the only way to define keys; its purpose is to simplify the process by defaulting several attributes to achieve the described behaviors.

Content models

See *appendix (428)* for information about this element in OASIS document type shells.

Inheritance

+ map/topicref mapgroup-d/keydef

Example

The following example defines keys that can be used to refer to the indicated topics. These keys can be used within the map, from any topic in this map, or in any context where this map is imported. Note that the `@processing-role` attribute defaults to "resource-only", which ensures that specified topics will not be

rendered in a print document or in a navigation TOC based on this definition in the map. In addition, it means that links will not be generated to or from the `<keydef>` elements.

```
<map>
  <title>Defining bird keys</title>
  <keydef keys="darwinfinch galapagosfinch"  href="galapagosfinch.dita"/>
  <keydef keys="goldfinch"  href="about-goldfinches.dita"/>
  <keydef keys="puffin"     href="about-puffins.dita"/>
  <keydef keys="loon diver" href="common-loon.dita"/>
</map>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with a narrowed definition of `@href`, given below), *Attributes common to many map elements (351)* (with a narrowed definition of `@processing-role`, given below), *Topicref element attributes group (358)*, *outputclass*, *@keyref (360)*, and the attributes defined below.

@keys (REQUIRED)

On this element the `@keys` attribute is required, because the purpose of the element is to define a key. Otherwise, the attribute is the same as described in *The keys attribute (360)*.

@href

A pointer to the resource represented by the `<topicref>`. See *The href attribute (359)* 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.

@processing-role

For this element, the default value for `@processing-role` is "resource-only". Otherwise, the definition matches the one found in *Attributes common to many map elements (351)*.

3.3.2.3 <mapref>

The `<mapref>` element is a convenience element that is equivalent to a `<topicref>` element with the `@format` attribute set to "ditamap". 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.

Content models

See *appendix (432)* for information about this element in OASIS document type shells.

Inheritance

+ map/topicref mapgroup-d/mapref

Example

```
<map id="lib">
  <topicref href="netlib.dita"/>
  <topicref href="dblib.dita"/>
  <!-- ... -->
</map>
```

Figure 62: Sample document lib.ditamap that is reusable in other locations

```
<map id="standardlib">
  <topichead navtitle="Developing with standard libraries">
    <mapref href="lib.ditamap"/>
  </topichead>
```

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

Figure 63: Map that reuses lib.ditamap

```
<map id="standardlib">
  <topichead navtitle="Developing with standard libraries">
    <topicref href="netlib.dita"/>
    <topicref href="dblib.dita"/>
    <!-- ... -->
  </topichead>
  <!-- ... -->
</map>
```

Figure 64: Rendered result

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with narrowed definitions of `@href` and `@format`, given below), *Attributes common to many map elements (351)*, *Topicref element attributes group (358)*, *outputclass*, *@keyref (360)*, and *@keys (360)*.

@format

On this element the `@format` attribute sets a default value of "ditamap", because the purpose of the element is to reference a ditamap document. Otherwise, the attribute is the same as described in *Link relationship attribute group (357)*.

@href

A pointer to the resource represented by the `<topicref>`. See *The href attribute (359)* 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.

3.3.2.4 <topicgroup>

The `<topicgroup>` element groups `<topicref>` elements for common treatment without affecting the structural hierarchy of the map, as opposed to nesting `<topicref>` elements, which does imply a structural hierarchy. The `<topicgroup>` element can provide linking relationships and shared, inherited attributes to the set of elements that it contains without affecting the resulting table of contents or navigation.

Beginning with DITA 1.2, you are able to specify a `<navtitle>` element within the `<topicmeta>` element inside of a `<topicgroup>`. The `<topicgroup>` element is meant as a non-titled grouping element, so adding a `<navtitle>` element to the `<topicgroup>` element has no defined purpose, and processors *MUST* ignore the title. Processors *MAY* issue a message when ignoring the title.

Content models

See *appendix (450)* for information about this element in OASIS document type shells.

Inheritance

+ map/topicref mapgroup-d/topicgroup

Example

Each `<topicref>` element in the following example inherits the `@audience` and `@linking` attributes. In this way the common attributes are set for the entire group of `<topicref>` elements without affecting the navigation hierarchy.

```
<topicgroup audience="novice" linking="none">
  <topicref href="this.dita"/>
  <topicref href="that.dita"/>
```

```
    <topicref href="theother.dita"/>
  </topicgroup>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Attributes common to many map elements (351)*, and *outputclass*. Although @locktitle is available as part of *Attributes common to many map elements (351)*, it has no defined purpose for this element.

The @scope, @format, and @type attributes from *Link relationship attribute group (357)* are also available.

3.3.2.5 \<topichead>

The \<topichead> element provides a title-only entry in a navigation map, which should appear as a heading when the map is rendered as a table of contents. In print contexts it should also appear as a heading in the rendered content.

Beginning with DITA 1.2, the navtitle can be specified by using a \<navtitle> element within the \<topicmeta> element, so the \<topichead> element no longer requires the @navtitle attribute. In order to ensure backward compatibility with earlier versions of DITA, the new \<navtitle> element is not required. However, a \<topichead> element must contain either a @navtitle attribute or a \<topicmeta> element that contains a \<navtitle> element. DITA processors *SHOULD* generate a warning if a navigation title is not specified.

Content models

See *appendix (451)* for information about this element in OASIS document type shells.

Inheritance

+ map/topicref mapgroup-d/topichead

Example

Note that in the following example, the first \<topichead> element uses a \<navtitle> element to provide the title, while the second \<topichead> element uses a @navtitle attribute. This is only to illustrate that both uses are valid; in general, the element is preferred over the attribute.

```
<map>
  <topichead>
    <topicmeta><navtitle>Computers</navtitle></topicmeta>
    <topicref href="eniac.dita"/>
    <topicref href="system360.dita"/>
    <topicref href="pdp8.dita"/>
  </topichead>
  <topichead navtitle="Books">
    <topicref href="hardback.dita"/>
    <topicref href="paperback.dita"/>
  </topichead>
</map>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Attributes common to many map elements (351)*, @navtitle and @copy-to from *Topicref element attributes group (358)*, and *outputclass*. Although @locktitle is available as part of *Attributes common to many map elements (351)*, it has no defined purpose for this element.

The @scope, @format, and @type attributes from *Link relationship attribute group (357)* are also available.

3.3.2.6 \<topicset>

The \<topicset> element defines a complete unit of content that can be reused in other DITA maps or other \<topicset> elements. Unlike the base \<topicref> element, the \<topicset> is explicitly intended to define a

set of topics that are often or always used as a unit; it can be especially useful for task composition in which larger tasks are composed of smaller tasks. The `@id` attribute on a `<topicset>` is required, which ensures that the complete unit is available for reuse in other contexts.

A `<topicset>` is similar to a source file that contains nested topics, in that the combination of topics constitutes a complete self-contained unit. That unit of content can stand independently of the containing, prior, and following content within the original map context.

Content models

See *appendix (451)* for information about this element in OASIS document type shells.

Inheritance

+ map/topicref mapgroup-d/topicset

Example

This `<topicset>` represents a set of overview information about SQL. The information is reusable as a unit.

```
<topicset id="sqlbasics" href="sqlOverview.dita">
  <topicref href="sqlSelection.dita"/>
  <topicref href="sqlJoin.dita"/>
  <topicref href="sqlFilter.dita"/>
  <!-- ... -->
</topicset>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (with a narrowed definition of `@id`, given below), *Link relationship attribute group (357)* (with a narrowed definition of `@href`, given below), *Attributes common to many map elements (351)*, *Topicref element attributes group (358)*, *outputclass*, *@keys (360)*, and *@keyref (360)*.

@id (REQUIRED)

This ID is the target for references by to the current set of information. The ID is required in order to ensure that a `<topicset>` is defined as a reusable unit of information. See *ID attribute (62)* for more details.

@href

A pointer to the resource represented by the `<topicref>`. See *The href attribute (359)* 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.

3.3.2.7 <topicsetref>

The `<topicsetref>` element references a `<topicset>` element. The referenced `<topicset>` element can be defined in the current map or in another map.

When possible, applications should treat the referenced `<topicset>` as an independent unit. For example, an application that renders DITA for a dynamic navigation platform might generate a reusable navigation structure for each `<topicset>`, and each `<topicsetref>` is retained as a reference to that structure. This differs slightly from the processing of the `@conref` attribute, which results in a literal copy of the referenced content.

For situations that do not support reusing a topic set as an independent unit, such as a rendered PDF, applications *MAY* resolve the `<topicsetref>` element as for other `<topicset>` (or `<topicref>`) elements that have the `@format` attribute set to "ditamap".

As with other cases where the attribute `format="ditamap"` is specified or used as a default, the use of topic references nested inside of `<topicsetref>` is undefined.

Content models

See *appendix (452)* for information about this element in OASIS document type shells.

Inheritance

+ map/topicref mapgroup-d/topicsetref

Example

The `sql-tutorial.ditamap` file contains a `<topicset>` element that groups together several topics that collectively comprise an overview of SQL.

```
<!-- Excerpt from sql-tutorial.ditamap -->
<topicset id="sqlbasics" href="sqlOverview.dita">
  <topicref href="sqlSelection.dita"/>
  <topicref href="sqlJoin.dita"/>
  <topicref href="sqlFilter.dita"/>
  <!-- ... -->
</topicset>
```

Figure 65: DITA map that contains a <topicset> element

Another map can include that topic set, in addition to content related to programming with JDBC.

```
<topichead navtitle="Mastering JDBC">
  <topicsetref href="sql-tutorial.ditamap#sqlbasics"/>
  <topicref href="jdbcPrepare.dita"/>
  <!-- ... -->
</topichead>
```

Figure 66: DITA map that contains a <topicsetref> element

A reader of the JDBC information will see the content integrated as a single unit.

```
<topichead navtitle="Mastering JDBC">
  <topicset id="sqlbasics" href="sqlOverview.dita">
    <topicref href="sqlSelection.dita"/>
    <topicref href="sqlJoin.dita"/>
    <topicref href="sqlFilter.dita"/>
    <!-- ... -->
  </topicset>
  <topicref href="jdbcPrepare.dita"/>
  <!-- ... -->
</topichead>
```

Figure 67: Result of the reuse

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with narrowed definitions of `@href`, `@format`, and `@type`, given below), *Attributes common to many map elements (351)*, *Topicref element attributes group (358)*, *outputclass*, `@keys` *(360)*, and `@keyref` *(360)*.

@href

A pointer to the `<topicset>` represented by `<topicsetref>`. See *The href attribute (359)* for detailed information on syntax.

@format

The `@format` attribute identifies the format of the resource being referenced. For the `<topicsetref>` element, this attribute defaults to "ditamap", because the element typically references a branch of a map. See *The format attribute (372)* for details on other supported values.

@type

> Describes the target of a reference. For the `<topicsetref>` element, this attribute defaults to "topicset". See *The type attribute (370)* for detailed information on other supported values and processing implications.

3.4 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.

3.4.1 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.

3.4.1.1 <prolog>

The `<prolog>` element contains information about the topic as an whole (for example, author information or subject category) that is either entered by the author or maintained by a software application. Much of the metadata inside the `<prolog>` will not be displayed with the topic when the topic is rendered, but it might be used by processes that generate search indexes or customize navigation.

Content models

See *appendix (438)* for information about this element in OASIS document type shells.

Inheritance

- topic/prolog

Example

```
<prolog>
  <metadata>
    <audience type="user" job="using" experiencelevel="novice"/>
  </metadata>
</prolog>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.4.1.2 <audience>

The `<audience>` metadata element indicates, through the value of its `@type` attribute, the intended audience for a topic.

Since a topic can have multiple audiences, you can include multiple audience elements. For each audience you specify, you can identify the high-level task they are trying to accomplish with the `@job` attribute, and the level of experience expected 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.

Many of the attributes on the `<audience>` element have enumerated values, which can be restricted by using constraints or extended by using associated attributes. For instance, the `@othertype` attribute can be used to extend the audience type enumeration.

Content models

See *appendix (410)* for information about this element in OASIS document type shells.

Inheritance

- topic/audience

Example

For a command reference topic for experienced programmers, the following might be an appropriate indication of that audience:

```
<audience type="programmer" job="programming" experiencelevel="expert"/>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and the attributes defined below.

@type

Indicates the kind of person for whom the content of the topic is intended. Note that this differs from the `@type` attribute on many other DITA elements. Beginning with DITA 1.2, values in this attribute are not limited to a small number of choices; the following values were used in DITA 1.0 and DITA 1.1, and are still provided as sample values: user, purchaser, administrator, programmer, executive, services, other, and *-dita-use-conref-target (362)*.

@othertype

Indicates an alternate audience type, when the type is not available in the `@type` attribute value list. This value is used as the user-provided audience when the `@type` attribute value is set to "other."

@job

Indicates 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. Beginning with DITA 1.2, values in this attribute are not limited to a small number of choices; the following values were used in DITA 1.0 and DITA 1.1, and are still provided as sample values: installing, customizing, administering, programming, using, maintaining, troubleshooting, evaluating, planning, migrating, other, and *-dita-use-conref-target (362)*.

@otherjob

If the `@job` attribute value is "other" the value of this attribute is used to identify a kind of job other than the default ones provided by the `@job` attribute.

@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. Beginning with DITA 1.2, values in this attribute are not limited to a small number of choices; the following values were used in DITA 1.0 and DITA 1.1, and are still provided as sample values: novice, general, expert, and *-dita-use-conref-target (362)*.

@name

Used to associate the `<audience>` element with values used in the `@audience` attribute.

3.4.1.3 <author>

The `<author>` metadata element contains the name of the topic's author.

The author is usually the person, organization, or application that created the content. This element is equivalent to the `<Creator>` element in Dublin Core.

Content models

See *appendix (411)* for information about this element in OASIS document type shells.

Inheritance

- topic/author

Example

```
<prolog>
  <author type="creator">Jane</author>
  <author type="contributor">John</author>
</prolog>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with a narrowed definition for `@type`, given below), and *@keyref (360)*.

@type

Describes the target of a reference. See *The type attribute (370)* for detailed information on supported values and processing implications. Note that this differs from the `@type` attribute on many other DITA elements. Beginning with DITA 1.2, values in this attribute are not limited to a small number of choices; the following values were used in DITA 1.0 and DITA 1.1, and are also recognized for the `<author>` element (and its specializations):

creator

The primary or original author of the content.

contributor

An additional author who is not primary.

-dita-use-conref-target

See *Using the -dita-use-conref-target value (362)* for more information.

3.4.1.4 <brand>

The `<brand>` element indicates the manufacturer or brand associated with the product described by the parent `<prodinfo>` element.

Content models

See *appendix (412)* for information about this element in OASIS document type shells.

Inheritance

- topic/brand

Example

```
<prodinfo>
 <prodname>Some Product</prodname>
 <vrmlist><vrm version="1"/></vrmlist>
 <brand>eServer</brand>
 <series>iSeries</series>
</prodinfo>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.4.1.5 <category>

The `<category>` element represents any category by which a topic might be classified for retrieval or navigation. For example, the categories could be used to group topics in a generated navigation bar. Topics can belong to multiple categories.

Such classifications are likely to come from an enumerated or hierarchical set.

This element is equivalent to both the `<Coverage>` element and the `<Subject>` element in Dublin Core.

Content models

See *appendix (413)* for information about this element in OASIS document type shells.

Inheritance

- topic/category

Example

```
<prolog>
 <metadata>
  <category>Things that are blue</category>
 </metadata>
</prolog>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.4.1.6 <component>

The `<component>` element describes the component of the product that this topic is concerned with. For example, 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. 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.

Content models

See *appendix (413)* for information about this element in OASIS document type shells.

Inheritance

- topic/component

Example

```
<prodinfo>
 <prodname>BatCom</prodname>
 <vrmlist>
  <vrm version="v5r2"/>
 </vrmlist>
 <component>TCP/IP</component>
</prodinfo>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.4.1.7 <copyright>

The `<copyright>` element specifies legal ownership of the content.

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.

Content models

See *appendix (414)* for information about this element in OASIS document type shells.

Inheritance

- topic/copyright

Example

```
<prolog>
 <copyright>
  <copyryear year="2001-04-12"></copyryear>
  <copyrholder>IBM</copyrholder>
 </copyright>
 <copyright type="secondary">
  <copyryear year="2002-03-03"></copyryear>
  <copyrholder>Schweetones Publishing, Inc.</copyrholder>
 </copyright>
</prolog>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* 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. Beginning with DITA 1.2, values in this attribute are not limited to a small number of choices; the following values were used in DITA 1.0 and DITA 1.1, and are still provided as sample values:

primary

The copyright holder with first claim on the copyright

secondary

An additional copyright holder who is not primary

-dita-use-conref-target

See *Using the -dita-use-conref-target value (362)* for more information.

3.4.1.8 <copyrholder>

The `<copyrholder>` element names the copyright holder that holds legal rights to the material contained in the topic.

Content models

See *appendix (414)* for information about this element in OASIS document type shells.

Inheritance

- topic/copyrholder

Example

```
<copyright>
  <copyryear year="2001"></copyryear>
  <copyrholder>IBM</copyrholder>
</copyright>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.4.1.9 <copyryear>

The `<copyryear>` element contains the copyright year as specified by the `@year` attribute.

Content models

See *appendix (414)* for information about this element in OASIS document type shells.

Inheritance

- topic/copyryear

Example

```
<copyright>
  <copyryear year="2001"></copyryear>
  <copyrholder>IBM</copyrholder>
</copyright>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and the attribute defined below.

@year
> The year in YYYY format.

3.4.1.10 <created>

The `<created>` element specifies the document creation date using the `@date` attribute.

Content models

See *appendix (415)* for information about this element in OASIS document type shells.

Inheritance

- topic/created

Example

```
<prolog>
 <critdates>
  <created date="2001-06-12"></created>
  <revised golive="2001-08-20"></revised>
 </critdates>
</prolog>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Date attributes group (355)*, and the attribute defined below.

@date (REQUIRED)
> The document creation date. Enter the date 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. See *A Summary of the International Standard Date and Time Notation* for background.

3.4.1.11 <critdates>

The `<critdates>` element contains the critical dates in a document life cycle, such as the creation date and multiple revision dates.

This element is equivalent to the `<Date>` element in Dublin Core.

Content models

See *appendix (415)* for information about this element in OASIS document type shells.

Inheritance

- topic/critdates

Example

```
<prolog>
  <critdates>
    <created date="2001-06-12"></created>
    <revised modified="2001-08-20"></revised>
  </critdates>
</prolog>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.4.1.12 <featnum>

The `<featnum>` element contains the feature number of a product in the metadata.

Content models

See *appendix (422)* for information about this element in OASIS document type shells.

Inheritance

- topic/featnum

Example

```
<prodinfo>
 <prodname>BatCom</prodname>
 <vrmlist>
  <vrm version="v5r2"/>
 </vrmlist>
 <featnum>135</featnum>
 <component>TCP/IP</component>
</prodinfo>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.4.1.13 <keywords>

The `<keywords>` element contains a list of terms from a controlled or uncontrolled subject vocabulary that applies to the topic or map. The keywords can be used by a search engine. The keywords are marked up using the `<indexterm>` and/or `<keyword>` elements.

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.

> **Note:** While the *<keyword> (204)* 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.

Content models

See *appendix (428)* for information about this element in OASIS document type shells.

Inheritance

- topic/keywords

Example

The following example is metadata from an installation task:

```
<prolog>
 <metadata>
  <keywords>
   <keyword>installing</keyword>
   <keyword>uninstalling</keyword>
   <keyword>prerequisites</keyword>
   <keyword>helps</keyword>
   <keyword>wizards</keyword>
  </keywords>
 </metadata>
</prolog>
```

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)*.

3.4.1.14 <metadata>

The `<metadata>` section of the `<prolog>` contains information about a topic such as audience and product information. Metadata can be used by computational processes to select particular topics or to prepare search indexes or to customize navigation. Elements inside of `<metadata>` provide information about the content and subject of a topic; `<prolog>` elements outside of `<metadata>` provide lifecycle information for the content unit (such as the author or copyright), which are unrelated to the subject.

Beginning with DITA 1.2, the `<metadata>` element is available inside `<topicmeta>` in maps, although the contents of `<metadata>` are still available directly inside `<topicmeta>`. As with the `<prolog>`, the `<metadata>` element within `<topicmeta>` allows you to group elements that describe the content or subject of the target. The primary purpose for enabling the `<metadata>` element within maps is to allow easier reuse between topics and maps.

Content models

See *appendix* *(433)* for information about this element in OASIS document type shells.

Inheritance

- topic/metadata

Example

Metadata within a topic:

```
<prolog>
  <metadata>
    <audience type="user" job="using" experiencelevel="novice"/>
  </metadata>
</prolog>
```

Metadata within a map:

```
<topicref href="metadata.dita" navtitle="metadata element">
  <topicmeta>
    <metadata>
      <keywords>
        <indexterm>metadata element</indexterm>
      </keywords>
    </metadata>
```

```
    </topicmeta>
  </topicref>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *mapkeyref*.

3.4.1.15 <othermeta>

The `<othermeta>` element can be used to identify properties not otherwise included in `<metadata>` and to assign name/content values to those properties. The `@name` attribute identifies the property and the `@content` attribute specifies the property's value. All `<othermeta>` elements are considered part of the topic's metadata and should be reflected in the output as appropriate for the output medium.

Content models

See *appendix (435)* for information about this element in OASIS document type shells.

Inheritance

- topic/othermeta

Example

```
<othermeta name="ThreadWidthSystem" content="metric"/>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and the attributes defined below.

@name (REQUIRED)
   The name of the metadata property.

@content (REQUIRED)
   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 (362)*.

3.4.1.16 <permissions>

The `<permissions>` element in a topic's metadata specifies the level of entitlement needed to access the content.

The `<permissions>` element indicates any preferred controls for access to content.

Content models

See *appendix (436)* for information about this element in OASIS document type shells.

Inheritance

- topic/permissions

Example

```
<prolog>
  <permissions view="entitled"/>
</prolog>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and the attribute defined below.

@view

Defines the classifications of viewers allowed to view the document. Beginning with DITA 1.2, values in this attribute are not limited to a small number of choices; the following values were used in DITA 1.0 and DITA 1.1, and are still provided as sample values:

internal

For internal use only.

classified

For a certain group, only.

all

The world.

entitled

Special folks, only.

-dita-use-conref-target

See *Using the -dita-use-conref-target value (362)* for more information.

3.4.1.17 <platform>

The `<platform>` metadata element contains a description of the operating system and/or hardware related to the product being described by the `<prodinfo>` element. The `<platform>` element can be used to provide a more detailed definition of values used throughout the map or topic on the `@platform` attribute.

Content models

See *appendix (437)* for information about this element in OASIS document type shells.

Inheritance

- topic/platform

Example

See *prodinfo (265)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.4.1.18 <prodinfo>

The `<prodinfo>` metadata element contains information about the product or products that are the subject matter of the current topic. 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.

Content models

See *appendix (437)* for information about this element in OASIS document type shells.

Inheritance

- topic/prodinfo

Example

```
<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>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.4.1.19 <prodname>

The `<prodname>` metadata element contains the name of the product that is supported by the information in this topic.

Content models

See *appendix (437)* for information about this element in OASIS document type shells.

Inheritance

- topic/prodname

Example

See *prodinfo (265)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.4.1.20 <prognum>

The `<prognum>` metadata element identifies the program number of the associated product. This is typically an order number or a product tracking code that could be replaced by an order number when a product completes development.

Content models

See *appendix (438)* for information about this element in OASIS document type shells.

Inheritance

- topic/prognum

Example

See *prodinfo (265)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.4.1.21 `<publisher>`

The `<publisher>` metadata element contains the name of the person, company, or organization responsible for making the content or subject of the topic available.

This element is equivalent to the `<Publisher>` element in Dublin Core.

Content models

See *appendix (438)* for information about this element in OASIS document type shells.

Inheritance

- topic/publisher

Example

```
<prolog>
  <author>Ivan</author>
  <publisher>AJ Printing Inc.</publisher>
</prolog>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)*, and *@keyref (360)*.

3.4.1.22 `<resourceid>`

The `<resourceid>` element provides an identifier for applications that must use their own identifier scheme, such as context-sensitive help systems and databases.

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.

Versions of DITA before 1.3 used the `@id` attribute on the `<resourceid>` element to specify an ID for an external application. Beginning with DITA 1.3, this use of the `@id` attribute is deprecated in favor of using the `@appid` attribute.

Content models

See *appendix (440)* for information about this element in OASIS document type shells.

Inheritance

- topic/resourceid

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>
```

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)* (with a narrowed definition of `@id`, given below) and the attributes defined below.

@id

Before DITA 1.3, this attribute specified a value that was used by a specific application to identify the topic. That usage is deprecated in favor of using the `@appid` attribute. This attribute is defined with the XML Data Type CDATA.

@appname

A name for the external application that references the topic.

@appid

An ID used by an application to identify the topic.

@ux-context-string

Contains 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.

Related Links

2.2.4.7 Context hooks and window metadata for user assistance (61)

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.

3.4.1.23 <revised>

The `<revised>` metadata element is used to maintain tracking dates that are important in a topic development cycle, such as the last modification date, the original availability date, and the expiration date.

Content models

See *appendix (441)* for information about this element in OASIS document type shells.

Inheritance

- topic/revised

Example

```
<prolog>
  <critdates>
    <created date="1999-01-01" golive="1999-02-15" expiry="9999-09-09"/>
    <revised modified="2003-03-03" golive="2002-02-03" expiry="9999-09-09"/>
  </critdates>
</prolog>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Date attributes group (355)*, 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.

3.4.1.24 <series>

The `<series>` metadata element contains information about the product series that the topic supports.

Content models

See *appendix (442)* for information about this element in OASIS document type shells.

Inheritance

- topic/series

Example

```
<prodinfo>
 <prodname>BatCom</prodname>
 <vrmlist><vrm version="5"/></vrmlist>
 <series>tSeries</series>
 <prognum>5412-SS1</prognum>
 <featnum>135</featnum>
 <component>TCP/IP</component>
</prodinfo>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.4.1.25 <source>

The `<source>` element identifies a resource from which the present topic is derived, either completely or in part.

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. It is implementation-dependent what it means when the element has both content and an attribute-based reference to another resource.

This element is equivalent to the `<Source>` element in Dublin Core.

Content models

See *appendix (444)* for information about this element in OASIS document type shells.

Inheritance

- topic/source

Example

```
<prolog>
 <source>Somewhere, someplace</source>
</prolog>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with a narrowed definition for `@href`, given below), and *@keyref (360)*.

@href
 Provides a reference to a resource from which the present resource is derived. See *The href attribute (359)* for detailed information on supported values and processing implications.

3.4.1.26 <vrmlist>

The `<vrmlist>` element contains a set of `<vrm>` elements for logging the version, release, and modification information for multiple products or versions of products to which the topic applies.

Content models

See *appendix (455)* for information about this element in OASIS document type shells.

Inheritance

- topic/vrmlist

Example

The recent versions of a hypothetical product might be logged thus using the `<vrmlist>` markup:

```
<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).

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.4.1.27 <vrm>

The `<vrm>` element contains information about a single product's version, modification, and release, to which the current topic applies.

Content models

See *appendix (454)* for information about this element in OASIS document type shells.

Inheritance

- topic/vrm

Example

The recent versions of a hypothetical product might be logged thus using the `<vrmlist>` markup:

```
<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).

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and the attributes defined below.

@version (REQUIRED)

Indicates the released version number of the product(s) that the document describes.

@release

Contains the product release identifier.

@modification

Indicates the modification level of the current version and release.

3.4.2 Indexing group elements

The indexing domain provides elements for use with indexing. The elements allow authors to define "See" and "See also" references, and to override the default sort order for a term.

Indexing domain elements typically work with the `<indexterm>` and `<index-base>` elements; `<index-base>` is grouped with elements that are typically useful only in specialization contexts.

Related Links

3.7.5 index-base *(335)*

The `<index-base>` *element allows indexing extensions to be added by specializing this element.*

index-base *(335)*

The `<index-base>` *element allows indexing extensions to be added by specializing this element.*

3.4.2.1 <indexterm>

The content of an `<indexterm>` element is used to produce an index entry in a generated index. You can nest `<indexterm>` elements to create multi-level indexes. The content is not output as part of the topic content, only as part of an index.

An `<indexterm>` element without the `@start` or `@end` attribute specified is interpreted as a point reference that contributes the number of the current page to an index entry; the content of the index entry is the content of the `<indexterm>` element. All `<indexterm>` elements 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.

In the case of nested `<indexterm>` elements, each `<indexterm>` with no `<indexterm>` children (the "leaves") contributes a page number to the generated index; the ancestral `<indexterm>` elements for each leaf `<indexterm>` provide the higher levels for the multilevel entry.

An `<indexterm>` that occurs in a topic `<prolog>` is interpreted as a point reference to the title of the topic. Likewise, in a DITA map, an `<indexterm>` that occurs in `<topicmeta>` inside of a `<topicref>` is interpreted as a point reference to the title of the referenced topic.

It is an error if an `<indexterm>` containing no `<indexterm>` children contains both an `<index-see>` and an `<index-see-also>`. (Note: `<index-see>` and `<index-see-also>` elements within `<indexterm>` elements that do contain `<indexterm>` children are ignored.) In the case of this error condition, an implementation *MAY* give an error message, and might recover by treating all such `<index-see>` elements as `<index-see-also>` elements.

Note: The `<index-see>` and `<index-see-also>` elements are domain specializations of the `<index-base>` element, and are discussed in detail with the indexing domain.

The `@start` and `@end` attribute on `<indexterm>` can be used in cases where one wants to index an extended discussion that might continue over a number of pages. The start of a range is indicated by an `<indexterm>` with a `@start` attribute. The end of a range is indicated with an `<indexterm>` with an `@end` attribute whose value matches that of the `@start` attribute on the start-of-range `<indexterm>`. Such markup contributes to the generated index a page range covering all pages in the index range.

The end-of-range `<indexterm>` should have no content of its own; if it contains content, that content is ignored. There is no reason for the end-of-range `<indexterm>` to have any `<indexterm>` ancestors; however, an implementation should be able to handle an end-of-range `<indexterm>` that is nested within one or more `<indexterm>` elements.

The `@start` and `@end` attributes are defined as CDATA, although it is a best practice that the values should not contain any whitespace characters (such as a space or tab) or control characters. Matching of `@start` and `@end` attributes is done as a character-by-character comparison with all characters significant and no case folding

occurring. The `@start` and `@end` attributes are ignored if they occur on an `<indexterm>` element that has child `<indexterm>` elements.

Index range indications can occur in the `<topicmeta>` of a `<topicref>` at the map level, in the prolog of a topic, or in the body of a topic, and are interpreted as follows (see *Figure 70: Index ranges (274)* for samples):

- In a map, the start range points to the start of the topic title of the topic being referenced by its containing `<topicref>`. The end range points to the end of the final child contained by the topic being referenced by its containing `<topicref>`, or to the end of the final topic referenced by the current map (whichever comes first). When a start and end range occur in the same `<topicmeta>`, the range applies to the containing `<topicref>` and its children.

- In the prolog of a topic, the start range points to the start of the containing topic's title. The range ends with a matching index range end in the same prolog, regardless of whether the end range is specified. The range applies to the containing topic and all its children including child relationships defined in a map.

- In the body of a topic, the range starts where the start `<indexterm>` occurs and ends at the matching index range end indication within the same body, or at the end of the body, whichever comes first. Such an index range does not span sub-topics of the topic.

When index ranges with the same identifier overlap, the widest range applies, and end ranges are matched with start ranges by last-in-first-out. In other words, the ranges are interpreted as nested rather than overlapping with the highest-level container taking precedence over narrower contained ranges.

As defined above, there is no such thing as an index range start that isn't terminated by either a matching end or some maximum scope. There can, however, be unmatched index range end indications; these are ignored.

Content models

See *appendix (427)* for information about this element in OASIS document type shells.

Inheritance

- topic/indexterm

Example

- The following `<indexterm>` is a point reference to a specific paragraph within a topic:

```
<p><indexterm>databases</indexterm>Databases are used to ...</p>
```

- The following `<indexterm>` is a point reference to the start of the title of the concept:

```
<concept id="db">
  <title>About databases</title>
  <prolog>
    <metadata>
      <keywords><indexterm>databases</indexterm></keywords>
    </metadata>
  </prolog>
  <body><!-- content... --></body>
</concept>
```

- The following `<indexterm>` is a point reference to the start of the title of `aboutdatabases.dita`:

```
<topicref href="aboutdatabases.dita">
  <topicmeta>
    <keywords><indexterm>databases</indexterm></keywords>
  </topicmeta>
  <!-- other topicref elements -->
</topicref>
```

Figure 68: Single point index terms

The following sample represents three levels of index markup:

```
<indexterm>cheese
  <indexterm>sheeps milk
    <indexterm>pecorino</indexterm>
  </indexterm>
  <indexterm>goats milk
    <indexterm>chevre</indexterm>
  </indexterm>
</indexterm>
```

The previous sample is equivalent to the following sample:

```
<indexterm>cheese
  <indexterm>sheeps milk
    <indexterm>pecorino</indexterm>
  </indexterm>
</indexterm>
<indexterm>cheese
  <indexterm>goats milk
    <indexterm>chevre</indexterm>
  </indexterm>
</indexterm>
```

In each case, a generated index would include something like the this:

- cheese

  – goats milk

    - chevre 14

  – sheeps milk

    - pecorino 14

Figure 69: Nested index terms

A simple index range will look something like this:

```
<indexterm start="cheese">Cheese</indexterm>
<!-- ... additional content -->
<indexterm end="cheese"/>
```

The previous combination of terms will generate a top-level index term for "Cheese" that covers a series of pages, such as:

- Cheese 18-24

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"/>
```

The generated index for that range would look something like this:

- cheese

  – sheeps milk

    - pecorino 18-24

There are three locations that can declare a range - the body of a topic, the prolog of a topic, and a map.

- In the following example, the range begins at the start of the second paragraph, and continues to the last paragraph. If the matching end range was not included, the range would end at the end of the body element.

```
<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>
```

- In the following example, the range begins with the start of the topic's title, and covers the entire topic and any sub-topics. The range ends within the same prolog, regardless of whether `<indexterm end="acct"/>` is specified in the prolog.

```
<topic id="accounting">
  <title>Accounting rugulations</title>
  <prolog>
    <metadata>
      <keywords><indexterm start="acct">Accounting</indexterm></keywords>
    </metadata>
  </prolog>
  <!-- Body and sub-topics -->
</topic>
```

- Now assume that the topic in the previous sample is named `acct.dita`. Ranges defined in a prolog cover sub-topics, including those nested based on a map; in the following example, this means that the range covers all of `acct.dita`, as well as `procedures.dita` and `forms.dita`:

```
<topicref href="acct.dita">
  <topicref href="procedures.dita"/>
  <topicref href="forms.dita"/>
</topicref>
```

- In the final example, the range is specified in a map. The index range for "Accounting" begins with the start of the first topic title in `acct.dita`, and covers that file as well as any sub-topics. The index range for "Government forms" begins with the start of the first topic title in `acct.dita`, and continues until the end of the last element in the file `taxfiling.dita`. If the end range for "govt" was not specified, the range would continue to the end of the map.

```
<topicref href="acct.dita">
  <topicmeta>
    <keywords>
      <indexterm start="acct">Accounting</indexterm>
      <indexterm end="acct"/>
      <indexterm start="govt">Government forms</indexterm>
    </keywords>
  </topicmeta>
  <!-- Nested topicref elements -->
</topicref>
<topicref href="taxfiling.dita">
  <topicmeta>
    <keywords>
      <indexterm end="govt"/>
    </keywords>
```

```
        </topicmeta>
    </topicref>
```

Figure 70: Index ranges

```
<p>Einstein's most famous equation
E=mc<sup>2</sup><indexterm>E=mc<sup>2</sup></indexterm>
expresses the relationship between mass and energy.</p>
```

All the elements in the highlighting domain are specializations of `<ph>`.

Figure 71: Index term with <ph> or <ph> specializations

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *@keyref (360)*, and the attributes defined below.

@start

Specifies that an index entry is positioned at the beginning of a range. The value matches the `@end` attribute on another `<indexterm>`.

@end

Specifies that an index entry is positioned at the end of a range; value matches the `@start` attribute on another `<indexterm>`.

3.4.2.2 <indextermref>

This element is not completely defined; it is reserved for future use.

Content models

See *appendix (427)* for information about this element in OASIS document type shells.

Inheritance

- topic/indextermref

Example

Examples will be added when this element is fully defined.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *@keyref (360)*.

3.4.2.3 <index-see>

An `<index-see>` element within an `<indexterm>` element redirects the reader to another index entry that the reader should reference instead of the current one.

The `<index-see>` and `<index-see-also>` elements allow a form of redirection to another index entry within the generated index. The `<index-see>` element refers to an index entry that the reader should use *instead of* the current one, whereas the `<index-see-also>` element refers to an index entry that the reader should use *in addition to* the current one.

Processors should ignore `<index-see>` and `<index-see-also>` elements if their parent `<indexterm>` element contains any `<indexterm>` children.

Because an `<index-see>` indicates a redirection to use instead of the current entry, it is an error if, for any `<index-see>`, there is also an `<index-see-also>` or an `<indexterm>` for the same index entry (that is, another entry with an identical sort key). For example, if an `<indexterm>` element with the content "Memory stick" also includes `<index-see>USB drive</index-see>`, it is an error if there is also an `<indexterm>`

with the contents "Memory stick". This is to prevent index entries that are both a redirect and a page reference, such as:

* Memory stick 42, 106
* See USB drive

An implementation *MAY* give an error message when it encounters this condition, and *MAY* recover from this error condition by treating the `<index-see>` as an `<index-see-also>`.

There can be multiple `<index-see>` elements for a single index entry.

Content models

See *appendix (426)* for information about this element in OASIS document type shells.

Inheritance

+ topic/index-base indexing-d/index-see

Example

The following example illustrates the use of an `<index-see>` redirection element within an `<indexterm>`:

```
<indexterm>Carassius auratus
    <index-see>Goldfish</index-see>
</indexterm>
```

This will typically generate an index entry without a page reference:

 • Carassius auratus, *see* Goldfish

The following example illustrates the use of an `<index-see>` redirection element to a more complex (multilevel) `<indexterm>`:

```
<indexterm>Feeding goldfish
    <index-see>Goldfish <indexterm>feeding</indexterm></index-see>
</indexterm>
```

This is part of the indexing markup that might generate index entries such as:

 • Feeding goldfish

   – *see* Goldfish feeding

 • Goldfish

   – feeding, 56

   – flushing, 128, 345

The following example illustrates using a specialization of `<ph>` within `<index-see>`:

```
<indexterm>Einstein's mass and energy equation
  <index-see>E=mc<sup>2</sup></index-see>
</indexterm>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *@keyref (360)*.

3.4.2.4 <index-see-also>

An `<index-see-also>` element within an `<indexterm>` redirects the reader to another index entry that the reader should reference in addition to the current one.

The `<index-see>` and `<index-see-also>` elements allow a form of redirection to another index entry within the generated index. The `<index-see>` element refers to an index entry that the reader should use *instead of* the current one, whereas the `<index-see-also>` element refers to an index entry that the reader should use *in addition to* the current one.

Processors should ignore `<index-see>` and `<index-see-also>` elements if their parent `<indexterm>` element contains any `<indexterm>` children.

In addition to its "see also" redirection, an `<index-see-also>` functions as a pointwise index term, thereby typically generating a page reference as well as the "see also" indication.

It is not an error for there to be multiple `<index-see-also>` elements for a single index entry.

Content models

See *appendix (426)* for information about this element in OASIS document type shells.

Inheritance

+ topic/index-base indexing-d/index-see-also

The following example illustrates the use of an `<index-see-also>` redirection element within an `<indexterm>`:

```
<indexterm>Carp
    <index-see-also>Goldfish</index-see-also>
</indexterm>
```

This will typically generate a page reference to "Carp" and a redirection:

- Carp, 56
  - *see also* Goldfish

The following example illustrates the use of an `<index-see-also>` redirection element to a more complex (multilevel) `<indexterm>`:

```
<indexterm>Feeding
    <index-see-also>Goldfish <indexterm>feeding</indexterm></index-see-also>
</indexterm>
```

This is part of the indexing markup that might generate index entries such as:

- Feeding, 348
  - *see also* Goldfish feeding
- Goldfish
  - feeding, 56
  - flushing, 128, 345

The following example illustrates using a specialization of `<ph>` within `<index-see-also>`:

```
<indexterm>μ = E<sub>0</sub>/V<sup>2</sup>
  <index-see-also>E=mc<sup>2</sup></index-see-also>
</indexterm>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and `@keyref` *(360)*.

3.4.2.5 <index-sort-as>

The `<index-sort-as>` element specifies a sort phrase under which an index entry would be sorted.

This element 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.

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.

When an `<index-sort-as>` element is specified, processors that sort the containing index term *MUST* construct the effective sort phrase by prepending the content of the `<index-sort-as>` element to the textual content of its parent `<indexterm>` element. This ensures that two index entries with the same `<index-sort-as>` element but different base sort phrases will sort in the appropriate order, and will not merge into a single index entry.

An `<index-sort-as>` element provides sort phrase information for the `<indexterm>` that is its parent; therefore, in a multiple level `<indexterm>`, the `<index-sort-as>` only affects the level in which it occurs.

It is an error if there is more than one `<index-sort-as>` child for a given `<indexterm>`. An implementationmight give an error message, and might recover from this error condition by ignoring all but the last `<index-sort-as>`.

When located within the `<indexterm>` element, the `<sort-as>` element is equivalent to `<index-sort-as>`. It is an error for an `<indexterm>` element to directly contain both `<sort-as>` and `<index-sort-as>` elements.

Content models

See *appendix (426)* for information about this element in OASIS document type shells.

Inheritance

+ topic/index-base indexing-d/index-sort-as

This is an example of an index entry for `<data>` that will be sorted as "data":

```
<indexterm>&lt;data&gt;<index-sort-as>data</index-sort-as></indexterm>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and `@keyref` *(360)*.

Related Links

> *2.4.8 Sorting (122)*
>> *Processors can be configured to sort elements. Typical processing includes sorting glossary entries, lists of parameters or reference entries in custom navigation structures, and tables based on the contents of cells in specific columns or rows.*

3.4.3 Delayed conref resolution elements

The delayed conref resolution domain provides several elements for use when using DITA in situations that enable delayed or delivery time resolution of conref. The elements allow users to resolve some conref values statically, while delaying others for later resolution.

Many publishing systems for which DITA is used as a source format do not have a way to dynamically resolve content references; those systems will not see any benefit from this domain. When DITA is used for those systems, behaviors related to this domain should be ignored.

3.4.3.1 <exportanchors>

The `<exportanchors>` element is used to delay `@conref` resolution within DITA documents. This allows you to process or display DITA content in a way that will resolve only some of the `@conref` values in that content, while remaining values are left for later resolution. The element contains a list of IDs or keys that are not resolved during the initial preparation of the content for display; those IDs and keys will be preserved after that preparation, as will the conref relationship itself.

The `<exportanchors>` element canbe used within a topic prolog, in which case the defined IDs apply to IDs within that topic (excluding sub-topics). Alternatively it can be defined in a `<topicmeta>` element in a map. In the second case the IDs apply to the single topic referenced by the current `<topicref>` element. If the `<topicref>` references a file without referencing a specific topic, it is treated as a reference to the first or root topic. In order to define anchor ids for a topic that is not the first or root topic, a `<topicref>` must directly reference the desired sub-topic.

> **Note:** When an element's ID is defined for delayed resolution, it must contain only the element ID, not the usual "topicid/elementid" syntax that is required for most other DITA references. The *<anchorid> (283)* topic explains the format in detail.

One possible way to use this is with a system that renders DITA dynamically. A user can process information locally in a way that resolves `@conref` for all static information, while delaying resolution for information that is subject to change. The `<exportanchors>` element is used to define `@conref` values that are delayed.

Another potential use is when DITA is used as the source format for a publishing system that is able to render information dynamically. In this case some `@conref` values might be resolved, while leaving pre-selected values to be resolved live in that publishing system.

Many publishing systems for which DITA is used as a source format do not have a way to dynamically resolve content references; those systems will not see any benefit from this element. When DITA is used for those systems, behaviors related to this element are ignored.

Content models

See *appendix (421)* for information about this element in OASIS document type shells.

Inheritance

+ topic/keywords delay-d/exportanchors

Example

1. Author 1 creates topics for information component A, which is a common component used by many products. The configuration task for component A is often reused in whole or in part, so the author assigns ids to each of the steps in the procedure and exports them.

```
<task id="configA">
  <title>ABC</title>
  <shortdesc>...</shortdesc>
  <prolog><metadata>
    <exportanchors>
      <anchorid id="step1"/>
```

```
        <anchorid id="step2"/>
        <anchorid id="step3"/>
      </exportanchors>
  </metadata></prolog>
  <taskbody>
    <steps>
      <step id="step1"><cmd>Do this</cmd></step>
      <step id="step2"><cmd>Do the other</cmd></step>
      <step id="step3"><cmd>And then finish</cmd></step>
    </steps>
  </taskbody>
</task>
```

2. Author 2 is working on information component B, which has information component A as a prerequisite.

3. Author 2 creates a configuration task that reuses two steps from the configuration task in information component A.

```
<task id="configB">
  <title>..</title>
  <shortdesc>..</shortdesc>
  <taskbody>
    <steps>
      <step><cmd>Do the very first thing</cmd></step>
      <step conref="componentA/configA.dita#configA/step1"><cmd/></step>
      <step><cmd>Do the middle thing</cmd></step>
      <step conref="componentA/configA.dita#configA/step2"><cmd/></step>
    </steps>
  </taskbody>
</task>
```

4. Author 2 builds the content for component B into a deliverable format that supports dynamic content resolution. As with traditional conref, the source for component A must be available during this process. Because the ids in configA are exported, the build process knows to preserve the reuse relationship rather than resolve it - so the `@conref` reference to the steps becomes an equivalent reuse artifact in that deliverable format. This way the relationship to component A can be resolved at runtime, and pick up the user's version of component A, which might be more up-to-date than the one used by Author 2 when component B was built.

Figure 72: Use case 1: Runtime resolution of @conref to an id, determined by original author

1. Author 1 is creating content that will be packaged into multiple deliverable components. In one of those components, component A, the ids should be exported for runtime reuse by other components. In other components, the ids should not be exported because all reuse is local (for example, the output is a single infocenter, or a helpset that has only one component).

2. When author 1 builds component A, the author uses a map that exports the ids, rather than exporting the ids from the topic `<prolog>`.

```
<map>
  <topicref href="componentA/configA.dita">
    <topicmeta>
      <exportanchors>
        <anchorid id="step1"/>
        <anchorid id="step2"/>
        <anchorid id="step3"/>
      </exportanchors>
    </topicmeta>
  </topicref>
</map>
```

3. The rest of the use case is the same as previous - the `@conref` reference is passed on to the runtime/ display format to deal with, rather than being resolved during native DITA processing.

Figure 73: Use case 2: Runtime resolution to an id exported by the information builder

The ID on an `<anchorid>` element is first compared with the topic's id, and then with elements inside that topic. This results in the following situation.

```
<map>
  <topicref href="componentA/this.dita">
    <topicmeta>
      <exportanchors>
        <anchorid id="this"/>
        <anchorid id="that"/>
      </exportanchors>
    </topicmeta>
  </topicref>
</map>

<topic id="this">
  <title>This and that</title>
  <shortdesc>Oh, you know, this and that.</shortdesc>
  <body>
    <fig id="that"><p>more of that</p></fig>
  </body>
</topic>
```

- The first ID to be exported is "this", which matches the topic id, so resolution of `@conref` values that target the topic should be delayed.
- The second value is "that", which matches a figure within the topic, so resolution of `@conref` values that target the figure should be delayed.
- Note that if the "this" is also used within the topic (which is legal from a DITA perspective), it will not be possible to export that id, because processors will match on the topic's id first.

Figure 74: Delaying resolution for a topic

Example

In this example, a set of information contains multiple components. Some references to component A use keys rather than a direct reference, so that `@conref` can be redirected to a different component when component A is not installed. The keys might be exported, in addition to the IDs, so that some references become bound to the actual component while other references might be redirected.

```
<map>
  <topicref keys="componentAconfig commonconfig"
          href="componentA/configA.dita#configA">
    <topicmeta>
      <exportanchors>
        <anchorkey keyref="commonconfig"/>
        <anchorid id="step1"/>
        <anchorid id="step2"/>
      </exportanchors>
    </topicmeta>
  </topicref>
</map>
```

The `@keys` attributes declares two distinct keys that can be used to refer to this topic (componentAconfig and commonconfig). Only the second is preserved using `<anchorkey>`. A task topic from another component might reuse steps within this topic in a variety of ways.

```
<steps>
  <step conkeyref="componentAconfig/step1"><cmd/></step>
  <step conkeyref="componentAconfig/step1.5"><cmd/></step>
  <step conkeyref="commonconfig/step2"><cmd/></step>
  <step conkeyref="commonconfig/step2.5"><cmd/></step>
  <step><cmd>And that is the end of that</cmd></step>
</steps>
```

- The componentAconfig key is not preserved, so the first `<step>` becomes `<step conref="componentA/configA.dita#configA/step1"><cmd/></step>`. At that point the

`<anchorid>` element instructs the step1 ID to be preserved; for runtime applications which support it, this relationship will be preserved in the processed DITA output.

- The second `<step>` with the same key becomes `<step conref="componentA/configA.dita#configA/step1.5"><cmd/></step>`. However, conref relationships to step1.5 are not preserved, so this conref should be resolved into static content.

- For `<step>` three, the map instructs that both the key commonconfig and the ID step2 should be preserved in any content generated for this DITA topic. For formats that support runtime resolution through keys, a process must convert the `@conkeyref` value into an equivalent value for that format.

- Although resolution for the key used in `<step>` four is delayed, the specific element that is referenced should not be delayed. Thus the fourth step becomes `<step conref="componentA/configA.dita#configA/step2.5"><cmd/></step>`. This value is then processed as an ordinary `@conref` value.

This allows the information assembler to resolve references that must be to componentA while deferring references that can be fulfilled by alternative component content.

Note: This example demonstrates why the `<anchorid>` element cannot reference an element with the usual topicid/elementid format. If the two `<anchorid>` elements in the example had been set to config/step1 and config/step2, then they would only ever apply in a topic with id="config". It would not be possible to redirect the key to another topic, but still preserve conref behaviors as desired.

Note: Although it is not specifically called out in this example, it is possible to delay `@conref` resolution for an entire topic using the key. If `@conkeyref` on a task topic element is set to "componentAconfig", which *is not* delayed, the `@conref` will be evaluated as usual. However, if `@conkeyref` on the task is set to "commonconfig", which *is* delayed, resolution of `@conref` on that element should be delayed by a processor.

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)*.

3.4.3.2 <anchorid>

The `<anchorid>` element allows an author to define a `@conref` target that will be resolved dynamically when rendered for an end user of the content. This element is useful when doing an initial process of the DITA content prior to final rendering of the content; it causes specified IDs to be preserved after that process, and `@conref` relationships that reuse the element will not be resolved during the initial process.

When the `<anchorid>` element is defined within a topic prolog, the specified IDs will be found within that topic. When an `<anchorid>` element is defined within a `<topicref>` element, the specified IDs will be found within the referenced topic (if the `<topicref>` references a collection of topics, such as a reference that uses only a file name, the IDs will be found within the first or root topic).

The only difference between specifying an `<anchorid>` in the `<prolog>` and specifying it in the `<topicmeta>` is that from the map it is possible to export the ID of the entire referenced topic. If `<anchorid id="zero"/>` is specified in the `<topicmeta>`, and the referenced topic has an id of "zero", this means that the `<anchorid>` is a reference to the entire topic. If the topic id is not "zero", then the `<anchorid>` is a reference to the element with id="zero" within that topic.

Along with the preservation of the element's ID, any `@conref` attribute that references the element's ID will not be resolved during an initial process. In that case, the `@conref` will be resolved during a later rendering process.

This description does not imply that IDs are not discarded when `<anchorid>` is not used; though this element requires that IDs be preserved in some manner, it is also common for IDs to be preserved when `<anchorid>` is not used. Thus the primary impact of the `<anchorid>` element is on `@conref` resolution.

Many publishing systems for which DITA is used as a source format do not have a way to dynamically resolve content references; those systems will not see any benefit from this element. When DITA is used for those systems, behaviors related to this element are ignored.

Why not use topicid/elementid?

This element differs from normal DITA referencing syntax in that it can reference an element within a topic without using the topic's ID. There are two reasons for this. First, the `<anchorid>` element might only be defined in a situation that refers unambiguously to a single topic (in the `<prolog>`, or in the `<topicmeta>` for a reference to a topic). Second, it allows the `<anchorid>` to be combined with key values.

It is possible to combine an `<anchorid>` with a key in order to delay resolution of `@conref` in the topic represented by that key (see the second set of examples below). This would not be possible if the `<anchorid>` element required both the topic id and the element id. That is, `@keyref` allows a modifiable reference to a topic, so a map might instruct processors to delay conref for item "step1" in the topic represented by the key "commonconfig". If the `<anchorid>` element required a topic id, the delayed conref would always be bound to that specific topic.

Content models

See *appendix (409)* for information about this element in OASIS document type shells.

Inheritance

+ topic/keyword delay-d/anchorid

Example

1. Author 1 creates topics for information component A, which is a common component used by many products. The configuration task for component A is often reused in whole or in part, so the author assigns ids to each of the steps in the procedure and exports them.

```
<task id="configA">
  <title>ABC</title>
  <shortdesc>...</shortdesc>
  <prolog><metadata>
    <exportanchors>
      <anchorid id="step1"/>
      <anchorid id="step2"/>
      <anchorid id="step3"/>
    </exportanchors>
  </metadata></prolog>
  <taskbody>
    <steps>
      <step id="step1"><cmd>Do this</cmd></step>
      <step id="step2"><cmd>Do the other</cmd></step>
      <step id="step3"><cmd>And then finish</cmd></step>
    </steps>
  </taskbody>
</task>
```

2. Author 2 is working on information component B, which has information component A as a prerequisite.

3. Author 2 creates a configuration task that reuses two steps from the configuration task in information component A.

```
<task id="configB">
  <title>..</title>
  <shortdesc>..</shortdesc>
  <taskbody>
    <steps>
      <step><cmd>Do the very first thing</cmd></step>
      <step conref="componentA/configA.dita#configA/step1"><cmd/></step>
      <step><cmd>Do the middle thing</cmd></step>
      <step conref="componentA/configA.dita#configA/step2"><cmd/></step>
    </steps>
```

```
        </taskbody>
    </task>
```

4. Author 2 builds the content for component B into a deliverable format that supports dynamic content resolution. As with traditional conref, the source for component A must be available during this process. Because the ids in configA are exported, the build process knows to preserve the reuse relationship rather than resolve it - so the `@conref` reference to the steps becomes an equivalent reuse artifact in that deliverable format. This way the relationship to component A can be resolved at runtime, and pick up the user's version of component A, which might be more up-to-date than the one used by Author 2 when component B was built.

Figure 75: Use case 1: Runtime resolution of @conref to an id, determined by original author

1. Author 1 is creating content that will be packaged into multiple deliverable components. In one of those components, component A, the ids should be exported for runtime reuse by other components. In other components, the ids should not be exported because all reuse is local (for example, the output is a single infocenter, or a helpset that has only one component).

2. When author 1 builds component A, the author uses a map that exports the ids, rather than exporting the ids from the topic `<prolog>`.

```
<map>
    <topicref href="componentA/configA.dita">
        <topicmeta>
            <exportanchors>
                <anchorid id="step1"/>
                <anchorid id="step2"/>
                <anchorid id="step3"/>
            </exportanchors>
        </topicmeta>
    </topicref>
</map>
```

3. The rest of the use case is the same as previous - the `@conref` reference is passed on to the runtime/display format to deal with, rather than being resolved during native DITA processing.

Figure 76: Use case 2: Runtime resolution to an id exported by the information builder

The ID on an `<anchorid>` element is first compared with the topic's id, and then with elements inside that topic. This results in the following situation.

```
<map>
  <topicref href="componentA/this.dita">
    <topicmeta>
      <exportanchors>
        <anchorid id="this"/>
        <anchorid id="that"/>
      </exportanchors>
    </topicmeta>
  </topicref>
</map>

<topic id="this">
  <title>This and that</title>
  <shortdesc>Oh, you know, this and that.</shortdesc>
  <body>
    <fig id="that"><p>more of that</p></fig>
  </body>
</topic>
```

• The first ID to be exported is "this", which matches the topic id, so resolution of `@conref` values that target the topic should be delayed.

• The second value is "that", which matches a figure within the topic, so resolution of `@conref` values that target the figure should be delayed.

- Note that if the "this" is also used within the topic (which is legal from a DITA perspective), it will not be possible to export that id, because processors will match on the topic's id first.

Figure 77: Delaying resolution for a topic

Example

In this example, a set of information contains multiple components. Some references to component A use keys rather than a direct reference, so that `@conref` can be redirected to a different component when component A is not installed. The keys might be exported, in addition to the IDs, so that some references become bound to the actual component while other references might be redirected.

```
<map>
  <topicref keys="componentAconfig commonconfig"
          href="componentA/configA.dita#configA">
    <topicmeta>
      <exportanchors>
        <anchorkey keyref="commonconfig"/>
        <anchorid id="step1"/>
        <anchorid id="step2"/>
      </exportanchors>
    </topicmeta>
  </topicref>
</map>
```

The `@keys` attributes declares two distinct keys that can be used to refer to this topic (componentAconfig and commonconfig). Only the second is preserved using `<anchorkey>`. A task topic from another component might reuse steps within this topic in a variety of ways.

```
<steps>
  <step conkeyref="componentAconfig/step1"><cmd/></step>
  <step conkeyref="componentAconfig/step1.5"><cmd/></step>
  <step conkeyref="commonconfig/step2"><cmd/></step>
  <step conkeyref="commonconfig/step2.5"><cmd/></step>
  <step><cmd>And that is the end of that</cmd></step>
</steps>
```

- The componentAconfig key is not preserved, so the first `<step>` becomes `<step conref="componentA/configA.dita#configA/step1"><cmd/></step>`. At that point the `<anchorid>` element instructs the step1 ID to be preserved; for runtime applications which support it, this relationship will be preserved in the processed DITA output.
- The second `<step>` with the same key becomes `<step conref="componentA/configA.dita#configA/step1.5"><cmd/></step>`. However, conref relationships to step1.5 are not preserved, so this conref should be resolved into static content.
- For `<step>` three, the map instructs that both the key commonconfig and the ID step2 should be preserved in any content generated for this DITA topic. For formats that support runtime resolution through keys, a process must convert the `@conkeyref` value into an equivalent value for that format.
- Although resolution for the key used in `<step>` four is delayed, the specific element that is referenced should not be delayed. Thus the fourth step becomes `<step conref="componentA/configA.dita#configA/step2.5"><cmd/></step>`. This value is then processed as an ordinary `@conref` value.

This allows the information assembler to resolve references that must be to componentA while deferring references that can be fulfilled by alternative component content.

Note: This example demonstrates why the `<anchorid>` element cannot reference an element with the usual topicid/elementid format. If the two `<anchorid>` elements in the example had been set to config/step1 and config/step2, then they would only ever apply in a topic with id="config". It would not be possible to redirect the key to another topic, but still preserve conref behaviors as desired.

Note: Although it is not specifically called out in this example, it is possible to delay `@conref` resolution for an entire topic using the key. If `@conkeyref` on a task topic element is set to "componentAconfig", which *is not* delayed, the `@conref` will be evaluated as usual. However, if `@conkeyref` on the task is set to "commonconfig", which *is* delayed, resolution of `@conref` on that element should be delayed by a processor.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (with a narrowed definition of `@id`, given below), *@keyref (360)*, and *outputclass*.

@id (REQUIRED)

Indicates an ID within the specific topic that will be preserved during processing. Any `@conref` values referencing the indicated ID will not be resolved; when possible, the original relationship is preserved in any processed document. Note that this element creates an exception to the general rules that IDs can only be used once within a single topic or within a map; this is because the ID is actually a pointer to another target, rather than being a target itself. This attribute is defined with the XML Data Type CDATA.

3.4.3.3 <anchorkey>

The `<anchorkey>` element allows an author to define a `@conref` target that will be resolved dynamically when rendered for an end user of the content. This element is useful when doing an initial process of the DITA content prior to final rendering of the content; it allows specified keys to be preserved after that process, and conref relationships which use that key will not be resolved during that initial process.

When a `@keyref` attribute is specified on an `<anchorkey>` element, it indicates that any conref relationships using that key will not be resolved. Applications that support run-time resolution of `@conref` with keys will then be able to dynamically resolve this `@conref` at display time.

There is no difference between specifying `<anchorkey>` within a map (in `<topicmeta>`) and specifying `<anchorkey>` within a topic. In both cases, processors are instructed to delay resolution of that key for the current set of information. However, the best practice is to only use `<anchorkey>` within a map. If it is specified in a topic, that topic will define a usage for the key for every user of that topic. This makes the topic less portable, because users that do not want to delay resolution of that specific key will not be able to include the topic in their information.

Many publishing systems for which DITA is used as a source format do not have a way to dynamically resolve content references; those systems will not see any benefit from this element. When DITA is used for those systems, behaviors related to this element are ignored.

Content models

See *appendix (410)* for information about this element in OASIS document type shells.

Inheritance

+ topic/keyword delay-d/anchorkey

Example

In this example, a set of information contains multiple components. Some references to component A use keys rather than a direct reference, so that `@conref` can be redirected to a different component when component A is not installed. The keys might be exported, in addition to the IDs, so that some references become bound to the actual component while other references might be redirected.

```
<map>
  <topicref keys="componentAconfig commonconfig"
         href="componentA/configA.dita#configA">
    <topicmeta>
      <exportanchors>
        <anchorkey keyref="commonconfig"/>
        <anchorid id="step1"/>
```

```
            <anchorid id="step2"/>
        </exportanchors>
      </topicmeta>
    </topicref>
</map>
```

The `@keys` attributes declares two distinct keys that can be used to refer to this topic (componentAconfig and commonconfig). Only the second is preserved using `<anchorkey>`. A task topic from another component might reuse steps within this topic in a variety of ways.

```
<steps>
  <step conkeyref="componentAconfig/step1"><cmd/></step>
  <step conkeyref="componentAconfig/step1.5"><cmd/></step>
  <step conkeyref="commonconfig/step2"><cmd/></step>
  <step conkeyref="commonconfig/step2.5"><cmd/></step>
  <step><cmd>And that is the end of that</cmd></step>
</steps>
```

- The componentAconfig key is not preserved, so the first `<step>` becomes `<step conref="componentA/configA.dita#configA/step1"><cmd/></step>`. At that point the `<anchorid>` element instructs the step1 ID to be preserved; for runtime applications which support it, this relationship will be preserved in the processed DITA output.

- The second `<step>` with the same key becomes `<step conref="componentA/configA.dita#configA/step1.5"><cmd/></step>`. However, conref relationships to step1.5 are not preserved, so this conref should be resolved into static content.

- For `<step>` three, the map instructs that both the key commonconfig and the ID step2 should be preserved in any content generated for this DITA topic. For formats that support runtime resolution through keys, a process must convert the `@conkeyref` value into an equivalent value for that format.

- Although resolution for the key used in `<step>` four is delayed, the specific element that is referenced should not be delayed. Thus the fourth step becomes `<step conref="componentA/configA.dita#configA/step2.5"><cmd/></step>`. This value is then processed as an ordinary `@conref` value.

This allows the information assembler to resolve references that must be to componentA while deferring references that can be fulfilled by alternative component content.

Note: This example demonstrates why the `<anchorid>` element cannot reference an element with the usual topicid/elementid format. If the two `<anchorid>` elements in the example had been set to config/step1 and config/step2, then they would only ever apply in a topic with id="config". It would not be possible to redirect the key to another topic, but still preserve conref behaviors as desired.

Note: Although it is not specifically called out in this example, it is possible to delay `@conref` resolution for an entire topic using the key. If `@conkeyref` on a task topic element is set to "componentAconfig", which *is not* delayed, the `@conref` will be evaluated as usual. However, if `@conkeyref` on the task is set to "commonconfig", which *is* delayed, resolution of `@conref` on that element should be delayed by a processor.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, and the attribute defined below.

@keyref (REQUIRED)

Defines a key that, when possible, is preserved in content generated from the DITA source material. Conref relationships that use this key are not resolved when generating that material, so that `@conref` might be resolved at run-time when an end user is reading the content.

3.5 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.

3.5.1 Hazard statement elements

The hazard statement domain elements are used to provide information about product safety hazards. The domain can be included in any topic type or map. Its elements are used to inform readers about potential hazards, consequences, and avoidance strategies.

3.5.1.1 <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 type of hazard, add information about the specific hazard and how to avoid it, and add one or more safety symbols.

Content models

See *appendix (424)* for information about this element in OASIS document type shells.

Inheritance

+ topic/note hazard-d/hazardstatement

Example

Danger: Indicates an imminently hazardous situation which, if not avoided, **will** result in death or serious injury.

```
<hazardstatement type="danger">
  <messagepanel>
    <typeofhazard>Rotating blade.</typeofhazard>
    <consequence>Moving parts can crush and cut.</consequence>
    <howtoavoid>Follow lockout procedure before servicing.</howtoavoid>
  </messagepanel>
  <hazardsymbol href="rotatingblade.png"/>
</hazardstatement>
```

Warning: Indicates a potentially hazardous situation which, if not avoided, **could** result in death or serious injury.

```
<hazardstatement type="warning">
  <messagepanel>
    <typeofhazard>Hot surfaces inside.</typeofhazard>
    <consequence>Contact might cause burn.</consequence>
    <howtoavoid>Wear protective gear before servicing internal parts.</howtoavoid>
  </messagepanel>
  <hazardsymbol href="hotsurface.png"/>
</hazardstatement>
```

Caution: Indicates a potentially hazardous situation which, if not avoided, **might** result in minor or moderate injury.

```
<hazardstatement type="caution">
  <messagepanel>
    <typeofhazard>Lifting Hazard.</typeofhazard>
    <consequence>Might result in injury.</consequence>
    <howtoavoid>See safety manual for lifting instructions.</howtoavoid>
  </messagepanel>
  <hazardsymbol href="heavy.png"/>
</hazardstatement>
```

Notice: Indicates a potential situation which, if not avoided, might result in **property damage** or in an **undesirable result or state**.

```
<hazardstatement type="notice">
  <messagepanel>
    <typeofhazard>Battery low</typeofhazard>
    <howtoavoid>Push and hold for charge state test.</howtoavoid>
  </messagepanel>
  <hazardsymbol href="general.png"/>
</hazardstatement>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, *spectitle*, and the attributes defined below.

@type

Describes the level of hazard. Safety hazard level definitions correspond to the same values in the ANSI Z535 and the ISO 3864 standards. Note that this differs from the `@type` attribute on many other DITA elements. See *The type attribute (370)* for detailed information on supported values and processing implications. Available values are note, tip, fastpath, restriction, important, remember, attention, caution, notice, danger, warning, other, and *-dita-use-conref-target (362)*.

@othertype

Indicates an alternate note type, when the type is not available in the `@type` attribute value list. This value is used as the user-provided note title when the `@type` attribute value is set to "other.".

3.5.1.2 <consequence>

The `<consequence>` element specifies the consequence of failing to avoid a hazard, for example, "Contact may cause burn."

Content models

See *appendix (414)* for information about this element in OASIS document type shells.

Inheritance

+ topic/li hazard-d/consequence

Example

```
<hazardstatement type="warning">
  <messagepanel>
    <typeofhazard>Hot surfaces inside.</typeofhazard>
    <consequence>Contact may cause burn.</consequence>
    <howtoavoid>Wear protective gear before servicing internal parts.</howtoavoid>
  </messagepanel>
  <hazardsymbol href="hotsurface.png"></hazardsymbol>
</hazardstatement>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.5.1.3 <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.

Content models

See *appendix (424)* for information about this element in OASIS document type shells.

Inheritance

+ topic/image hazard-d/hazardsymbol

Example

```
<hazardstatement type="danger">
  <messagepanel>
    <typeofhazard>Rotating blade.</typeofhazard>
    <consequence>Moving parts can crush and cut.</consequence>
    <howtoavoid>Follow lockout procedure before servicing.</howtoavoid>
  </messagepanel>
  <hazardsymbol href="rotatingblade.png"/>
</hazardstatement>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, *@keyref (360)*, and the attributes defined below.

@href

Provides a reference to the image. See *The href attribute (359)* for detailed information on supported values and processing implications.

@scope

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 (362)*. See *The scope attribute (373)* for more information on values.

@height

Indicates the vertical dimension for the resulting image display. If necessary, the image *SHOULD* be scaled to the specified size. 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". If a height value is specified and no width value is specified, the width *SHOULD* be scaled by the same factor as the height. 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.

@width

Indicates the horizontal dimension for the resulting image display. If necessary, the image *SHOULD* be scaled to the specified size. 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". If a width value is specified and no height value is specified, the height *SHOULD* be scaled 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.

@align

Controls 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 this image's `@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

Allow an image to be scaled up or down to fit within available space. Allowable values are yes, no, and *-dita-use-conref-target (362)*. If, for a given image, 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 (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

Indicates whether an image should be displayed inline or separated from the surrounding text. The default is inline. Allowable values are: inline, break, or and *-dita-use-conref-target (362)*.

@longdescref (DEPRECATED)

A reference to a textual description of the graphic or object. This attribute supports creating accessible content. See *The href attribute (359)* for detailed information on supported values and processing implications. For examples of how this attribute is used in output, see this topic on *long descriptions*. NOTE: This attribute is deprecated in favor of the *longdescref (206)* subelement to this element.

3.5.1.4 <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."

Content models

See *appendix (424)* for information about this element in OASIS document type shells.

Inheritance

+ topic/li hazard-d/howtoavoid

Example

```
<hazardstatement type="notice">
  <messagepanel>
    <typeofhazard>Machinery Damage</typeofhazard>
    <howtoavoid>
        <sl>
          <sli>Do NOT use solvents to clean the drum surface</sli>
          <sli>Read manual for proper drum cleaning procedure</sli>
        </sl>
    </howtoavoid>
  </messagepanel>
  <hazardsymbol href="readmanual.png"></hazardsymbol>
  <hazardsymbol href="agressivesolvent.png"></hazardsymbol>
</hazardstatement>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.5.1.5 <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.

Content models

See *appendix (432)* for information about this element in OASIS document type shells.

Inheritance

+ topic/ul hazard-d/messagepanel

Example

```
<hazardstatement type="caution">
  <messagepanel>
    <typeofhazard>Lifting Hazard.</typeofhazard>
    <consequence>May result in injury.</consequence>
    <howtoavoid>See safety manual for lifting instructions.</howtoavoid>
  </messagepanel>
  <hazardsymbol href="heavy.png"></hazardsymbol>
</hazardstatement>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *compact*, *outputclass*, and *spectitle*.

3.5.1.6 <typeofhazard>

The `<typeofhazard>` element contains a description of the type of hazard, for example, "Hot surfaces inside."

Content models

See *appendix (453)* for information about this element in OASIS document type shells.

Inheritance

+ topic/li hazard-d/typeofhazard

Example

```
<hazardstatement type="caution">
  <messagepanel>
    <typeofhazard>Lifting Hazard.</typeofhazard>
    <consequence>May result in injury.</consequence>
    <howtoavoid>See safety manual for lifting instructions.</howtoavoid>
  </messagepanel>
  <hazardsymbol href="heavy.png"></hazardsymbol>
</hazardstatement>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.5.2 Highlighting elements

The highlighting elements are used to highlight text with styles (such as bold, italic, and monospace). Never use these elements when a semantically specific element is available. These elements are not intended for use by specializers, and are intended solely for use by authors when no semantically appropriate element is available and a formatting effect is required.

3.5.2.1 <b>

The `<b>` element is typically used to apply bold highlighting to the content of the element. This element is part of the highlighting domain. Use this element only when a more semantically appropriate element is not available. For example, for specific items such as GUI controls, use the `<uicontrol>` element.

Note: Style sheets can be used to format the `<b>` element differently, just like any other element can be restyled. Thus, it is not the case that content in `<b>` elements will necessarily be bold.

Content models

See *appendix (411)* for information about this element in OASIS document type shells.

Inheritance

+ topic/ph hi-d/b

Example

```
<p><b>STOP!</b> This is <b>very</b> important!</p>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.5.2.2 <i>

The `<i>` element is typically used to apply italic highlighting to the content of the element. This element is part of the highlighting domain. Use this element only when a more semantically appropriate element is not available. For example, for specific items such as citations, use the `<cite>` element.

Note: Style sheets can be used to format the `<i>` element differently, just like any other element can be restyled. Thus, it is not the case that content in `<i>` elements will necessarily be italicized.

Content models

See *appendix (425)* for information about this element in OASIS document type shells.

Inheritance

+ topic/ph hi-d/i

Example

```
<p>Unplug the unit <i>before</i> placing the metal screwdriver
against the terminal screw.</p>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.5.2.3 <sup>

The `<sup>` element indicates that text should appear with superscript highlighting, or vertically raised in relationship to the surrounding text. Superscripts are usually a smaller font than the surrounding text. This element is part of the highlighting domain. Use this element only when a more semantically appropriate element is not available.

Note: Style sheets can be used to format the `<sup>` element differently, just like any other element can be restyled. Thus, it is not the case that content in `<sup>` elements will necessarily be rendered as superscript.

Content models

See *appendix (447)* for information about this element in OASIS document type shells.

Inheritance

+ topic/ph hi-d/sup

Example

```
<p>The power produced by the electrohydraulic dam was 10<sup>10</sup> more than
the older electric plant.  The difference was H<sub>2</sub>O.</p>
```

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)* and *outputclass*.

3.5.2.4 <sub>

The `<sub>` element indicates that text should appear with subscript highlighting, or placed lower in relationship to the surrounding text. Subscripted text is often a smaller font than the surrounding text. Formatting might vary depending on your output process. This element is part of the highlighting domain. Use this element only when a more semantically appropriate element is not available.

Note: Style sheets can be used to format the `<sub>` element differently, just like any other element can be restyled. Thus, it is not the case that content in `<sub>` elements will necessarily be rendered as subscript.

Content models

See *appendix* *(445)* for information about this element in OASIS document type shells.

Inheritance

+ topic/ph hi-d/sub

Example

```
<p>The power produced by the electrohydraulic dam was 10<sup>10</sup> more than
the older electric plant.  The difference was H<sub>2</sub>O.</p>
```

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)* and *outputclass*.

3.5.2.5 <tt>

The `<tt>` (teletype) element is typically used to apply monospaced highlighting to the content of the element. This element is part of the highlighting domain. Use this element only when a more semantically appropriate element is not available. For example, for specific items such as inline code fragments, use the `<codeph>` element.

Note: Style sheets can be used to format the `<tt>` element differently, just like any other element can be restyled. Thus, it is not the case that content in `<tt>` elements will necessarily be rendered as monospaced.

Content models

See *appendix* *(452)* for information about this element in OASIS document type shells.

Inheritance

+ topic/ph hi-d/tt

Example

```
<p>Make sure that the screen displays <tt>File successfully created</tt> before
proceeding to the next stage of the task.</p>
```

While the previous example demonstrates a potential use of `<tt>`, it might be more properly encoded using `<systemoutput>`.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.5.2.6 <u>

The `<u>` element is typically used to apply underline highlighting to the content of the element. This element is part of the highlighting domain. Use this element only when a more semantically appropriate element is not available. For example, for specific items such as GUI controls, use the `<uicontrol>` element.

Note: Style sheets can be used to format the `<u>` element differently, just like any other element can be restyled. Thus, it is not the case that content in `<u>` elements will necessarily be rendered as underlined.

Content models

See *appendix (453)* for information about this element in OASIS document type shells.

Inheritance

+ topic/ph hi-d/u

Example

```
<p>Beware: <u>overuse</u> <i>of</i> <b>highlighting</b> is
sometimes known as font-itis!</p>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.5.2.7 <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. This element is part of the highlighting domain. Use this element only when a more semantically appropriate element is not available.

Note: Style sheets can be used to format the `<line-through>` element differently, just like any other element can be restyled. Thus, it is not the case that content in `<line-through>` elements will necessarily be rendered with a line struck through.

Content models

See *appendix (429)* for information about this element in OASIS document type shells.

Inheritance

+ topic/ph hi-d/line-through

Example

```
<p>Line-through: DITA technology can be
    <line-through>maddening</line-through>a challenge to implement.</p>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.5.2.8 <overline>

The `<overline>` element indicates content that is rendered with a line above it. This element is part of the highlighting domain. Use this element only when a more semantically appropriate element is not available.

Note: Style sheets can be used to format the `<overline>` element differently, just like any other element can be restyled. Thus, it is not the case that content in `<overline>` elements will necessarily be rendered with a line above.

Content models

See *appendix (435)* for information about this element in OASIS document type shells.

Inheritance

+ topic/ph hi-d/overline

Example

```
<p>Overline: <overline><i>x</i></overline> is the
average value of<i>x<sub>i</sub></i></p>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.5.3 Utilities elements

The utilities domain elements represent common features of a language that might not necessarily be semantic, such as image maps.

3.5.3.1 <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.

Content models

See *appendix (410)* for information about this element in OASIS document type shells.

Inheritance

+ topic/figgroup ut-d/area

Example

```
<area>
 <shape>rect</shape>
 <coords>54,1,117,60</coords>
 <xref href="d1-s2.dita"></xref>
</area>
```

A more complete example is located in the description for *imagemap (298)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.5.3.2 <coords>

The `<coords>` element specifies the coordinates of a linkable region in an `<imagemap>`.

This element contains text data representing coordinate data for image maps. 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.

Content models

See *appendix (414)* for information about this element in OASIS document type shells.

Inheritance

+ topic/ph ut-d/coords

Example

```
<area>
 <shape>rect</shape>
 <coords>54,1,117,60</coords>
 <xref href="d1-s2.dita"></xref>
</area>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (with a narrowed definition of `@translate`, given below), *outputclass*, and *@keyref (360)*.

@translate

Indicates whether the content of the element should be translated or not. For this element the default value is "no". Setting to "yes" will override the default. 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. Available values are:

no

The content of this element is not translateable.

yes

The content of this element is translateable.

-dita-use-conref-target

See *Using the -dita-use-conref-target value (362)* for more information.

3.5.3.3 <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.

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.

An `<imagemap>` structure can be output either to a standard HTML image map or to alternative forms of navigation (such as table-based image maps). When output as PDF, the minimal form would be to represent at least the image; advanced PDF output processors should provide equivalent region-oriented hyperlinks.

The `<xref>` content within `<area>` contains the intended alternative text or hover text for that image map area.

Content models

See *appendix (425)* for information about this element in OASIS document type shells.

Inheritance

+ topic/fig ut-d/imagemap

Example

A simple `<imagemap>` looks like this (note that the rendering will depend on how this markup is supported for particular output formats):

```
<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:
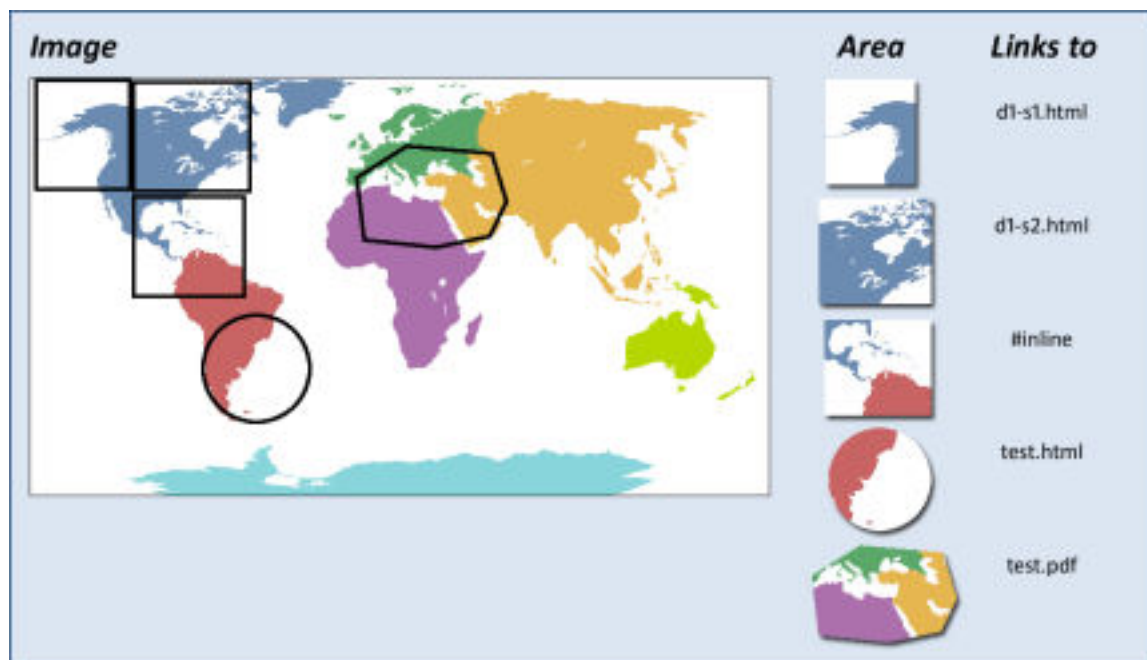
The values for use in the `<shape>` and `<coords>` elements follow the guidelines defined for image maps in HTML 4.1, *Client-side image maps: the MAP and AREA elements*

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Display attribute group (355)*, *outputclass*, and *spectitle*.

3.5.3.4 <shape>

The `<shape>` element defines the shape of a linkable area in an `<imagemap>`.

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.

Content models

See *appendix (442)* for information about this element in OASIS document type shells.

Inheritance

+ topic/keyword ut-d/shape

Example

```
<area>
 <shape>rect</shape>
 <coords>54,1,117,60</coords>
 <xref href="d1-s2.dita"></xref>
</area>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (with a narrowed definition of `@translate`, given below), *outputclass*, and *@keyref (360)*.

@translate
  Indicates whether the content of the element should be translated or not. For this element the default value is "no". Setting to "yes" will override the default. 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. Available values are:

no
    The content of this element is not translateable.

yes
    The content of this element is translateable.

-dita-use-conref-target
    See *Using the -dita-use-conref-target value (362)* for more information.

3.5.3.5 <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. The 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 is useful for elements where the base sort phrase is inadequate or non-existent, for example, a glossary entry for a Japanese Kanji phrase.

Content models

See *appendix (444)* for information about this element in OASIS document type shells.

The `<sort-as>` element can contain `<text>` and `<keyword>` elements in order to enable content referencing. 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.

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 *Sorting (122)* for more information on sorting.

Some elements in the base DITA vocabulary are natural candidates for sorting, including topics, definition list 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>`, `<searchtitle>`, or `<navtitle>` 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 topic prolog anywhere `<data>` is allowed.

- For glossary entry topics, the `<sort-as>` element can be included directly in `<glossterm>` or as a direct child of `<prolog>`.

- For topic references, the `<sort-as>` element can be included directly in the `<navtitle>` or `<title>` element within `<topicmeta>` or as a child of `<topicmeta>`.

- For definition list items, include the `<sort-as>` element in the `<dt>` element.

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.

- When located within the `<indexterm>` element, the `<sort-as>` element is equivalent to `<index-sort-as>`. It is an error for an `<indexterm>` element to directly contain both `<sort-as>` and `<index-sort-as>` elements.

- 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".

Inheritance

+ topic/data ut-d/sort-as

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):

```
<glossentry id="gloss-dada">
  <glossterm><sort-as value="dada"/>&#x5927;&#x5927;</glossterm>
    <glossdef>Literally "big big".</glossdef>
</glossentry>
```

Figure 78: The <sort-as> element located within <glossterm>

```
<glossentry id="gloss-dada">
  <glossterm>&#x5927;&#x5927;</glossterm>
  <prolog>
    <sort-as>dada</sort-as>
  </prolog>
    <glossdef>Literally "big big".</glossdef>
</glossentry>
```

Figure 79: The <sort-as> element within <prolog>

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)* and the attributes defined below.

@name

Names 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

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.

Related Links

2.4.8 Sorting *(122)*

Processors can be configured to sort elements. Typical processing includes sorting glossary entries, lists of parameters or reference entries in custom navigation structures, and tables based on the contents of cells in specific columns or rows.

3.5.4 DITAVAL reference domain

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.

3.5.4.1 <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 (303)* 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.

Content models

See *appendix (418)* for information about this element in OASIS document type shells.

Inheritance

+ map/topicref ditavalref-d/ditavalref

Example

See *Examples of branch filtering (105)* for several examples of the `<ditavalref>` element.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (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 *The href attribute (359)* 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 *The format attribute (372)* 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.

3.5.4.2 <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 author and navigation title, that might be useful for map architects but is not rendered in the output.

Content models

See *appendix (417)* for information about this element in OASIS document type shells.

Inheritance

+ map/topicmeta ditavalref-d/ditavalmeta

Example

See *ditavalref (302)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (except for `@conkeyref`, which is removed for all elements in this domain).

3.5.4.3 <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 group (358)*.

Content models

See *appendix (420)* for information about this element in OASIS document type shells.

Inheritance

+ topic/data ditavalref-d/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`.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (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".

3.5.4.4 <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 group (358)*, with one exception. Where `@copy-to` and `<dvrResourcePrefix>` might include path information, path information is not valid in `<dvrResourceSuffix>`.

Content models

See *appendix (420)* for information about this element in OASIS document type shells.

Inheritance

+ topic/data ditavalref-d/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`.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (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".

3.5.4.5 <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.

Content models

See *appendix (420)* for information about this element in OASIS document type shells.

Inheritance

+ topic/data ditavalref-d/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".

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (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".

3.5.4.6 <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.

Content models

See *appendix (420)* for information about this element in OASIS document type shells.

Inheritance

+ topic/data ditavalref-d/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".

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)* (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".

3.6 Classification elements

Classification elements support managing metadata. Those in the subject scheme map are used to define controlled values and to bind the controlled values to DITA attributes as enumerations. Those declared in the classification domain are used in other maps to classify content according to the scheme.

3.6.1 Subject scheme elements

Subject scheme maps are used to define sets of controlled values and taxonomic subjects; bind controlled values to attributes; and specify relationships among taxonomic subjects.

3.6.1.1 <subjectScheme>

The `<subjectScheme>` element is a specialization of `<map>`; it defines taxonomic subjects and controlled values.

Content models

See *appendix* *(446)* for information about this element in OASIS document type shells.

Inheritance

- map/map subjectScheme/subjectScheme

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="os">
    <subjectdef keys="linux"/>
    <subjectdef keys="mswin"/>
    <subjectdef keys="zos"/>
  </subjectdef>
  <!-- Define application values -->
```

```
  <subjectdef keys="app" navtitle="Applications">
    <subjectdef keys="apacheserv" href="subject/apache.dita"/>
    <subjectdef keys="mysql"      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, mswin, 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:
       apacheserv, mysql -->
  <enumerationdef>
    <attributedef name="otherprops"/>
    <subjectdef keyref="app"/>
  </enumerationdef>
</subjectScheme>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (with a narrowed definition of `@id`, given below), *Attributes common to many map elements (351)* (with narrowed definitions of `@processing-role` and `@toc`, given below), *Architectural attribute group (350)*, *outputclass*, and the attributes defined below. This element also uses `@type`, `@scope`, and `@format` from *Link relationship attribute group (357)*.

@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.

@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.

@processing-role

For this element, the default value for `@processing-role` is "resource-only". Otherwise, the definition matches the one found in *Attributes common to many map elements (351)*.

@toc

For this element, the default value for `@toc` is "no". Otherwise, the definition matches the one found in *Attributes common to many map elements (351)*.

3.6.1.2 <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.

Content models

See *appendix (441)* for information about this element in OASIS document type shells.

Inheritance

- map/topicref subjectScheme/schemeref

See *subjectScheme (308)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with narrowed definitions of `@href`, `@format`, and `@type`, given below), `@navtitle` and `@query` from *Topicref element attributes group (358)*, *outputclass*, *@keys (360)*, and *@keyref (360)*.

@href

A pointer to the resource represented by the `<topicref>`. See *The href attribute (359)* 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.

@format

On this element the `@format` attribute sets a default value of "ditamap", because the purpose of the element is to reference a ditamap document. Otherwise, the attribute is the same as described in *Link relationship attribute group (357)*.

@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.

3.6.1.3 <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.

Content models

See *appendix (423)* for information about this element in OASIS document type shells.

Inheritance

- map/topicref subjectScheme/hasInstance

Example

This example specifies that New York City, Reykjavik, and Moscow are each specific instances of a city.

```
<subjectScheme>
  <hasInstance>
    <subjectdef keys="city" navtitle="City">
      <subjectdef keys="nyc"       navtitle="New York City"/>
      <subjectdef keys="reykjavik" navtitle="Reykjavik"/>
      <subjectdef keys="moscow"    navtitle="Moscow"/>
    </subjectdef>
  </hasInstance>
</subjectScheme>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with a narrowed definition of `@href`, given below), `@processing-role` from *Attributes common to many map elements (351)*, `@navtitle` from *Topicref element attributes group (358)*, *outputclass*, *@keys (360)*, and *@keyref (360)*.

@href

A pointer to the resource represented by the `<topicref>`. See *The href attribute (359)* 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.

3.6.1.4 <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.

Content models

See *appendix (423)* for information about this element in OASIS document type shells.

Inheritance

- map/topicref subjectScheme/hasKind

Example

This example specifies that cities, towns, and villages are each a kind of settlement. Additionally, bigcity, mediumcity, and smallcity are each a kind of city.

```
<subjectScheme>
  <hasKind>
    <subjectdef keys="settlement" navtitle="Human settlement">
      <subjectdef keys="city" navtitle="City">
        <subjectdef keys="bigcity" navtitle="Big city"/>
        <subjectdef keys="mediumcity" navtitle="Medium city"/>
        <subjectdef keys="smallcity" navtitle="Small city"/>
      </subjectdef>
      <subjectdef keys="town" navtitle="Town"/>
      <subjectdef keys="village" navtitle="Village"/>
    </subjectdef>
  </hasKind>
</subjectScheme>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with a narrowed definition of `@href`, given below), `@processing-role` from *Attributes common to many map elements (351)*, `@navtitle` from *Topicref element attributes group (358)*, *outputclass*, *@keys (360)*, and *@keyref (360)*.

@href

A pointer to the resource represented by the `<topicref>`. See *The href attribute (359)* 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.

3.6.1.5 <hasNarrower>

For subjects within the `<hasNarrower>` element, the container subject is more general than each of the contained subjects. That is, this element makes the default hierarchical relationship explicit, although the way in which a relationship is narrower is not specified.

Content models

See *appendix (424)* for information about this element in OASIS document type shells.

Inheritance

- map/topicref subjectScheme/hasNarrower

Example

This example specifies that Planting Roses is a narrower subject category than Horticulture, although it is part of the Horticulture subject area.

```
<subjectScheme>
  <hasNarrower>
    <subjectdef keys="horticulture" navtitle="Horticulture">
      <subjectdef keys="plantrose"  navtitle="Planting Roses"/>
    </subjectdef>
  </hasNarrower>
</subjectScheme>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with a narrowed definition of `@href`, given below), `@processing-role` from *Attributes common to many map elements (351)*, `@navtitle` from *Topicref element attributes group (358)*, *outputclass*, *@keys (360)*, and *@keyref (360)*.

@href

A pointer to the resource represented by the `<topicref>`. See *The href attribute (359)* 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.

3.6.1.6 <hasPart>

The `<hasPart>` element specifies that the contained hierarchy expresses PART-OF relationships between subjects.

Content models

See *appendix (424)* for information about this element in OASIS document type shells.

Inheritance

- map/topicref subjectScheme/hasPart

Example

This example specifies that a tire and a horn are each a part of a car.

```
<subjectScheme>
  <hasPart>
    <subjectdef keys="car" navtitle="Car">
      <subjectdef keys="tire" navtitle="Tire"/>
      <subjectdef keys="horn" navtitle="Horn"/>
    </subjectdef>
  </hasPart>
</subjectScheme>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with a narrowed definition of `@href`, given below), `@processing-role` from *Attributes common to many map elements (351)*, `@navtitle` from *Topicref element attributes group (358)*, *outputclass*, *@keys (360)*, and *@keyref (360)*.

@href

A pointer to the resource represented by the `<topicref>`. See *The href attribute (359)* 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.

3.6.1.7 <hasRelated>

The `<hasRelated>` element identifies an associative relationship between the container subject and each of the contained subjects.

Content models

See *appendix (424)* for information about this element in OASIS document type shells.

Inheritance

- map/topicref subjectScheme/hasRelated

Example

This example specifies that myProgram runs on Linux and Windows.

```
<subjectScheme>
  <subjectdef keys="myProgram" navtitle="My Program">
    <hasRelated keys="runsOn" navtitle="runs on">
      <subjectdef keys="linux" navtitle="Linux"/>
      <subjectdef keys="mswin" navtitle="Microsoft Windows"/>
    </hasRelated>
  </subjectdef>
</subjectScheme>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with a narrowed definition of `@href`, given below), processing-role from *Attributes common to many map elements (351)*, navtitle from *Topicref element attributes group (358)*, *outputclass*, *@keys (360)*, and *@keyref (360)*.

@href

A pointer to the resource represented by the `<topicref>`. See *The href attribute (359)* 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.

3.6.1.8 <enumerationdef>

The `<enumerationdef>` element identifies one attribute and one or more categories that contain the controlled values for the enumeration. The `@type` attribute has a default value of "keys".

When the `<enumerationdef>` element contains both an `<attributedef>` 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="value"/>` and `<elementdef name="lomDifficulty"/>`, only the `@value` attribute on the `<lomDifficulty>` element is limited to the specified enumeration. The possible values for the `@value` attribute on other elements are not affected.

When the `<enumerationdef>` element contain an `<attributedef>` element but no `<elementdef>` element, the controlled set of 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.

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.

Content models

See *appendix (421)* for information about this element in OASIS document type shells.

Inheritance

- map/topicref subjectScheme/enumerationdef

Example

In this example, enumerations are specified for the `@platform` and `@otherprops` attributes. Note that the enumeration identifies a category of values; the values within the category are valid, while the category itself is not a valid value. For example, in the code sample here, the `@platform` attribute is associated with the enumeration for the category "os"; all values within the "os" category are thus valid on the `@platform` attribute, while the value "os" itself is not.

```
<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="os">
    <subjectdef keys="linux"/>
    <subjectdef keys="mswin"/>
    <subjectdef keys="zos"/>
  </subjectdef>
  <!-- Define application values -->
  <subjectdef keys="app" navtitle="Applications">
    <subjectdef keys="apacheserv" href="subject/apache.dita"/>
    <subjectdef keys="mysql"      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, mswin, 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:
       apacheserv, mysql -->
  <enumerationdef>
    <attributedef name="otherprops"/>
    <subjectdef keyref="app"/>
  </enumerationdef>
</subjectScheme>
```

Attributes

The following attributes are available on this element: *ID attribute group (347)*, `@status` and `@base` from *Metadata attribute group (348)*, *outputclass*, *Debug attribute group (350)*, and *class (Not for use by authors)*.

3.6.1.9 <elementdef>

The `<elementdef>` element identifies an element to which an attribute is bound. If the `<enumerationdef>` element does not contain an `<elementdef>` element, the enumeration is bound to the attribute in all elements.

Content models

See *appendix (421)* for information about this element in OASIS document type shells.

Inheritance

- topic/data subjectScheme/elementdef

Example

In this example, the `@value` attribute for the `<lomDifficulty>` element is bound to a specified set of values. Processors should limit the values for the `@value` attribute on the `<lomDifficulty>` element to the following set of values: easy, medium, or difficult. Other elements that have a `@value` attribute are not affected.

```
<subjectScheme>
  <subjectdef keys="difficulty">
    <subjectdef keys="easy"/>
    <subjectdef keys="medium"/>
    <subjectdef keys="difficult"/>
  </subjectdef>
  <!-- ... -->
  <enumerationdef>
    <elementdef name="lomDifficulty"/>
    <attributedef name="value"/>
    <subjectdef keyref="difficulty"/>
  </enumerationdef>
</subjectScheme>
```

Attributes

The following attributes are available on this element: *ID attribute group (347)*, `@status` and `@base` from *Metadata attribute group (348)*, *outputclass*, *Debug attribute group (350)*, *class (Not for use by authors)*, and the attributes defined below.

@name (REQUIRED)

  Defines the element for which an attribute enumeration is defined.

@translate

  Indicates whether the content of the element should be translated or not. For this element the default value is "no". Setting to "yes" will override the default. 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. Available values are:

no

  The content of this element is not translateable.

yes

  The content of this element is translateable.

-dita-use-conref-target

  See *Using the -dita-use-conref-target value (362)* for more information.

3.6.1.10 <attributedef>

The `<attributedef>` element specifies the attribute that is bound to a set of controlled values. This binding restricts the permissible values for the attribute to those that are contained in the set of controlled values.

Content models

See *appendix (410)* for information about this element in OASIS document type shells.

Inheritance

- topic/data subjectScheme/attributedef

Example

In this example, enumerations are specified for the `@platform` and `@otherprops` attributes. Note that the enumeration identifies a category of values; the values within the category are valid, while the category itself is not a valid value. For example, in the code sample here, the `@platform` attribute is associated with the enumeration for the category "os"; all values within the "os" category are thus valid on the `@platform` attribute, while the value "os" itself is not.

```
<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="os">
    <subjectdef keys="linux"/>
    <subjectdef keys="mswin"/>
    <subjectdef keys="zos"/>
  </subjectdef>
  <!-- Define application values -->
  <subjectdef keys="app" navtitle="Applications">
    <subjectdef keys="apacheserv" href="subject/apache.dita"/>
    <subjectdef keys="mysql"      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, mswin, 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:
       apacheserv, mysql -->
  <enumerationdef>
    <attributedef name="otherprops"/>
    <subjectdef keyref="app"/>
  </enumerationdef>
</subjectScheme>
```

Attributes

The following attributes are available on this element: *ID attribute group (347)*, `@status` and `@base` from *Metadata attribute group (348)*, *outputclass*, *Debug attribute group (350)*, *class (Not for use by authors)*, and the attributes defined below.

@name (REQUIRED)

Defines an attribute that will take a set of enumerated values.

@translate

Indicates whether the content of the element should be translated or not. For this element the default value is "no". Setting to "yes" will override the default. 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. Available values are:

no

The content of this element is not translateable.

yes

The content of this element is translateable.

-dita-use-conref-target

See *Using the -dita-use-conref-target value (362)* for more information.

3.6.1.11 <defaultSubject>

The `<defaultSubject>` element is used within an attribute enumeration to set the default value for the attribute in cases where no value is specified for the attribute. The default subject must be one of the controlled values that are bound to the attribute.

Content models

See *appendix (417)* for information about this element in OASIS document type shells.

Inheritance

- map/topicref subjectScheme/defaultSubject

Example

The following example declares that each of the four defined "os" values is valid within the `@platform` attribute; if no value is specified, 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>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with a narrowed definition of `@href`, given below), *Topicref element attributes group (358)*, *outputclass*, *@keys (360)*, and *@keyref (360)*. This element also uses `@processing-role`, `@locktitle`, and `@toc` from *Attributes common to many map elements (351)*.

@href

A pointer to the resource represented by the `<topicref>`. See *The href attribute (359)* 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.

3.6.1.12 <subjectHead>

The `<subjectHead>` element provides a heading for a group of subjects, for use if the scheme is displayed. For instance, a schememight be displayed 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.

Content models

See *appendix (446)* for information about this element in OASIS document type shells.

Inheritance

- map/topicref subjectScheme/subjectHead

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>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, `@navtitle` from *Topicref element attributes group (358)*, and *outputclass*. This element also uses `@processing-role`, `@toc`, and narrowed definitions of `@collection-type` and `@linking` from *Attributes common to many map elements (351)*.

@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 *Using the -dita-use-conref-target value (362)* 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.

3.6.1.13 <subjectHeadMeta>

The `<subjectHeadMeta>` element allows a navigation title and short description to be associated with a subject heading.

Content models

See *appendix (446)* for information about this element in OASIS document type shells.

Inheritance

- map/topicmeta subjectScheme/subjectHeadMeta

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>
```

Attributes

The following attributes are available on this element: *Universal attribute group* *(347)* and the attribute defined below.

@lockmeta

By default, metadata in the map supplements or overrides metadata that is specified at the topic level, unless the `@lockmeta` attribute of the `<topicmeta>` element is set to "no". Allowable values are "yes", "no", and *Using the -dita-use-conref-target value* *(362)*.

3.6.1.14 <subjectdef>

The `<subjectdef>` element defines a subject (also known as a controlled value) within a subject scheme. The `@keys` attribute specified on the `<subjectdef>` element assigns a key to the subject. A subject with a key can be addressed using a `@keyref` attribute.

The `<subjectdef>` also can use a `<navtitle>` element to supply a label for the subject or the `@href` attribute to reference a topic that captures the consensus definition for the subject.

As with normal `<topicref>` processing, when the `<subjectdef>` element specifies a `<navtitle>` and refers to a topic, processors should use the actual topic title in place of the `<navtitle>`. To use the `<navtitle>` as a subject label, set the `@locktitle` attribute on the `<subjectdef>` element to "yes".

Content models

See *appendix* *(446)* for information about this element in OASIS document type shells.

Inheritance

- map/topicref subjectScheme/subjectdef

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="os">
    <subjectdef keys="linux"/>
    <subjectdef keys="mswin"/>
    <subjectdef keys="zos"/>
  </subjectdef>
  <!-- Define application values -->
  <subjectdef keys="app" navtitle="Applications">
    <subjectdef keys="apacheserv" href="subject/apache.dita"/>
```

```
      <subjectdef keys="mysql"      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, mswin, 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:
         apacheserv, mysql -->
    <enumerationdef>
      <attributedef name="otherprops"/>
      <subjectdef keyref="app"/>
    </enumerationdef>
  </subjectScheme>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Topicref element attributes group (358)*, *Link relationship attribute group (357)* (with a narrowed definition of `@href`, given below), *outputclass*, *@keys (360)*, and *@keyref (360)*. This element also uses `@processing-role`, `@toc`, `@collection-type`, `@linking`, and `@locktitle` from *Attributes common to many map elements (351)*.

@href

A pointer to the resource represented by the `<topicref>`. See *The href attribute (359)* 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.

3.6.1.15 <relatedSubjects>

The `<relatedSubjects>` element establishes associative relationships between each child subject and every other child subject (unless the association is restricted by the `@linking` attribute of the subjects).

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.

The content provider can identify the relationship by specifying a `@keys` attribute, label the relationship by specifying a `<navtitle>` element or `@navtitle` attribute, and provide a consensus definition of the relationship including 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.

Content models

See *appendix (439)* for information about this element in OASIS document type shells.

Inheritance

- map/topicref subjectScheme/relatedSubjects

Example

The following scheme establishes that the Linux, the Apache Web Server, and the MySQL Database are related:

```
<subjectScheme>
  <!-- ... -->
  <relatedSubjects>
    <subjectdef keys="linux"     navtitle="Linux"/>
    <subjectdef keys="apacheweb" navtitle="Apache Web Server"/>
    <subjectdef keys="mysql"     navtitle="MySQL Database"/>
  </relatedSubjects>
```

```
  <!-- ... -->
</subjectScheme>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with a narrowed definition of `@href`, given below), `@navtitle` and `@query` from *Topicref element attributes group (358)*, *outputclass*, *@keys (360)*, and *@keyref (360)*. This element also uses `@processing-role`, `@collection-type`, and a narrowed definition of `@linking` (given below) from *Attributes common to many map elements (351)*.

@href

A pointer to the resource represented by the `<topicref>`. See *The href attribute (359)* 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.

@linking

On this element, the `@linking` attribute has a default value of "normal". Otherwise, the attribute is the same as defined in *Attributes common to many map elements (351)*.

3.6.1.16 <subjectRelTable>

The `<subjectRelTable>` element is a specialized relationship table which establishes relationships between the subjects in different columns of the same row. 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.

Content models

See *appendix (446)* for information about this element in OASIS document type shells.

Inheritance

- map/reltable subjectScheme/subjectRelTable

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 10: <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"/>`

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *Attributes common to many map elements (351)* (with a narrowed definition of `@toc`, given below). This element also uses `@type`, `@scope`, and `@format` from *Link relationship attribute group (357)*.

@toc

Specifies whether a topic appears in the table of contents (TOC). If the value is not specified locally, but is specified on an ancestor, the value will cascade from the closest ancestor. On this element the default value for `@toc` is "no". See *Attributes common to many map elements (351)* for a complete definition of `@toc`.

3.6.1.17 <subjectRelHeader>

The `<subjectRelHeader>` element specifies the roles played by subjects in associations.

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.

Content models

See *appendix (446)* for information about this element in OASIS document type shells.

Inheritance

- map/relrow subjectScheme/subjectRelHeader

Example

For a complete example in the context of the `<subjectRelTable>`, see *Example (321)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.6.1.18 <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.

Content models

See *appendix (446)* for information about this element in OASIS document type shells.

Inheritance

- map/relrow subjectScheme/subjectRel

Example

For a complete example in the context of the `<subjectRelTable>`, see *Example (321)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.6.1.19 <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>`. 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.

Content models

See *appendix (446)* for information about this element in OASIS document type shells.

Inheritance

- map/relcell subjectScheme/subjectRole

Example

For a complete example in the context of the `<subjectRelTable>`, see *Example (321)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *Topicref element attributes group (358)*. This element also uses `@scope`, `@format`, and `@type` from *Link relationship attribute group (357)*.

3.6.2 Classification domain elements

The classification domain elements are used to identify the subject matter of content that is referenced in a map. These subjects must be subjects defined in a subject scheme map. In particular, this allows an author to classify content in new subject categories that are not bound to existing metadata attributes.

3.6.2.1 <subjectref>

The `<subjectref>` element identifies a subject to classify content. The `<subjectref>` can identify the subject with a `@keyref` attribute (if the scheme has a `<subjectdef>` with a `@keys` attribute that assigns a key to the subject) or an `@href` attribute (if the scheme is not available and a topic exists that defines the subject).

Content models

See *appendix (446)* for information about this element in OASIS document type shells.

Inheritance

+ map/topicref classify-d/subjectref

Example

In the following example, the map is classified as covering the Linux subject and the "Developing web applications" topic 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="webapp.dita" navtitle="Developing web applications">
    <topicsubject>
      <subjectref keyref="web"/>
      <subjectref keyref="development"/>
    </topicsubject>
    <!-- ... -->
  </topicref>
  <!-- ... -->
</map>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with a narrowed definition of `@href`, given below), `@navtitle` and `@query` from *Topicref element attributes group (358)*, *outputclass*, *@keyref (360)*, and *@keys (360)*. This element also uses `@collection-type`, `@linking`, and narrowed definitions of `@processing-role` and `@toc` (given below), from *Attributes common to many map elements (351)*.

@href

A pointer to the resource represented by the `<topicref>`. See *The href attribute (359)* 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.

@processing-role

For this element, the default value for `@processing-role` is "resource-only". Otherwise, the definition matches the one found in *Attributes common to many map elements (351)*.

@toc

Specifies whether a topic appears in the table of contents (TOC). If the value is not specified locally, but is specified on an ancestor, the value will cascade from the closest ancestor. On this element the default value for `@toc` is "no". See *Attributes common to many map elements (351)* for a complete definition of `@toc`.

3.6.2.2 <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.

Content models

See *appendix (450)* for information about this element in OASIS document type shells.

Inheritance

+ map/topicref classify-d/topicapply

Example

The map content should be retrieved for Apache Tomcat and hidden as irrelevant for operating systems other than RedHat 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>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with a narrowed definition of `@href`, given below), `@navtitle` and `@query` from *Topicref element attributes group (358)*, *outputclass*, *@keyref (360)*, and *@keys (360)*. This element also uses `@collection-type`, `@linking`, and narrowed definitions of `@processing-role` and `@toc` (given below), from *Attributes common to many map elements (351)*.

@href

A pointer to the resource represented by the `<topicref>`. See *The href attribute (359)* 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.

@processing-role

For this element, the default value for `@processing-role` is "resource-only". Otherwise, the definition matches the one found in *Attributes common to many map elements (351)*.

@toc

Specifies whether a topic appears in the table of contents (TOC). If the value is not specified locally, but is specified on an ancestor, the value will cascade from the closest ancestor. On this element the default value for `@toc` is "no". See *Attributes common to many map elements (351)* for a complete definition of `@toc`.

3.6.2.3 <topicsubject>

The `<topicsubject>` element identifies the subjects covered by a topic or map.

In order to identify a primary subject, refer to the subject with the `<topicsubject>` itself. Subjects can be identified by `@keys` (if defined in the scheme) or, if the subject definition topic exists, by `@href` (as with ordinary topic references).

Additional secondary subjects can be specified by nested `<subjectref>` elements.

Content models

See *appendix (452)* for information about this element in OASIS document type shells.

Inheritance

+ map/topicref classify-d/topicsubject

Example

In the following example, the map is classified as covering Linux as the primary subject; the topic "Developing web applications" also covers the secondary 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="webapp.dita" navtitle="Developing web applications">
    <topicsubject>
      <subjectref keyref="web"/>
      <subjectref keyref="development"/>
    </topicsubject>
    <!-- ... -->
  </topicref>
  <!-- ... -->
</map>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)* (with a narrowed definition of `@href`, given below), `@navtitle` and `@query` from *Topicref element attributes group (358)*, *outputclass*, *@keyref (360)*, and *@keys (360)*. This element also uses narrowed definitions of `@processing-role` and `@toc` (given below) from *Attributes common to many map elements (351)*.

@href

A pointer to the resource represented by the `<topicref>`. See *The href attribute (359)* 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.

@processing-role

For this element, the default value for `@processing-role` is "resource-only". Otherwise, the definition matches the one found in *Attributes common to many map elements (351)*.

@toc

Specifies whether a topic appears in the table of contents (TOC). If the value is not specified locally, but is specified on an ancestor, the value will cascade from the closest ancestor. On this element the default value for `@toc` is "no". See *Attributes common to many map elements (351)* for a complete definition of `@toc`.

3.6.2.4 <topicSubjectTable>

The `<topicSubjectTable>` element is a specialized relationship table which allows a map to use relationship tables to associate topics with subjects. Tools (such as search tools) might use these classifications to retrieve content that is relative to a specific subject or combination of subjects.

In a `<topicSubjectTable>`, the first column is reserved for references to content. Subsequent columns are reserved for subjects that classify the content, each column supplying the subjects for a different category as 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.

Note: In a traditional `<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. Instead, relationships are defined between subjects using a subject scheme map (which provides a `<subjectRelTable>` element for non-hierarchical relationships between subjects).

Content models

See *appendix (452)* for information about this element in OASIS document type shells.

Inheritance

+ map/reltable classify-d/topicSubjectTable

Example

The topic subject table below classifies topics with goals for retrieval and with operating systems for filtering. The map makes use of definitions in a subject scheme map, defined separately.

```
<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 80: Subject scheme map

The following `<topicSubjectTable>` classifies several topics according to subjects 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 "Configuring cron for efficient startup" and "Allocating raw storage" are each classified by the goal of "performance"; in addition, they are classified by the operating systems "linux" and "unix".

- The topics "Analyzing web logs for service issues" and "Detecting denial-of-service attacks" are each classified by the goal of "reliability"; in addition, they 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="webServerStart.dita" navtitle="Configuring cron for efficient startup"/>
      <topicref href="dbDisk.dita" navtitle="Allocating raw storage"/>
    </topicCell>
    <subjectCell>
      <topicsubject keyref="performance"/>
    </subjectCell>
    <subjectCell>
      <topicapply keyref="linux"/>
      <topicapply keyref="unix"/>
    </subjectCell>
  </topicSubjectRow>
  <topicSubjectRow>
    <topicCell>
      <topicref href="webLogAnalyze.dita" navtitle="Analyzing web logs for service issues"/>
      <topicref href="webDenialService.dita" navtitle="Detecting denial-of-service attacks"/>
    </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 titles, and related subjects are displayed using only their `@keyref` attribute value.

task	*goal*	*os*
Configuring cron for efficient startup Allocating raw storage	performance	linux unix
Analyzing web logs for service issues Detecting denial-of-service attacks	reliability	linux unix windows

Figure 81: Topic subject table

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *Attributes common to many map elements (351)* (with a narrowed definition of `@toc`, given below). This element also uses `@type`, `@scope`, and `@format` from *Link relationship attribute group (357)*.

@toc

Specifies whether a topic appears in the table of contents (TOC). If the value is not specified locally, but is specified on an ancestor, the value will cascade from the closest ancestor. On this element the default value for `@toc` is "no". See *Attributes common to many map elements (351)* for a complete definition of `@toc`.

3.6.2.5 <topicSubjectHeader>

The `<topicSubjectHeader>` element specifies constraints on the subjects used in classifications.

Use the `<topicSubjectHeader>` element to supply a header row for a topic classification table when you want to encourage classification with subjects from different categories (also known as a 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.

Content models

See *appendix (452)* for information about this element in OASIS document type shells.

Inheritance

+ map/relrow classify-d/topicSubjectHeader

Example

For a complete example in the context of the `<topicSubjectTable>`, see *Example (327)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.6.2.6 <topicSubjectRow>

The `<topicSubjectRow>` is a grouping element that contains one row of a subject table. It contains topic references in the first column, and relates those references to the subjects in each following column.

Content models

See *appendix (452)* for information about this element in OASIS document type shells.

Inheritance

+ map/relrow classify-d/topicSubjectRow

Example

For a complete example in the context of the `<topicSubjectTable>`, see *Example (327)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)*.

3.6.2.7 <topicCell>

The `<topicCell>` element contains topics that will be associated with subjects in each following column of the current row in the `<topicSubjectTable>`.

Content models

See *appendix (450)* for information about this element in OASIS document type shells.

Inheritance

+ map/relcell classify-d/topicCell

Example

For a complete example in the context of the `<topicSubjectTable>`, see *Example (327)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *Attributes common to many map elements (351)*. This element also uses `@scope`, `@format`, and `@type` from *Link relationship attribute group (357)*.

3.6.2.8 <subjectCell>

The `<subjectCell>` element contains subjects that are associated with topics in the first column of the current row in the `<topicSubjectTable>`. The subjects themselves have no defined relationship across columns, other than the fact that they apply to the same content.

Content models

See *appendix (446)* for information about this element in OASIS document type shells.

Inheritance

+ map/relcell classify-d/subjectCell

Example

For a complete example in the context of the `<topicSubjectTable>`, see *Example (327)*.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *Attributes common to many map elements (351)*. This element also uses `@scope`, `@format`, and `@type` from *Link relationship attribute group (357)*.

3.7 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.

3.7.1 <boolean>

The `<boolean>` element was deprecated in DITA version 1.1. It was originally intended to express one of two opposite values, such as yes / no or on / off.

> **Note:** This element is deprecated. It is functionally equivalent to `<state value="yes|no"/>`, which is recommended as its replacement in all cases.

Content models

See *appendix (412)* for information about this element in OASIS document type shells.

Inheritance

- topic/boolean

Example

```
She said "<boolean state="yes"/>" when I asked her to marry me!
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, and the attribute defined below.

@state (REQUIRED)

  The state of the `<boolean>` element. Allowable values are "yes", "no", and "-dita-use-conref-target".

3.7.2 <data>

The `<data>` element represents a property within a DITA topic or map. While the `<data>` element can be used directly to capture properties, it is particularly useful as a basis for specialization. Default processing should treat the content as an unknown kind of metadata and ignore it for rendering, but custom processing might match the `@name` attribute or specialized element and use the element for automated manipulation or to format data associated with the body flow. For example, a specialized `<data>` element might be used to format properties as sidebars or other adornments or to harvest properties for automated processing.

The subject of the property is ordinarily the container of the `<data>` element. In the content model for the `<prolog>` and `<metadata>` elements, the property applies to the topic as a whole. In the `<topicmeta>` element, the property applies to the referenced topic. The `<data-about>` element can be used to identify the subject of the property with an explicit reference.

The `@name` attribute names the property for processes. A `<title>` subelement can provide a label for the property. The `@datatype` attribute can be used to identify the type for the value. The value of the property can be any of the following:

- A simple text value expressed with the `@value` attribute or textual content.

- A reference to either DITA content or a non-DITA resource expressed with the `@href` attribute.

- An image or other non-textual object.

- A brief unit of descriptive text that is not part of the body text flow.

- A complex structure composed of nested `<data>` elements.

> **Caution:** Processors should ignore the content of the `<data>` element by default, so the `<data>` element should only be used for properties and not to embed text for formatting as part of the flow of the topic body. It might be tempting to specialize the `<data>` element for text that is part of the body flow, so as to escape the restrictions of the base content models. This abuse of the DITA architecture will cause problems. For example, if a particular kind of paragraph is specialized from `<data>` rather than from `<p>`, then when the

content is exchanged with others that do not recognize the specialized element, their processors will skip the content.

The `<data>` element can be nested to create structures for complex properties. The `@name` attribute might be used to distinguish different semantics associated with different instances of the `<data>` element such as addresses, times, amounts, and so on. In many cases, however, it is preferable to specialize the `<data>` element for more precise semantics, structures, and values. For instance, a specialization can specify an enumeration for the `@value` attribute.

A `<data>` element containing properties of a topic as a whole should be located in the topic's `<prolog>` or `<metadata>` element, or in a `<topicmeta>` element related to a `<topicref>` that references the topic. The `<data>` element generally goes at the beginning of the element to which the properties in it refer. Where this is unwieldy, the `<data>` element can go in the `<prolog>`, with the `<data-about>` element identifying which specific element in the topic is the reference.

Content models

See *appendix (415)* for information about this element in OASIS document type shells.

Inheritance

- topic/data

Example

Uses of the `<data>` element can include the following:

- Complex metadata properties such as bibliographic records corresponding to citations.

- Hybrid documents with data values as part of the content, such as word processor formats using form fields.

- Messages in which the payload includes human-readable content. Such applications can use the `<data>` element to define the addressing on the message envelope. For instance, a topic could model an email message by representing the address with specialized `<data>` elements in the `<prolog>` element and the content with the `<body>` element.

- Transactional documents in which the values are processed but also displayed with human-readable content. In particular, a library of building blocks for transaction documents can be implemented through a DITA domain as specialized `<data>` elements including those from the UN/CEFACT Core Components Technical Specification (*http://www.unece.org/cefact/*).

This structure identifies the library and version demonstrated by a code sample. The `@name` attribute is used to identify both the grouping `<data>` element and the nested `<data>` elements that provide specific properties. These properties will not appear in the output unless a processor is customized to recognize these `@name` attribute values.

```
<codeblock>
  <data name="exampleOf">
    <data name="library" href="ajaxLibrary.js"/>
    <data name="version" value="2006-6-19"/>
  </data>
  <!-- ... -->
</codeblock>
```

Figure 82: Using the @name attribute on unspecialized <data> elements

The following example specifies the delimited source code for a code fragment so an automated process can refresh the code fragment. The `<fragmentSource>`, `<sourceFile>`, `<startDelimiter>`, and `<endDelimiter>` elements are specialized from `<data>` but the `<codeFragment>` is specialized from

`<codeblock>`. The properties specialized from `<data>` would not appear in formatted output unless a processor is explicitly instructed to display them:

```
<example>
    <title>An important coding technique</title>
    <codeFragment>
        <fragmentSource>
            <sourceFile     value="helloWorld.java"/>
            <startDelimiter value="FRAGMENT_START_1"/>
            <endDelimiter   value="FRAGMENT_END_1"/>
        </fragmentSource>
        <!-- ... -->
    </codeFragment>
</example>
```

Figure 83: Specializing <data> to annotate a code sample

The following example identifies a real estate property as part of a house description. The `<realEstateProperty>` element and its child elements are specialized from `<data>`. The `<houseDescription>` element is specialized from `<section>`. A specialized process can format the values as part of a brochure if they meet criteria for inclusion.

```
<houseDescription>
  <title>A great home for sale</title>
  <realEstateProperty>
    <realEstateBlock value="B7"/>
    <realEstateLot   value="4003"/>
    <!-- ... -->
  </realEstateProperty>
  <p>This elegant....</p>
  <object data="B7_4003_tour360Degrees.swf"/>
</houseDescription>
```

Figure 84: Specializing <data> to annotate housing information

Attributes

The following attributes are available on this element: *Data element attributes group (355)*, *Link relationship attribute group (357)*, *Universal attribute group (347)*, *@keyref (360)*, and *outputclass*.

3.7.3 <data-about>

The `<data-about>` element identifies the subject of a property when the subject isn't associated with the context in which the property is specified. The property itself is expressed by the `<data>` element. The `<data-about>` element handles exception cases where a property must be expressed somewhere other than inside the actual subject of the property. The `<data-about>` element is particularly useful as a basis for specialization in combination with the `<data>` element.

> **Important:** Do not use the `<data-about>` element to identify the object of a property. The `@href` attribute of the `<data>` element serves that purpose.

Content models

See *appendix (416)* for information about this element in OASIS document type shells.

Inheritance

- topic/data-about

Example

The full properties of a cited book can be maintained conveniently in the `<prolog>`:

```
<topic id="questions">
    <title>Questions and answers</title>
    <prolog>
        <data-about href="urn:isbn:0156983508" scope="external">
            <data name="title">The World Doesn't End</data>
            <data name="author">
                <data name="firstname">Charles</data>
                <data name="lastname">Simic</data>
            </data>
            <data name="published" datatype="year">1989</data>
            <!-- ... -->
        </data-about>
        <!-- ... -->
    </prolog>
    <body>
        <!-- ... -->
        <lq href="urn:isbn:0156983508">In a forest of question marks ...
        </lq>
        <!-- ... -->
    </body>
</topic>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *Link relationship attribute group (357)*, and *outputclass*

3.7.4 <foreign>

The `<foreign>` element allows the introduction of non-DITA content, for example, MathML, SVG, or Rich Text Format (RTF). 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 generally is implemented as a domain, but architects looking for more control over the content can implement foreign vocabularies as structural specializations.

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 `<div>` in HTML).

- If alternate content is desired, specialize the `<desc>` element to contain it. This specialization of `<desc>` should be used within the element specialized from `<foreign>`. Such alternate content must of course be valid wherever the `<foreign>` specialization is valid.

- 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.

Content models

See *appendix (423)* for information about this element in OASIS document type shells.

Inheritance

- topic/foreign

SVG Example within a <p> element

```
<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>
```

MathML Example within an <object> element

```
<p>... as in the formula
<object>
  <desc>4 + x</desc>
  <mathml>
    <m:math display="block">
      <m:mrow>
        <m:mo>sum</m:mo>
        <m:mn>4</m:mn>
        <m:mo>+</m:mo>
        <m:mi>x</m:mi>
      </m:mrow>
    </m:math>
  </mathml>
</object>.
</p>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.7.5 <index-base>

The <index-base> element allows indexing extensions to be added by specializing this element.

The <index-base> element can only exist as a child of an <indexterm> element. This characteristic makes it the appropriate element to specialize to add indexing extensions. For example, the *<index-see> (276)*, *<index-see-also> (278)*, and *<index-sort-as> (279)* elements only make sense as children of <indexterm> and so are specializations of <index-base>. Those elements are all part of the *indexing domain (272)*.

On its own, <index-base> has no meaning. Processors should ignore this element and its content if encountered in its unspecialized form.

Content models

See *appendix (426)* for information about this element in OASIS document type shells.

Inheritance

- topic/index-base

Example

The <index-see-also> element is specialized from <index-base>; see *index-see-also (278)* for an example of how <index-base> can be used with specialization.

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *@keyref (360)*.

3.7.6 <itemgroup>

The `<itemgroup>` element can be used to sub-divide or organize elements that occur inside a list item, definition, or parameter definition.

The `<itemgroup>` element is particularly useful as a basis for specialization, where it can be used to group content within specialized list items or definitions. For example, in the OASIS task specialization, many elements within the `<step>` element are specialized from `<itemgroup>`.

Content models

See *appendix (427)* for information about this element in OASIS document type shells.

Inheritance

- topic/itemgroup

Example

```
<li>Second point of a list.
 <itemgroup>related discourse</itemgroup>
</li>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.7.7 <no-topic-nesting>

The `<no-topic-nesting>` element is a placeholder in the DITA architecture. It is not actually used by the default DITA document types; it is for use only when creating a validly customized document type where the information designer wants to eliminate the ability to nest topics. Not intended for use by authors, and has no associated output processing.

Content models

See *appendix (433)* for information about this element in OASIS document type shells.

Inheritance

- topic/no-topic-nesting

Example

This element is not intended to be used in source files.

Attributes

The following attributes are available on this element: *Debug attribute group (350)* and *class (Not for use by authors)*.

3.7.8 <state>

The <state> element specifies a name/value pair whenever it is necessary to represent a named state that has a variable value. The element is primarily intended for use in specializations to represent specific states (like logic circuit states, chemical reaction states, airplane instrumentation states, and so forth).

Content models

See *appendix (444)* for information about this element in OASIS document type shells.

Inheritance

- topic/state

Example

```
<step><cmd>Verify the presence of  an "on" or high condition at the input gate
(ie,  <state name="inflag" value="high"/>)</cmd></step>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)*, *outputclass*, and the attributes defined below.

@name (REQUIRED)
　The name of the property whose state is being described.

@value (REQUIRED)
　The state of the property identified by the @name attribute.

3.7.9 <unknown>

The <unknown> element is an open extension that allows information architects to incorporate xml fragments that do not necessarily fit into an existing DITA use case. Processors should ignore this element unless otherwise instructed.

Content models

See *appendix (454)* for information about this element in OASIS document type shells.

Inheritance

- topic/unknown

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.

```
<body>
  <my-unknown class="+ topic/unknown mything/my-unknown ">
    <thing value="4"/>
    <otherthing value="16"/>
  </my-unknown>
</body>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* and *outputclass*.

3.8 Legacy conversion elements

Conversion elements exist primarily to aid in the conversion of content to DITA.

3.8.1 <required-cleanup>

A `<required-cleanup>` element is used as a placeholder for migrated elements that cannot be appropriately tagged without manual intervention. As the element name implies, the intent for authors is to clean up the contained material and eventually remove the `<required-cleanup>` element. Authors should not insert this element into documents.

Processing notes:

- Processors must strip this element from output by default. The content of `<required-cleanup>` is not considered to be verified data.

- Processor options might be provided to allow a draft view of migrated content in context.

Content models

See *appendix (440)* for information about this element in OASIS document type shells.

Inheritance

- topic/required-cleanup

Example

Presuming an original HTML document had contained some content within a `<center>` tag (for which there is no clear migrational equivalent in DITA), the following might be the result that is valid within an XML editor, but which requires an author to decide how to better tag or revise this original content:

```
<section>
  <title>Some section title</title>
  <required-cleanup remap="center">Some original content migrated
  from a &lt;center> tag.</required-cleanup>
</section>
```

Attributes

The following attributes are available on this element: *Universal attribute group (347)* (with a modified definition of `@translate`, given below), *outputclass*, and the attributes defined below.

@remap

Indicates the element that the contents of the `<required-cleanup>` element were mapped from (provides an idea about what the new intent should be).

@translate

Indicates 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 (362)* 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.

3.9 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 Links

2.2.4.2.1 Conditional processing attributes (50)
> *The metadata attributes specify properties of the content that can be used to determine how the content should be processed. Specialized metadata attributes can be defined to enable specific business-processing needs, such as semantic processing and data mining.*

2.4.3 Conditional processing (profiling) (95)
> *Conditional processing, also known as profiling, is the filtering or flagging of information based on processing-time criteria.*

2.4.3.1 Conditional processing values and groups (95)
> *Conditional processing attributes classify elements with metadata. The metadata is specified using space-delimited string values or grouped values.*

3.9.1 <val>

The `<val>` element is the root element of a DITAVAL file.

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 *Conditional processing (profiling) (95)*.

Contains

style-conflict (342) (*optional*) then (*prop (342)* or *revprop (344)*) (*any number*)

Example

```
<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 85: Sample DITAVAL file

```
<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 86: DITAVAL file that overrides the default "include" action

```
<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 *Conditional processing (profiling) (95)* for suggestions on how to handle similar groups on different attributes.

Figure 87: DITAVAL with conditions for groups

Attributes

This element does not define any attributes.

Related Links

2.4.3.2 Filtering (96)
> 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.

2.4.3.3 Flagging (98)
> 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.

2.4.3.5 Examples of conditional processing (100)
> This section provides examples that illustrate the ways that conditional processing attributes can be set and used.

3.9.2 <style-conflict>

The `<style-conflict>` element declares behavior to be used when one or more flagging methods collide on a single content element.

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.

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> / **<endflag>**	Add all flags that apply.
color	Follow the `<style-conflict>` `@foreground-conflict-color` setting, or use an output-appropriate default color if no conflict color is set.
backcolor	Follow the `<style-conflict>` `@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.

Contains

(empty)

Contained by

val *(339)*

Example

See the example in the *<val>* *(339)* description.

Attributes

The following attributes are available on this element:

@foreground-conflict-color
  The color to be used when more than one flagging color applies to a single content element.

@background-conflict-color
  The color to be used when more than one flagging background color applies to a single content element.

3.9.3 <prop>

The `<prop>` element identifies an attribute, and usually values in the attribute, to take an action on. The attribute either must be a conditional-processing attribute (`@platform`, `@product`, `@audience`, `@deliveryTarget`, `@props`, and `@otherprops`) or a specialization of the `@props` attribute.

A `<prop>` element can do one of the following:

- A `<prop>` element with no `@att` attribute specified sets a default action for every `<prop>` element. It is an error to use more than one `<prop>` element with no attribute in a single document. Recovery from this error is implementation dependent; in such cases processors *MAY* provide an error or warning message.

- A `<prop>` element with an `@att` attribute but no `@val` attribute sets a default action for that specific attribute or attribute group. For each specific attribute, it is an error to use more than one `<prop>` element with that

attribute and no value in a single document. Recovery from this error is implementation dependent; in such cases processors *MAY* provide an error or warning message.

- A `<prop>` element with an `@att` attribute and a `@val` attribute sets an action for that value within that attribute or attribute group. It is an error to use more than one `<prop>` element with the same attribute and value. Recovery from this error is implementation dependent; in such cases processors *MAY* provide an error or warning message.

Contains

startflag (345) (*optional*) then *endflag (346)* (*optional*)

Contained by

val (339)

Example

See the example in the *<val> (339)* description.

Attributes

The following attributes are available on this element:

@att

The attribute to be acted upon. If using a literal attribute name, it must be one of `@props`, `@audience`, `@platform`, `@product`, `@otherprops`, `@deliveryTarget`, or a specialization of `@props`. Otherwise, the value should be the name of a group used within the `@audience`, `@platform`, `@product`, or `@otherprops` attributes. If the `@att` attribute is absent, then the `<prop>` element declares a default behavior for any conditional processing attribute.

@val

The 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)

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 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).

@color

If flag has been set, the color to use to flag text. Colors can be entered by name or code. Processors *SHOULD* support the color names listed under the heading "<color>" in *http://www.w3.org/TR/2006/REC-xsl11-20061205/ #datatype* and for the 6 digit hex code form (#rrggbb, case insensitive). If flag has not been set, this attribute is ignored.

@backcolor

If flag has been set, the color to use as background for flagged text. Colors can be entered by name or code. Processor support is recommended for the color names listed under the heading "<color>" in *http://www.w3.org/TR/*

2006/REC-xsl11-20061205/#datatype and for the 6 digit hex code form (#rrggbb, case insensitive). If flag has not been set, this attribute is ignored.

@style

If flag has been set, the text styles to use for flagged text. This attribute can contain multiple space-delimited tokens. The following tokens *SHOULD* be processed by all DITAVAL processors:

- underline

- double-underline

- italics

- overline

- bold

In addition, processors might support other proprietary tokens for different types of styling. 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.

If flag has not been set, this attribute is ignored.

3.9.4 <revprop>

Identifies a value in the `@rev` attribute that should be flagged in some manner. Unlike the other conditional processing attributes, which can be used for both filtering and flagging, the `@rev` attribute can only be used for flagging.

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.

When no alternate text is specified for a revision flag, the default alternate text for `<revprop>` start of change is a localized translation of "Start of change", and the default alternate text for `<revprop>` end of change is a localized translation of "End of change".

The `@rev` attribute identifies when a particular section of a document was added in the lifetime of that document. 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.

Contains

startflag (345) (*optional*) then *endflag* (346) (*optional*)

Contained by

val (339)

Example

See the example in the *<val> (339)* description.

Attributes

The following attributes are available on this element:

@val

The 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)

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 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).

@changebar

When flag has been set, specify a changebar color, style, or character, according to the changebar support of the target output format. If flag has not been set, this attribute is ignored.

@color

If flag has been set, the color to use to flag text. Colors can be entered by name or code. Processors *SHOULD* support the color names listed under the heading "<color>" in *http://www.w3.org/TR/2006/REC-xsl11-20061205/#datatype* and for the 6 digit hex code form (#rrggbb, case insensitive). If flag has not been set, this attribute is ignored.

@backcolor

If flag has been set, the color to use as background for flagged text. Colors can be entered by name or code. Processor support is recommended for the color names listed under the heading "<color>" in *http://www.w3.org/TR/2006/REC-xsl11-20061205/#datatype* and for the 6 digit hex code form (#rrggbb, case insensitive). If flag has not been set, this attribute is ignored.

@style

If flag has been set, the text styles to use for flagged text. This attribute can contain multiple space-delimited tokens. The following tokens *SHOULD* be processed by all DITAVAL processors:

- underline

- double-underline

- italics

- overline

- bold

In addition, processors might support other proprietary tokens for different types of styling. 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.

If flag has not been set, this attribute is ignored.

Related Links

2.4.3.3 Flagging (98)

> *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.*

3.9.5 <startflag>

The `<startflag>` takes an optional image reference and an optional `<alt-text>` element for identifying the beginning of flagged content. If an image is specified, the specified image will be used to flag the beginning of the content, with the `<alt-text>` contents as alternative text. If `<alt-text>` is specified without an image, that text will 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. The `<startflag>` element is allowed inside either `<prop>` or `<revprop>`.

Contains

alt-text (346) (*optional*)

Contained by

prop (342), *revprop (344)*

Example

See the example in the *<val> (339)* description.

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 *The href attribute (359)* for information on supported values and processing implications.

3.9.6 <endflag>

The `<endflag>` element takes an optional image reference and an optional `<alt-text>` element for identifying the end of flagged content. If an image is specified, the specified image will be used to flag the end of the content, with the `<alt-text>` contents as alternative text. If `<alt-text>` is specified without an image, that text will 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. The `<endflag>` element is allowed inside either `<prop>` or `<revprop>`.

Contains

alt-text (346) (*optional*)

Contained by

prop (342), *revprop (344)*

Example

See the example in the *<val> (339)* description.

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 *The href attribute (359)* for information on supported values and processing implications.

3.9.7 <alt-text>

The `<alt-text>` element is allowed inside either `<startflag>` or `<endflag>` to provide alternate text for an image when the `@imageref` attribute sets an image to be used for flagging. When no alternate text is specified for a revision flag, the default alternate text for `<revprop>` start of change is a localized translation of "Start of change", and the default alternate text for `<revprop>` end of change is a localized translation of "End of change".

Contains

text data

Contained by

startflag (345), *endflag (346)*

Example

See the example in the *`<val>` (339)* description.

Attributes

This element does not define any attributes.

3.10 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.

3.10.1 Universal attribute group

The universal attribute group defines a set of common attributes available on most DITA elements. The universal attribute group includes all attributes from the metadata, ID, localization, and debug attribute groups, plus the `@class` attribute.

Note: Earlier versions of the DITA specification grouped some attributes under names taken directly from parameter entities in the standard DTD implementation of DITA. These groups used names such as "univ-atts", which had no meaning outside the context of a DTD module. In DITA 1.3, these groups are restructured and renamed in the specification document for greater clarity, and no longer have a connection to the DTD. For example, this grouping of "Universal attributes" was previously referred to as "univ-atts"; the updated group includes several universal attributes that are not part of the univ-atts group in the DTD.

@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 *class attribute rules and syntax (129)* 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.

3.10.1.1 ID attribute group

The ID attribute group includes attributes that enable the naming and referencing of elements in topics and maps.

@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 *ID attribute (62)* in the Architectural Specification for more details.

@conref

> This attribute is used to reference an ID on content that can be reused. See *The conref attribute (361)* 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 *The conrefend attribute (366)* for examples.

@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 *The conaction attribute (363)* for examples and details about the syntax.

@conkeyref

Allows the conref feature to operate using a key instead of a URI. See *The conkeyref attribute (369)* for more details about the syntax and behaviors.

3.10.1.2 Metadata attribute group

The metadata attribute group includes common metadata attributes, several of which support conditional processing (filtering and flagging) or the creation of new attribute domain specializations.

@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.

@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 `@props` and `@base` attributes each take a space-delimited set of values. However, when acting as a container for generalized attributes, the attribute values will be more complex; see *Attribute generalization (138)* for more details.

@platform

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

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.

@audience

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.

@otherprops

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.

The `@audience`, `@platform`, `@product`, and `@otherprops` attributes are property attributes which support conditional processing for filtering or flagging. Each takes a space-delimited set of values, with optional groups of space-delimited properties. Although these attributes are not specialized and not specializeable, the group syntax matches that for generalized attributes in *Attribute generalization (138)*.

@deliveryTarget

The intended delivery target of the content, for example "html", "pdf", or "epub". This attribute is a replacement for the now deprecated `@print` attribute.

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.

@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.

@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.

Related Links

2.2.4.2.1 Conditional processing attributes (50)
The metadata attributes specify properties of the content that can be used to determine how the content should be processed. Specialized metadata attributes can be defined to enable specific business-processing needs, such as semantic processing and data mining.

2.4.3 Conditional processing (profiling) (95)
Conditional processing, also known as profiling, is the filtering or flagging of information based on processing-time criteria.

2.4.3.1 Conditional processing values and groups (95)
Conditional processing attributes classify elements with metadata. The metadata is specified using space-delimited string values or grouped values.

2.4.3.2 Filtering (96)
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.

2.5.3 Specialization (126)
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.

2.6.3.6 DTD: Coding requirements for attribute domain modules (158)
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 `@domains` attribute for the attribute domain.

3.10.1.3 Localization attribute group

The localization attribute group defines a set of common attributes related to translation and localization. These attributes are available on most DITA elements.

@translate

Indicates whether the content of the element should be translated or not. Allowable values are yes, no, and -dita-use-conref-target. See *Element-by-element recommendations for translators: Base edition (391)* 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.

@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 bidi 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 *The dir attribute (120)* for more information.

The `@translate`, `@xml:lang`, and `@dir` attributes identify language-specific words or phrases for specific processing (or non-processing, in the case of `translate="no"`).

```
<p>The cordial response to the question is
<q translate="no" xml:lang="de-de" dir="ltr">nein.</q></p>
```

Related Links

2.4.6 Translation and localization (119)

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>` and `<index-sort-as>` elements provide 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.

3.10.1.4 Debug attribute group

The debug attributes group includes a set of debugging attributes that are normally hidden from authoring view. These attributes are defined for every DITA element.

These attributes are intended to store debugging information during intermediate processing. One possible implementation is to use `@xtrf` (xml-trace-filename) to store the original source file name through intermediate processing steps, and use `@xtrc` (xml-trace-counter) to store an element counter for repositioning authoring tools at the originating element location. If values are assigned in this way in the first stage of a processing stream, values in these attributes can then be used for error recovery.

Note: While this group of attributes is defined with the "Universal attribute group" within the specification, OASIS implementations of the DITA vocabulary do not currently group it with the common "univ-atts" attributes group.

These attributes are normally hidden from authors and exposed only to processing tools or editor macros. Values used in these attributes might be implementation-dependent.

@xtrf

xml-trace-filename, the original filename.

@xtrc

xml-trace-counter, an element counter for repositioning editors at a known edit location.

3.10.2 Architectural attribute group

The architectural attributes group includes a set of attributes defined for document level elements such as `<topic>` and `<map>`. These attributes are intended to provide information about the DITA namespace, what level of DITA is in use, and what vocabulary modules are in use.

@DITAArchVersion

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 "1.3".

@xmlns:ditaarch

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/".

@domains

Indicates the specialized domains that are included in the DTD or Schema. This attribute is defined with the XML data type CDATA, and each new document type *SHOULD* specify a default. The value will differ depending on what

domains are included in the current DTD or Schema; a sample value is "(topic ui-d) (topic hi-d) (topic pr-d) (topic sw-d) (topic ut-d) (topic indexing-d)".

3.10.3 Attributes common to many map elements

This attribute group collects several attributes that are used on a variety of map elements. For a few elements, the group is modified slightly to remove an attribute such as `@toc` or `@format`; in those cases the element definition will clarify that the element does not use this full set. That is generally done in order to specify a default for one attribute, such as defaulting `@format` to "ditamap" on the `<mapref>` element.

@cascade

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 *Cascading of metadata attributes in a DITA map (53)* for more information about how this attribute interacts with metadata attributes.

@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 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.

-dita-use-conref-target

See *Using the -dita-use-conref-target value (362)* for more information.

Usage of the `@collection-type` attribute on `<reltable>` and `<relcolspec>` is currently undefined and reserved for future use.

@processing-role

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 *Using the -dita-use-conref-target value (362)* for more information.

@locktitle

If the `@locktitle` attribute is set to "yes", the content of the `<navtitle>` element or `@navtitle` attribute is used for a navigation title, if it is present. If the `@locktitle` attribute is not present or set to "no", the content of the `<navtitle>` element or `@navtitle` attribute is ignored, and the title of the referenced topic is used as a navigation title.

Note: The `@navtitle` attribute is deprecated in favor of the `<navtitle>` element. When both a `<navtitle>` element and a `@navtitle` attribute are specified, the `<navtitle>` element should be used.

Allowable values for `@locktitle` are:

yes

The content of the `<navtitle>` element or `@navtitle` attribute is used for a navigation title.

no

The content of the `<navtitle>` element or `@navtitle` attribute is ignored. This is the processing default.

-dita-use-conref-target

See *Using the -dita-use-conref-target value (362)* for more information.

@linking

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 *Cascading of metadata attributes in a DITA map (53)*). 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 *Using the -dita-use-conref-target value (362)* for more information.

@toc

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 *Cascading of metadata attributes in a DITA map (53)*). 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 *Using the -dita-use-conref-target value (362)* for more information.

@print

Specifies whether the topic should be included in a print-specific rendition, such as PDF. The processing default is "yes". If the value is not specified locally, the value might cascade from another element in the map (for cascade rules, see *Cascading of metadata attributes in a DITA map (53)*).

Note: Beginning with DITA 1.3, the `@print` attribute is deprecated. It is replaced with a conditional processing attribute: `@deliveryTarget`. See *`@deliveryTarget`* for more details.

Allowable values are:

yes

Include the topic in the print-oriented file.

no

Do not include the topic in a print-oriented file.

printonly

Only include the topic when rendering the DITA content in a print-oriented context; the topic is not appropriate for other contexts, such as when rendering as HTML.

-dita-use-conref-target

See *Using the -dita-use-conref-target value (362)* for more information.

@search

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 *Cascading of metadata attributes in a DITA map (53)*). Allowable values are:

yes

no

-dita-use-conref-target (362)

@chunk

When a set of topics is transformed using a map, the `@chunk` attribute allows multi-topic documents 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 *Chunking (114)*.

@keyscope

Specifies that the element marks the boundaries of a key scope. See *The keyscope attribute (361)* for details on how to use the `@keyscope` attribute.

Related Links

2.2.4.4 Cascading of metadata attributes in a DITA map (53)

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.

3.10.4 Complex-table attribute group

The Complex-table attribute group includes several attributes that are defined on complex table 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. These attributes are not available for the `<simpletable>` elements.

@align

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 *Using the -dita-use-conref-target value (362)* for more information.

The `@align` attribute is available on the following table elements: `<tgroup>`, `<colspec>`, and `<entry>`.

@char

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

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>`.

@colsep

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>`.

@rowsep

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

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 *Using the -dita-use-conref-target value (362)* 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>`.

@valign

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 *Using the -dita-use-conref-target value (362)* for more information.

The `@valign` attribute is available on the following table elements: `<thead>`, `<tbody>`, `<row>`, and `<entry>`.

3.10.5 Data element attributes group

The data element attributes group includes attributes that are defined for the `<data>` element, and are reused on most or all specializations of the `<data>` element.

@name

Defines a unique name for the object.

@datatype

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.

@value

Specifies a value associated with the current property or element.

3.10.6 Date attributes group

The date attributes group includes attributes that take date values, and are defined on metadata elements that work with date information.

@expiry

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.

@golive

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.

3.10.7 Display attribute group

The display attribute group includes attributes whose values can be used for affecting the display of many elements.

@expanse

Determines the horizontal placement of the element. Allowable values are:

page

Places the element on the left page margin for left-to-right presentation, or right page margin for right-to-left presentation.

column

Aligns the element with the current column margin

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.

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".

-dita-use-conref-target

See *Using the -dita-use-conref-target value (362)* 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.

@frame

Specifies which portion of a border should surround the element. Allowable values are:

top

Draw a line before the element

bottom

Draw a line after the element

topbot

Draw a line both before and after the element

all

Draw a box around the element

sides

Draw a line at each side of the element

none

Don't draw any lines around this element

-dita-use-conref-target

See *Using the -dita-use-conref-target value (362)* for more information.

Some DITA processors or output formats might not be able to support all values.

@scale

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 (362)*. Some DITA processors or output formats might not be able to support all values.

3.10.8 Link relationship attribute group

The link relationship attribute group includes attributes whose values can be used for representing navigational relationships. These attributes occur only on elements that represent relationships among DITA elements or between DITA elements and non-DITA resources.

@href

   Provides a reference to a resource. See *The href attribute (359)* for detailed information on supported values and processing implications..

@format

   The `@format` attribute identifies the format of the resource being referenced. See *The format attribute (372)* for details on supported values.

@scope

   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 *The scope attribute (373)* for more information on these values.

@type

   Describes the target of a reference. See *The type attribute (370)* for detailed information on supported values and processing implications.

3.10.9 Other common attributes

These common attributes are used across a wide variety of elements, with the common definition included below. These attributes are not defined as a group, and many elements only specify one or two from this list.

@keyref

   `@keyref` provides a redirectable reference based on a key defined within a map. See *The keyref attribute (360)* for information on using this attribute.

@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.

@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 *Using the -dita-use-conref-target value (362)* for more information.

@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".

@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.

@mapkeyref

Identifies the map, if any, from which the contained links or metadata are derived. This value might be automatically generated by a process that creates the links or metadata based on map context, as a way to identify which map the material came from. If the `<linklist>`, or `<linkpool>`, or metadata is manually created by an author, there is no need to use this attribute. Note that this attribute is not related to the `@keyref` attribute, and is not used for key based processing.

3.10.10 Simpletable attribute group

The simpletable attribute group includes several attributes that are defined on the `<simpletable>` element and `<simpletable>` specializations. These attributes are not defined for the OASIS exchange table (`<table>`).

@keycol

Defines the column that can 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.

@relcolwidth

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%).

@refcols

The `@refcols` attribute is currently undefined, and is reserved for future use.

3.10.11 Specialization attributes group

These attributes are designed to be used by specializers, and are not intended for direct use by authors.

@specentry

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.

@spectitle

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.

3.10.12 Topicref element attributes group

The topicref element attributes group includes attributes that are defined for the `<topicref>` element, and are reused on most or all specializations of the `<topicref>` element.

@copy-to

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.

@navtitle

Specifies the title of the topic as it will appear in the navigation or tables of contents that are generated from the map. Beginning with DITA 1.2, the preferred way to specify the navigation title in a map is with the `<navtitle>` element, available inside the `<topicmeta>` element.

@query (DEPRECATED)

This attribute is deprecated. It might be removed in the future.

3.10.13 Complex attribute definitions

Several DITA attributes require more explanation than can fit in a single table cell. Those attributes are collected here.

3.10.13.1 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*]. It is an error if the value 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 *Processing xrefs and conrefs within a conref (92)* 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"`

Target	Syntax
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.

3.10.13.2 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 Links

    2.3.4 Indirect key-based addressing (66)

        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.

3.10.13.3 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 Links

    2.3.4 Indirect key-based addressing (66)

        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.

3.10.13.4 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"`).

3.10.13.5 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 *URI-based (direct) addressing (63)* 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 Links

2.4.2 Content reference (conref) (90)

> 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

2.4.2.1 Conref overview (90)

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.

3.10.13.5.1 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 map where the `<topichead>` element uses `@conref`. It specifies the `@navtitle` attribute as well as the `@toc` attribute. In the resolved element, `@navtitle` 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.

Note: In earlier versions of DITA, `@navtitle` was required on the `<topichead>` element. While it is no longer required, the example still illustrates the expected processing for both required and non-required attributes.

```
<map><title>Conref demonstration</title>
  <topichead id="heading"
             navtitle="This is a heading"
             toc="yes"
             linking="normal">
    <topicref href="topic.dita" navtitle="content"/>
  </topichead>

  <topichead conref="#heading"
             navtitle="-dita-use-conref-target"
             toc="no">
  </topichead>
</map>
```

Figure 88: Pre-resolution

```
<map><title>Conref demonstration</title>
  <topichead id="heading"
             navtitle="This is a heading"
             toc="yes"
             linking="normal">
    <topicref href="topic.dita" navtitle="content"/>
  </topichead>

  <topichead navtitle="This is a heading"
             toc="no"
             linking="normal">
    <topicref href="topic.dita" navtitle="content"/>
  </topichead>
</map>
```

Figure 89: Effective result post-resolution

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

When resolving conrefs, processors need to combine the attributes that are specified on the referencing and referenced element.

3.10.13.6 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 "example", and contains a `<step>` with the id "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.

Related Links

> *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.*

3.10.13.7 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

Several items must be taken into account when using or implementing `@conrefend`.

- Processors will resolve the range by pulling in the start target and following sibling XML nodes across to and including the end target.

- The start and end elements of a range *MUST* be of the same type as the referencing element or generalizable to the referencing element. For example, `@conref` and `@conrefend` attributes on an `<li>` element might reference other `<li>` elements, or they might reference specializations of `<li>` such as `<step>`.

- 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.

- 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. For example, it is possible to pull a range from `<conbody>` into `<body>`, because `<conbody>` is generalizable to `<body>`. It is not possible to pull a range from `<body>` into `<conbody>`, because the result might not be valid in `<conbody>`.

- 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

```
<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 90: List example: Source topic.dita with ids

```
    <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 91: List example: Reusing topic with conrefs

```
    <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>
```

Figure 92: List example: Processed result of reusing topic

Example: Reusing a set of blocks

```
<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 93: Block level example: Source topic.dita with ids

```
    <topic id="y">
        ...
        <body>
            <p conref="topic.dita#x/p1" conrefend="topic.dita#x/p2"/>
```

```
        </body>
    </topic>
```

Figure 94: Block level example: Reusing topic with conrefs

```
<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>
```

Figure 95: Block level example: Processed result of reusing 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 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

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 96: Defining and referencing a key with @conkeyref

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`.

Figure 97: Combining @conref, @conkeyref, and @conrefend

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.

Related Links

2.4.2.1 Conref overview (90)
> *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.*

3.10.13.8 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* *(368)* for more information and for examples of this behavior.

Related Links

2.4.2.1 Conref overview *(90)*

> 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.*

3.10.13.9 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 thatcan 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 *Using the -dita-use-conref-target value (362)* 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 *Using the -dita-use-conref-target value* *(362)* for more information.

3.10.13.10 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 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 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.

3.10.13.11 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 *Using the -dita-use-conref-target value* *(362)* 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.

3.10.13.12 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

Iindicates 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.

sample

Deprecated.

external

Deprecated--use the scope="external" attribute to indicate external links.

other

Indicates any other kind of relationship or role. Enter that role as the value for the `@otherrole` attribute.

-dita-use-conref-target

See *Using the -dita-use-conref-target value* *(362)* 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"`.

4 Conformance

Conformance to the DITA specification allows documents and document types that are used with different processors or different versions of a processor to produce the same or similar results with little or no reimplementation or modification.

4.1 Conformance of DITA implementations

A DITA implementation can consist of any combination of processing components that claim DITA awareness, custom vocabulary and constraint modules, or custom document-types shells.

For example, a DITA implementation might be a DITA-based authoring and publishing system within an enterprise; a task-specific product, such a DITA-aware XML editor or component management system; or a set of specialization and constraint modules that are packaged for integration with larger implementations.

Conforming DITA implementations *MUST* include a conformance statement that gives the version of the DITA specification that is supported. The conformance statement must include one of the following:

- List of the DITA features that are supported by the implementation in accordance with the requirements of the DITA specification
- Statement that the implementation includes all features except for a specific list of features that are not supported in accordance with the requirements of the DITA specification

Implementations that include some (but not all) DITA features are considered conforming as long as all of the features that are included follow the requirements that are given in the DITA specification.

An implementation that does not include a particular optional feature *MUST* be prepared to interoperate with other implementations that do include the feature, though perhaps with reduced functionality. An implementation that does include a particular optional feature *MUST* be prepared to interoperate with other implementations that do not include the feature.

Organizations and individuals can impose additional restrictions on their own use of DITA that go beyond the requirements imposed by the DITA specification. These restrictions can include enforcement of the restrictions by their local processors, as long as the result continues to meet the requirements that are given in the DITA specification. For example, a user community can impose rules on how files must be named or organized, even if those rules go beyond the requirements given in the DITA specification.

4.2 Conformance of DITA documents

A *conforming DITA document* is a document that meets all of the following criteria:

- It is a well-formed XML document.
- Its elements either are DITA elements or non-DITA elements that are contained within `<foreign>` or `<unknown>` elements.
- Its content conforms to all DITA requirements for element content and attribute values.

If using non-DITA-conforming grammar files for conforming DITA documents, those grammar files *MUST NOT* be structured in a way that affects the ability of processors to process those documents. The use of non-conforming document types or schemas might impede interchange or interoperation of those documents with tools that expect or require the use of conforming DITA document types or schemas.

4.3 Conformance of DITA processors

The conformance of processors can only be determined for *DITA-aware processors*. We define three types of DITA-aware processors.

DITA-aware processor

A processor that implements all required processing relevant to the vocabulary modules that it claims to support. A DITA-aware processor *MUST* support at least one map or topic type, whether defined by the DITA standard or defined as a custom vocabulary module.

Specialization-aware processor

A DITA-aware processor that can handle any document specialized from some set of DITA vocabulary modules, though it might not be able to support all base topic or map elements.

Fully aware processor

A DITA-aware processor that unconditionally supports all base vocabulary modules, which implies support for all non-vocabulary-specific DITA features, such as content references and key references. It applies relevant processing to all DITA elements based on their `@class` and `@domains` attribute values.

For example, a general-purpose processor that can process XML documents to produce a particular output using user-created configurations or scripts is not itself DITA-aware. However, that same processor packaged with a DITA-specific configuration or script would be a DITA-aware processor. A script or configuration for this processor that only operates on tag names as defined in specific DITA vocabulary modules would make the tool DITA aware but not specialization aware. A script or configuration that operated on DITA `@class` attribute values would be both DITA aware and specialization aware.

In general, specialization-aware processors will be able to reliably process all conforming DITA documents, providing at least some default behavior for all DITA elements, while DITA-aware processors might only be able to reliably process documents that use the vocabulary modules that those processors support.

DITA-aware processors support the following functional areas:

Production of output

The production of final form output from DITA documents. Examples of processors that perform this function are publishing systems and tools, such as the DITA Open Toolkit.

Editing, managing, and storage

The editing, managing, and storage of DITA documents. Examples of processors that perform this function are authoring tools and component content management systems (CCMS).

A given processor might provide some or all of this function. For example, a DITA-aware editor that includes the ability to generate print versions of DITA documents represents both a final-form processor and an editing processor. Likewise, a content or component management system might tightly integrate final-form DITA processors. Each processor type might have different conformance requirements, even though the processors are part of a single product or package.

For processors that produce final form output, all features that are relevant to the type of processing that the processor performs *MUST* be implemented, with the exception of features that are vocabulary-specific. In particular, such processors *MUST* implement address resolution and content reference resolution. Such processors *SHOULD* implement filtering.

For example, a specialization-aware processor that produces final-form output need not provide special presentation results for glossary entry topics, but it must implement resolution of key-based references to glossary entry topics from `<keyword>` or `<term>` elements, because address resolution is both required and not vocabulary specific.

Processors that store, manage, or edit DITA documents might not implement specific features that would be required for final-form processing. However, such processors *MUST* enable the creation or storage of DITA documents that use all DITA features, even if the processor is not aware of the DITA semantics for those features.

For example, a DITA-aware editor need not provide specific support for creating or resolving content references, but it must allow, using normal XML editing methods, the creation and editing of content references. A content management system that supports map types that allow relationship tables but does not directly support

relationship table processing must be able to store and manage conforming map documents that include relationship tables.

Appendix 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, IBM
Deb Bissantz, LSI
Doug Burgess, SAP
Tom Cihak, Freescale Semiconductor, Inc.
Don Day, Individual Member
Stanley Doherty, Individual Member
Kristen James Eberlein, Eberlein Consulting LLC
JoAnn Hackos, Comtech Services, Inc.
Richard Hamilton, Individual Member
Nancy Harrison, Individual Member
David Helfinstine, PTC
Scott Hudson, Comtech Services, Inc.
John Hunt, IBM
Eliot Kimber, Individual Member
Tom Magliery, JustSystems
Mark Myers, SAP
Chris Nitchie, Oberon Technologies
Joe Pairman, Mekon
Seth Park, Freescale Semiconductor, Inc.
Mark Poston, Mekon
Michael Priestley, IBM
Tony Self, Individual Member
Eric Sirois, IXIASOFT
Dawn Stevens, Comtech Services, Inc.
Amber Swope, Individual Member
Bob Thomas, Individual Member
Jim Tivy, Bluestream

In addition, the OASIS DITA Technical Committee also would like to recognize the following people for their insights and support:

George Bina
Jarno Elovirta

Appendix B Non-normative information

This section contains non-normative information, including topics about new features in DITA 1.3 and migrating from DITA 1.2 to DITA 1.3.

B.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.

As a convenience for readers, the DITA specification is published in several packages that contain different combinations of required and optional modules. The ability to easily define, manage, and publish these packages is a result of using DITA for the specification source. The source files for the DITA specification are managed in a version control 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, XMetaL Author Enterprise, and Arbortext Editor
- DITAweb
- Antenna House Formatter

B.2 Changes from previous versions

The following topics outline the changes from earlier versions of DITA to the current version. Each version of the DITA specification is designed to be backwards-compatible with applications that conform to earlier versions of the DITA specification.

B.2.1 Changes from DITA 1.2 to DITA 1.3

DITA 1.3 adds a number of new features and refinements to DITA.

Enhancements to keys

- Added scoped keys
- Added facility for key-based, cross-deliverable addressing

New XML grammars

The DITA standard now supports RELAX NG.

Enhancements to conditional processing (filtering)

- Enabled information architects to associate DITAVAL conditions with specific topic collections in a map (branch filtering). Prior to DITA 1.3, a set of DITAVAL conditions could only be associated with a root map.
- Expanded syntax for filtering attributes. The original property attributes (`@audience`, `@platform`, `@product`, and `@otherprops`) now accept grouped values, using the same syntax for generalized attribute values within `@props` and `@base`.

Enhancements to addressing

Added new (abbreviated) syntax for addressing an element within the same DITA topic

Enhancements to specialization

Enabled individual element types of structural specializations to be reused by other structural specializations without requiring one to be specialized from the other. For example, allowed the `<steps>` element from `<task>` to be reused in place of an `<ol>` within a `<remedy>` section of the troubleshooting document type.

Increased support for producing troubleshooting information

- Added a new value, "trouble", for the `@type` attribute of the `<note>` element

Increased support for producing user assistance

- Added the following new attributes to `<resourceid>`:

  - `@appid`

  - `@ux-context-string`

  - `@ux-source-priority`

  - `@ux-windowref`

- Added a new element, `<ux-window>`, designed to specify a window or viewport in which an online help topic or Web page is displayed

New base elements and attributes

- Added the `<div>` element. This element enables arbitrary grouping of content within a topic. It is primarily intended as a way to group content for content referencing or as a basis for specialization.

- Added new conditional processing attribute, `@deliveryTarget`, to replace the now-deprecated `@print` attribute. Users can set controlled values for the `@deliveryTarget` attribute by using a subject scheme map.

- Added `@orient` to `<table>` and `@rotate` to `<entry>`. These new attributes enable authors to control formatting more precisely for print-based output formats.

- Added `@scope` and `@headers` attributes to the `<entry>` element in tables. These new attributes are intended for extremely complex tables where accessibility cannot be handled based on existing header markup.

- Added a new attribute, `@cascade`, to better control how metadata attributes cascade within a DITA map

Modifications to the base content model

- Enabled `<draft-comment>` in more places

- Enabled `<text>` in more places

- Extended content model of `<data>` to allow additional phrase level markup.

- Extended the list of values allowed for the `@style` attribute on `<prop>` and `<revprop>` elements in DITAVAL files.

- Enabled `<ph>` within `<indexterm>`. This makes it possible to use specializations of the `<ph>` element, such as `<sup>` and `<sub>`, within index entries.

- Enabled `@keyref` for `<object>` and `<param>`

- Added `<data>` to the content model for lists

- Added new default values for the `@format` and `@scope` attributes

- Added `@rev` attribute to `<title>`

- Added `<cite>` to the content model of `<title>` and similar elements

- Added `<xref>` to the content model for `<shortdesc>`

Refinements to base domains

- Added element for sorting, `<sort-as>`, to the utilities domain

- Added new elements `<line-through>` and `<overline>` to the highlighting domain

B.2.2 Changes from DITA 1.1 to DITA 1.2

DITA 1.2 added a number of new features to DITA, including indirect addressing using map-defined keys; the ability to define content-model constraints for DITA document types; specializations for learning content and the machine industry; and taxonomies, ontologies, and controlled vocabularies. Other refinements include extended markup for glossaries and terminology.

New features

The following features are new in DITA 1.2:

• Keys and key references.

• Constraint modules. Constraint modules allow base content models to be further restricted without the need for specialization. For example, a constraint module can make optional elements required or disallow optional elements in a specific content model.

• Topic and map specializations for learning and training information, including interactive assessments.

• New elements for use with glossary entry topics for more complete description of terms, definition of acronyms, and so on.

• New map specialization for defining controlled vocabularies and taxonomies.

• New machine-industry task specialization.

• New general-task information type.

New element types

The following base element types are new in DITA 1.2:

<text>

  Allowed in most contexts where text is allowed but neither `<ph>` nor `<keyword>` are allowed. Enables reuse of text in almost any context.

<bodydiv>

  Allows creation of untitled containers within topic bodies. Intended primarily for specialization.

<sectiondiv>

  Allows creation of untitled containers within sections. Intended primarily for specialization.

<keydef>

  Topicref specialization for defining keys. Sets the default value for the processing-`@role` attribute to "resource-only".

<mapref>

  Topicref specialization for referring to DITA maps. Sets the default value for the `@format` attribute to "ditamap".

<topicset>

  Used to define sets of topicrefs that represent an atomic unit of reusable navigation structure. Requires the `@id` attribute be specified.

<topicsetref>

  References a `<topicset>` element. Enables preservation of the identity of the referenced topic set.

<anchor>

  Defines a point within a map to which `<topicref>` elements can be bound using the `<anchorref>` element.

<anchorref>

  "Pushes" one or more `<topicref>` elements onto an anchor point defined by an `<anchor>` element. Similar to a conref push but allows the relationship to be managed dynamically by the renderer.

Refinements to maps

- Map elements can use the `<title>` element in place of the `@title` attribute.

- Relationship table elements can have `<title>` as an optional first child.

- Topicref elements can use the `<navtitle>` element in place of the `@navtitle` attribute.

- Maps and topicrefs can now contain the same metadata elements as topic prologs.

- New `<topicref>` attribute named `@processing-role`. Indicates whether or not a topic reference contributes to the navigation structure of the containing map.

Refinements to content references

- Content references can now point to ranges of elements. For example, a single content reference from a `<step>` element can include a sequence of `<step>` elements.

- Content references can "push" elements into a target context, allowing unilateral augmentation of topics from other topics. For example, given a base topic with generic content, a using map could include both the generic topic and a separate topic that uses conref push to add map-specific content to the generic topic.

- Content reference resolution can be deferred so that it is done later in a rendering process or completely deferred so that it can be done by a separate delivery mechanism, for example, Eclipse information centers.

Refinements to topic elements

- The base task topic type has a more relaxed content model. This enables creation of a wider variety of specialized tasks, including task specializations that do not have formal markup for individual steps. The strict-task document type integrates a constraint module that imposes the same constrained content model as defined in the DITA 1.1 task topic type.

- A number of content elements allow the new `@keyref` attribute, including the `<ph>`, `<keyword>`, and `<term>` elements. When using the `@keyref` attribute, these elements can get their effective content from the key-defining `<topicref>` element and can also be treated as navigation links to the resource pointed to by the key-defining `<topicref>` element, if any. For example, a `<term>` element can use `@keyref` to link to the glossary entry topics for the term.

- The `<image>` element takes the new `@scalefit` attribute, which indicates whether or not the image should be scaled to fit the presentation context.

- The `<draft-comment>` element is now allowed in most contexts.

- The `<figgroup>` element now allows `<data>` as a subelement.

Refinements to specialization

- Structural and domain vocabulary modules can now both be listed in the `@domains` attribute. Structural modules can depend on and specialize elements from domains. For example, a structural domain for reference topics for a specific programming language could depend on the Programming domain (pr-d) and specialize elements from that domain.

- Information architects can indicate whether the use of a given vocabulary module requires strict or weak checking of content reference constraints.

- The implementation patterns for vocabulary modules have been refined. In particular, each element type now defines a separate parameter entity for its content model and attribute list, allowing per-element configuration of content models and attribute lists through constraint modules.

Other refinements

- The `<dita>` element now has the `@DITAArchVersion` attribute.

- A number of processing details have been clarified where they were underspecified in DITA 1.1.
- Most attributes that had enumerated values in DITA 1.1 are now unenumerated, allowing specializations to define different enumerations if they choose.

B.2.3 Changes from DITA 1.0 to DITA 1.1

The DITA 1.1 specification was designed to be backwards-compatible with applications that conform to the DITA 1.0 specification.

The following major changes to the architecture provided added functionality for DITA 1.1:

- A `<bookmap>` specialization for encoding book-specific information in a DITA map
- A `<glossentry>` specialization for glossary entries
- Indexing specializations for see, see-also, page ranges, and sort order
- Improvements to graphic scaling capability
- Improved short description flexibility through a new `<abstract>` element
- Specialization support for new global attributes, such as conditional processing attributes
- Support for integration of existing content structures through the `<foreign>` element
- Support for new kinds of information and structures through the `<data>` and `<unknown>` elements
- Formalization of conditional processing profiles

B.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 "sw-d" for the "software" domain, where "software" is the short name and "sw-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 other declarations	*ModuleName*Mod.xsd	conceptMod.xsd
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, "Taskbody" or "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`.

B.4 Migrating to new versions of DITA

DITA 1.3 is compatible with prior versions of the DITA specification; all valid DITA 1.0, 1.1, and 1.2 documents are valid DITA 1.3 documents. However, some changes to existing document-type shells and specializations

might be needed in order to maintain the same behavior under DITA 1.3 or to take full advantage of new DITA 1.3 features.

B.4.1 Migrating from DITA 1.2 to 1.3

DITA 1.3 defines two new domains for describing learning and training interactions: learningInteractionBase2Domain and learning2Domain. These two new domains mirror the DITA 1.2 learningInteractionBase and general learning domains, but the new versions provide support for multi-paragraph questions, answers, and feedback. The new domains are otherwise semantically identical to the original domains.

For backwards compatibility, both interactions domains are supported in DITA 1.3, and the document type shells provided with DITA 1.3 include both. In order to use the new elements, existing non-OASIS document type shells need to be updated to reference the new domain. See the OASIS provided DTD, RNG, or XSD shells for learningContent for examples of how to integrate these domains, or follow your normal process for integrating a new domain.

To migrate to the Learning 2 Domain, element tags in existing learning documents need to be transformed to the new names. This is done by adding a 2 to each element name in DITA documents that use the old domain. Once the element names have been changed, it is safe to remove the old domain from your shells. The complete list of elements to migrate is as follows:

DITA 1.2 element name	DITA 1.3 element name
lcAnswerContent	lcAnswerContent2
lcAnswerOption	lcAnswerOption2
lcAnswerOptionGroup	lcAnswerOptionGroup2
lcArea	lcArea2
lcAreaCoords	lcAreaCoords2
lcAreaShape	lcAreaShape2
lcAsset	lcAsset2
lcCorrectResponse	lcCorrectResponse2
lcFeedback	lcFeedback2
lcFeedbackCorrect	lcFeedbackCorrect2
lcFeedbackIncorrect	lcFeedbackIncorrect2
lcHotspot	lcHotspot2
lcHotspotMap	lcHotspotMap2
lcInstructornote	lcInstructornote2
lcInteractionBase	lcInteractionBase2
lcInteractionLabel	lcInteractionLabel2
lcItem	lcItem2
lcMatching	lcMatching2
lcMatchingHeader	lcMatchingHeader2

DITA 1.2 element name	DITA 1.3 element name
lcMatchingItem	lcMatchingItem2
lcMatchingItemFeedback	lcMatchingItemFeedback2
lcMatchingPair	lcMatchingPair2
lcMatchTable	lcMatchTable2
lcMultipleSelect	lcMultipleSelect2
lcOpenAnswer	lcOpenAnswer2
lcOpenQuestion	lcOpenQuestion2
lcQuestion	lcQuestion2
lcQuestionBase	lcQuestionBase2
lcSequence	lcSequence2
lcSequenceOption	lcSequenceOption2
lcSequenceOptionGroup	lcSequenceOptionGroup2
lcSequencing	lcSequencing2
lcSingleSelect	lcSingleSelect2
lcTrueFalse	lcTrueFalse2

B.4.2 Migrating from DITA 1.1 to 1.2

DITA 1.2 is compatible with prior versions of the DITA specification in that all valid DITA 1.0 and 1.1 documents are valid DITA 1.2 documents. However, some changes to existing document type shells and specializations might be needed in order to maintain the same behavior under DITA 1.2 or to take full advantage of new DITA 1.2 features.

Migration of task specializations

DITA 1.2 relaxes the content model for the `<taskbody>` element in the base `<task>` topic type. Specializations of `<task>` that use the `<taskbody>` element unchanged or use the content-model parameter entities for task body will use the more general DITA 1.2 task model when used with the DITA 1.2 task module.

If a document should use the strict task model instead of the new general task model:

- Documents that use the task topic type included with the OASIS DITA 1.2 specification **will not** need to migrate. This includes documents that use task inside of the ditabase document type included with the DITA 1.2 specification.
- Documents that use the standard task module inside of a new document type shell, or in a local ditabase document type, **will** need to migrate in order to preserve the module.
- Documents that use a specialization of task with a specialized element to replace `<taskbody>` **will not** need to migrate. Documents that specialize task but reuse the `<taskbody>` element **will** need to migrate.

For documents that do not use the standard OASIS DITA doctypes for which the strict task model from DITA 1.0 and 1.1 is desired, the strict task model can be restored by using the strict Taskbody Constraint module, by creating and integrating a constraint module similar to the strict Taskbody Constraint, or by adjusting the specialization's content models to impose the appropriate constraints.

To integrate the strict Taskbody Constraint module in a DTD, add the following content immediately before the section that imports the "topic.mod" file. The `strictTaskbodyConstraint.mod` file must be defined in any catalog used to locate DTD modules. See the `task.dtd` file in the OASIS DITA 1.2 DTD package for a sample:

```
<!ENTITY % strictTaskbody-c-def
  PUBLIC "-//OASIS//ELEMENTS DITA 1.2 Strict Taskbody Constraint//EN"
  "strictTaskbodyConstraint.mod">
%strictTaskbody-c-def;
```

To do the same thing for a schema, the `strictTaskbodyConstraintMod.xsd` file must be defined in the catalog used to resolve the schema using the following URN. The `<xs:include>` element should be placed immediately before the `<xs:include>` for `topicMod.xsd`:

```
<xs:include schemaLocation="urn:oasis:names:tc:dita:xsd:strictTaskbodyConstraintMod.xsd:1.2" />
```

Documents which use the task document type from OASIS and wish to make use of the general `<taskbody>` model must also be migrated. In this case, the documents must be updated to reference the General Task document type instead of the traditional Task document type. For DTD-based documents, this means that the document type declaration must change as follows within each task document:

```
DITA 1.1:
<!DOCTYPE task PUBLIC "-//OASIS//DTD DITA Task//EN"
 "task.dtd">

DITA 1.2:
<!DOCTYPE task PUBLIC "-//OASIS//DTD DITA General Task//EN"
 "generalTask.dtd">
```

For Schema based documents, the xsi:noNamespaceSchemaLocation attribute must change on each task element:

```
DITA 1.1:
<task xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="urn:oasis:names:tc:dita:xsd:task.xsd"

DITA 1.2:
<task xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="urn:oasis:names:tc:dita:xsd:generalTask.xsd"
```

Use of glossary document type shell and module files

The files named `glossary.dtd`, `glossary.mod`, and `glossary.ent` in DITA 1.1 have been renamed to `glossentry.dtd`, `glossentry.mod`, and `glossentry.ent` in order to follow the file naming requirements for vocabulary modules. The XML Schema files `glossary.xsd`, `glossaryGrp.xsd`, and `glossaryMod.xsd` have been renamed to `glossentry.xsd`, `glossentryGrp.xsd`, and `glossentryMod.xsd`.The DITA 1.2 entity resolution catalogs map the public identifiers for the `glossary.*` files to the corresponding `glossentry.*` files and URNs for the `glossary*.xsd` files to the corresponding `glossentry*.xsd` files, meaning that existing documents and shell document types that refer to the `glossary.*` files will continue to work with DITA 1.2.

However, shell document types that refer to the `glossary.*` files should be updated to refer to the corresponding `glossentry.*` files as it is convenient to do so.

Migration of specializations to reflect new implementation patterns

DITA 1.2 defines new implementation patterns for specializations. It is not necessary to update existing specialization modules to reflect these new patterns. However, specializations that want to enable the use of constraint modules must be updated to provide per-element-type content model and attribute list parameter entities.

Existing specializations might benefit from the new DITA 1.2 ability to have structural modules that depend on and specialize elements from domains.

Update of domains attributes for specializations

In DITA 1.1 the `@domains` attribute only listed the domains on which a given map or topic type depended. In DITA 1.2 the `@domains` attribute should also list structural modules and constraint modules, providing a complete list of the vocabulary modules that a given map or topic type depends on in the context of a given DITA document type.

Existing structural specializations should be updated to include the structural module names in their domains attribute configuration entities. Having a complete declaration of all structural and domains modules in the `@domains` attribute allows processors to determine the set of required modules for a given topic or map document without also examining all the `@class` attributes of all the elements in the document.

XML Schema declaration of domains attribute

For XML schema shell document types, the way the `@domains` attribute is declared has changed. In DITA 1.1 the `@domains` attribute was defined using an `<xs:redefine>` element. For DITA 1.2 the attribute is defined in document type shells and referenced from modules, like so:

```
<xs:attributeGroup name="domains-att">
    <xs:attribute name="domains" type="xs:string"
     default="(topic ui-d)
             (topic hi-d)
             (topic sw-d)
             (topic pr-d)
             (topic ut-d)
             (topic indexing-d)
             (topic hazard-d)
             (topic abbrev-d)"
    />
</xs:attributeGroup>
```

XML Schema document type shells must be updated to use the new declaration mechanism in order to continue to be valid XSD schemas.

B.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 MathML domain that ships with DITA 1.3. 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-about>X plus three</data-about>
</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-about>` element will remain:

```
<foreign class="+ topic/foreign mathml-d/mathml ">
  <object data="dita-generalized-topicid_mathml1.xml" type="DITA-foreign"/>
  <data-about>X plus three</data-about>
</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-about>X plus three</data-about>
  <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-about>X plus three</data-about>
  <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>
```

B.6 Element-by-element recommendations for translators: Base edition

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.

---

[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 lookup string to find the translation outside of the file, using accepted localization methods for generated text.

topic elements

Element name	Specialized from	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
`<abstract>`	N/A	block	block	yes	
`<alt>` [5]	N/A	block***[1]	block	yes	
`<audience>`	N/A	block (metadata)	block	yes	
`<author>`	N/A	block (metadata)	block	yes	
`<body>`	N/A	block	block	yes	
`<bodydiv>` *(new in DITA 1.2)*	N/A	block	block	yes	
`<boolean>`	N/A	inline	inline	n/a	
`<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)	
`<data-about>`	N/A	N/A (metadata)	block	no	
`<dd>`	N/A	block	block	yes	
`<ddhd>`	N/A	block	block	yes	
`<desc>`	N/A	block	block	yes	
`<div>` *(new in DITA 1.3)*	N/A	block	block	yes	

---

[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.

[5] The use of the `@alt` attribute is deprecated in favor of the `<alt>` element.

[6] 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?
`<dl>`	N/A	block	block	yes	`@spectitle`[2]
`<dlentry>`	N/A	block	block	yes	
`<dlhead>`	N/A	block	block	yes	
`<draft-comment>`	N/A	block***[1]	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	`@spectitle`[2]
`<featnum>`	N/A	block (metadata)	block	yes	
`<fig>`	N/A	block	block	yes	`@spectitle`[2]
`<figgroup>`	N/A	block	block	yes	
`<fn>`	N/A	block***[1]	block	yes	
`<foreign>` [7]	N/A	block[3]	block[3]	no[4]	
`<image>`	N/A	block when `@placement`=break, otherwise inline	block when `@placement`=break, otherwise inline	yes	`@alt`[5]
`<index-base>`	N/A	block***[1]	block	yes	
`<index-sort-as>` *(new in DITA 1.2)*	N/A	block***[1]	block	yes	
`<indexterm>`	N/A	block***[1]	block	yes	
`<indextermref>`	N/A	inline	n/a	n/a	
`<itemgroup>`	N/A	inline	inline	yes	
`<keyword>`	N/A	inline	inline (except when within `<keywords>` – see note above the table)	yes	
`<keywords>`	N/A	block	block	yes	
`<li>`	N/A	block	block	yes	
`<lines>`	N/A	block	block	yes	`@spectitle`[2]
`<link>`	N/A	block	block	yes	
`<linkinfo>`	N/A	block	block	yes	
`<linklist>`	N/A	block	block	yes	`@spectitle`[2]

---

[7] 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?
`<linkpool>`	N/A	block	block	yes	
`<linktext>`	N/A	block	block	yes	
`<lq>`	N/A	block	block	yes	`@reftitle`
`<metadata>`	N/A	block (metadata)	block	yes	
`<navtitle>`	N/A	block	block	yes	
`<no-topic-nesting>`	N/A	n/a	n/a	n/a	
`<note>`	N/A	block	block	yes	`@othertype`, `@spectitle`[2]
`<object>`	N/A	block	block	yes	`@standby`
`<ol>`	N/A	block	block	yes	`@spectitle`[2]
`<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	`@spectitle`[2]
`<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***[1]	block	**no**	
`<resourceid>`	N/A	block (metadata)	block	yes	
`<revised>`	N/A	block (metadata)	block	yes	
`<row>`	N/A	block	block	yes	
`<searchtitle>`	N/A	block	block	yes	
`<section>`	N/A	block	block	yes	`@spectitle`[2]

Element name	Specialized from	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
`<sectiondiv>` *(new in DITA 1.2)*	N/A	block	block	yes	
`<series>`	N/A	block (metadata)	block	yes	
`<shortdesc>`	N/A	block	block	yes	
`<simpletable>`	N/A	block	block	yes	`@spectitle`[2]
`<sl>`	N/A	block	block	yes	`@spectitle`[2]
`<sli>`	N/A	block	block	yes	
`<source>`	N/A	block (metadata)	block	yes	
`<state>`	N/A	inline	inline	yes	`@value`
`<stentry>`	N/A	block	block	yes	`@specentry`[2]
`<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>` *(new in DITA 1.2)*	N/A	inline	inline	yes	
`<tgroup>`	N/A	block	block	yes	
`<thead>`	N/A	block	block	yes	
`<title>`	N/A	block	block	yes	
`<titlealts>`	N/A	block	block	yes	
`<tm>`	N/A	inline	inline	yes	
`<topic>`	N/A	block	block	yes	
`<ul>`	N/A	block	block	yes	`@spectitle`[2]
`<unknown>` *(new in DITA 1.1)*	N/A	block	block	no	
`<vrm>`	N/A	block (metadata)	block	yes	
`<vrmlist>`	N/A	block (metadata)	block	yes	
`<xref>`	N/A	inline	inline	yes	

map elements

As of DITA 1.2, the `@navtitle` attribute is deprecated, for translation purposes, in favor of the new `<navtitle>` element. The `<navtitle>` element is also available in topics, and is included in the topic table above.

Element name	Specialized from	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
`<anchor>`	N/A	n/a	n/a	n/a	
`<linktext>`	N/A	block	block	yes	
`<map>`	N/A	block	block	yes	`@title` [8]
`<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	
`<searchtitle>`	N/A	block	block	yes	
`<shortdesc>`	N/A	block	block	yes	
`<topicmeta>`	N/A	block	block	yes	
`<topicref>`	N/A	block	block	yes	`@navtitle`
`<ux-window>`	N/A	N/A (empty)	N/A (empty)	no	

hi-d elements (highlight domain)

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>` *(new in DITA 1.3)*		yes	inline	inline	yes	
`<i>`	`<ph>`	yes	inline	inline	yes	
`<overline>` *(new in DITA 1.3)*	`<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	

---

[8] (deprecated as of DITA 1.2 for translation purposes)

indexing-d elements (indexing domain) *(new in DITA 1.1)*

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
`<index-see>`	`<index-base>`	yes	block***[1]	block	yes	
`<index-see-also>`	`<index-base>`	yes	block***[1]	block	yes	
`<index-sort-as>`	`<index-base>`	yes	block***[1]	block	yes	

ut-d elements (Utilities domain)

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`[2]
`<shape>`	`<keyword>`	**NO**	inline	inline	**no**	
`<sort-as>` *(new in DITA 1.3)*	`<data>`	**NO**	block***[1]	block	yes	

mapgroup-d elements (mapgroup domain)

As of DITA 1.2, the `@navtitle` attribute is deprecated, for translation purposes, in favor of the new `<navtitle>` element.

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
`<anchorref>` *(new in DITA 1.2)*	`<topicref>`	yes	block	block	yes	`@navtitle`
`<keydef>` *(new in DITA 1.2)*	`<topicref>`	yes	block	block	yes	`@navtitle`
`<mapref>` *(new in DITA 1.2)*	`<topicref>`	yes	block	block	yes	`@navtitle`
`<topicgroup>`	`<topicref>`	yes	block	block	yes	`@navtitle`

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
`<topichead>`	`<topicref>`	yes	block	block	yes	`@navtitle`
`<topicset>` *(new in DITA 1.2)*	`<topicref>`	yes	block	block	yes	`@navtitle`
`<topicsetref>` *(new in DITA 1.2)*	`<topicref>`	yes	block	block	yes	`@navtitle`

ditavalref-d elements (ditavalref domain) *(new in DITA 1.3)*

The `@navtitle` attribute is deprecated, for translation purposes, in favor of the `<navtitle>` element.

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	`@navtitle`
`<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	

Delayed Conref Resolution domain elements *(new in DITA 1.2)*

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
`<anchorid>`	`<keyword>`	**no**	N/A (metadata)	**n/a**	**no**	
`<anchorkey>`	`<keyword>`	**no**	N/A (metadata)	**n/a**	**no**	
`<exportanchors>`	`<keywords>`	**no**	N/A (metadata)	**n/a**	**no**	

Hazard Statement Domain *(new in DITA 1.2)*

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
`<consequence>`	`<li>`	yes	block	block	yes	
`<hazardstatement>`	`<note>`	yes	block	block	yes	`@othertype`, `@spectitle`[2]

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
`<hazardsymbol>`	`<image>`	yes	block when `@placement`=break, otherwise inline	block when `@placement`=break, otherwise inline	yes	*removes* `@alt`
`<howtoavoid>`	`<li>`	yes	block	block	yes	
`<messagepanel>`	`<ul>`	yes	block	block	yes	`@spectitle`[2]
`<typeofhazard>`	`<li>`	yes	block	block	yes	

classification domain elements *(new in DITA 1.2)*

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	`@navtitle`
`<topicapply>`	`<topicref>`	yes	block	block	yes	`@navtitle`
`<topicCell>`	`<relcell>`	yes	block	block	yes	
`<topicsubject>`	`<topicref>`	yes	block	block	yes	`@navtitle`
`<topicSubjectHeader>`	`<relrow>`	yes	block	block	yes	
`<topicSubjectRow>`	`<relrow>`	yes	block	block	yes	
`<topicSubjectTable>`	`<reltable>`	yes	block	block	yes	

subject scheme elements *(new in DITA 1.2)*

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
`<attributedef>`	`<data>`	yes	N/A (metadata	block	no	
`<defaultSubject>`	`<topicref>`	yes	block	block	yes	`@navtitle`
`<elementdef>`	`<data>`	yes	N/A (metadata)	block	no	
`<enumerationdef>`	`<topicref>`	yes	block	block	yes	`@navtitle`
`<hasInstance>`	`<topicref>`	yes	block	block	yes	`@navtitle`
`<hasKind>`	`<topicref>`	yes	block	block	yes	`@navtitle`
`<hasNarrower>`	`<topicref>`	yes	block	block	yes	`@navtitle`

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
`<hasPart>`	`<topicref>`	yes	block	block	yes	`@navtitle`
`<hasRelated>`	`<topicref>`	yes	block	block	yes	`@navtitle`
`<relatedSubjects>`	`<topicref>`	yes	block	block	yes	`@navtitle`
`<subjectdef>`	`<topicref>`	yes	block	block	yes	`@navtitle`
`<subjectHead>`	`<topicref>`	yes	block	block	yes	`@navtitle`
`<subjectHeadMeta>`	`<topicmeta>`	yes	block	block	yes	
`<schemeref>`	`<topicref>`	yes	block	block	yes	`@navtitle`
`<subjectRel>`	`<relrow>`	yes	block	block	yes	
`<subjectRelHeader>`	`<relrow>`	yes	block	block	yes	
`<subjectRelTable>`	`<reltable>`	yes	block	block	yes	
`<subjectRole>`	`<relcell>`	yes	block	block	yes	
`<subjectScheme>`	`<map>`	yes	block	block	yes	*removes* `@title`

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.

As of DITA 1.2, 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)

B.7 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 version number (for example, 1.0, 1.1, 1.2, or 1.3) or is omitted entirely.

- *information-type* is the name of the topic or map type in camel case, 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.

B.8 XML Schema catalog identifiers

Each Schema document must be uniquely identified using a Uniform Resource Name (URN). Each Schema document has a version-independent as well as a version-specific URN.

An XML resolver should be used to uniquely identify an XML Schema in order to map to a local resource. The DITA specification includes both URN- and URL-based versions of the XML schemas. The URL versions are included as a convenience for use with tools that do not support catalog based resolution and are not normative.

The catalog identifiers use the following patterns.

Creator	Pattern
OASIS DITA Technical Committee	`urn:oasis:names:tc:dita:xsd:`*filename*`.xsd:`*version*
Subcommittee of the OASIS DITA Technical Committee	`urn:oasis:names:tc:dita:spec:`*subcomittee-specialization*`:xsd:`*filename*`.xsd:`*version*

where:

- *subcomittee-specialization* indicates the subcommittee and specialization, for example, "machinery" or "learning".

- *filename* is the base name of the document-type shell or module component.

- *version* either is a DITA version number (for example, 1.0, 1.1, 1.2, or 1.3) or is omitted entirely.

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. The pattern "urn:oasis:names:tc:dita" is a convention that indicates that the artifact is DITA-related and defined by the OASIS DITA Technical Committee.

B.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.

B.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 11: Base domains

Domain	Description	Short name
Classify (map)	For associating content in a map with subjects in a subject scheme.	classify-d

Domain	Description	Short name
Conref delayed resolution (map/topic)	For delaying resolution of `@conref` or `@keyref` on some elements.	delay-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 imagemaps, sort keys, and other useful structures.	ut-d

B.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
Conref delayed resolution (map/topic)	• 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
Utilities (map/topic)	• Base map, base topic	• N/A

B.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 12: Domain usage in base document types

Document type	Includes these domains	Does not include
base map	• `@deliveryTarget` attribute • Conref delayed resolution (map/topic) • 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
		  – Conref delayed resolution (map/topic)
		  – Ditavalref (map)
		  – Map group (map)

B.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* *(72)* 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 topicrefs 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.

B.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 undesireable 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 interoperation 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

<div>

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.

Appendix C Content models: Base edition

For each element in the base edition, this section presents the content model and a list of parent elements that can contain that element.

C.1 A elements

Content models for elements beginning with "a".

Content models for ** *(187)*

In this document type	`<abstract>` contains
Base topic	(text data | *<dl>* | *<div>* | *<fig>* | *<imagemap>* | *<image>* | *<lines>* | *<lq>* | *<note>* | *<hazardstatement>* | *<object>* | * * | *<p>* | *<pre>* | *<simpletable>* | *<sl>* | *<table>* | * * | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<shortdesc>* | *<draft-comment>* | *<fn>* | *<indextermref>* | *<indexterm>* | *<required-cleanup>*)*

In this document type	`<abstract>` is contained by
Base topic	*<topic>*

Content models for *<alt>* *(191)*

In this document type	`<alt>` contains
Base map, Base topic	(text data | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<keyword>* | *<term>* | *<text>* | *<draft-comment>* | *<required-cleanup>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>*)*

In this document type	`<alt>` is contained by
Base map, Base topic	*<hazardsymbol>*, *<image>*

Content models for *<anchor>* *(240)*

In this document type	`<anchor>` contains
Base map	EMPTY

In this document type	`<anchor>` is contained by
Base map	*<foreign>*, *<keydef>*, *<map>*, *<required-cleanup>*, *<topicgroup>*, *<topichead>*, *<topicref>*, *<topicset>*, *<unknown>*

Content models for *<anchorid>* *(283)*

In this document type	`<anchorid>` contains
Base map	EMPTY

In this document type	`<anchorid>` is contained by
Base map	*`<exportanchors>`*

Content models for *`<anchorkey>`* *(287)*

In this document type	`<anchorkey>` contains
Base map	EMPTY

In this document type	`<anchorkey>` is contained by
Base map	*`<exportanchors>`*

Content models for *`<anchorref>`* *(249)*

In this document type	`<anchorref>` contains
Base map	*`<topicmeta>`*?, (*`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<topicref>`* | *`<anchorref>`* | *`<keydef>`* | *`<mapref>`* | *`<topicgroup>`* | *`<topichead>`* | *`<topicset>`* | *`<topicsetref>`* | *`<ditavalref>`*)*

In this document type	`<anchorref>` is contained by
Base map	*`<anchorref>`*, *`<keydef>`*, *`<map>`*, *`<relcell>`*, *`<relcolspec>`*, *`<topicgroup>`*, *`<topichead>`*, *`<topicref>`*, *`<topicset>`*, *`<topicsetref>`*

Content models for *`<area>`* *(297)*

In this document type	`<area>` contains
Base map, Base topic	*`<shape>`*, *`<coords>`*, *`<xref>`*

In this document type	`<area>` is contained by
Base map, Base topic	*`<imagemap>`*

Content models for *`<attributedef>`* *(315)*

None of the document type shells distributed with the base package use this element.

Content models for *`<audience>`* *(256)*

In this document type	`<audience>` contains
Base map, Base topic	EMPTY

In this document type	`<audience>` is contained by
Base map	*`<metadata>`*, *`<topicmeta>`*
Base topic	*`<metadata>`*

Content models for *<author>* *(257)*

In this document type	`<author>` contains
Base map, Base topic	(text data | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<keyword>* | *<term>* | *<text>*)*

In this document type	`<author>` is contained by
Base map	*<topicmeta>*
Base topic	*<prolog>*

C.2 B elements

Content models for elements beginning with "b".

Content models for *<b>* *(293)*

In this document type	`<b>` contains
Base map, Base topic	(text data | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<draft-comment>* | *<foreign>* | *<unknown>* | *<required-cleanup>*)*

In this document type	`<b>` is contained by
Base map	*<alt>*, *<b>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*
Base topic	**, *<alt>*, *<b>*, *<bodydiv>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<searchtitle>*, *<section>*, *<sectiondiv>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*

Content models for *<body>* *(189)*

In this document type	`<body>` contains
Base topic	(*<dl>* | *<div>* | *<fig>* | *<imagemap>* | *<image>* | *<lines>* | *<lq>* | *<note>* | *<hazardstatement>* | *<object>* | * * | *<p>* | *<pre>* | *<simpletable>* | *<sl>* | *<table>* | * * | *<data>* | *<sort-as>* | *<data-about>* | *<draft-comment>* | *<foreign>* | *<unknown>* | *<required-cleanup>* | *<bodydiv>* | *<example>* | *<section>*)*

In this document type	`<body>` is contained by
Base topic	*`<topic>`*

Content models for *`<bodydiv>`* (189)

In this document type	`<bodydiv>` contains
Base topic	(text data | *`<dl>`* | *`<div>`* | *`<fig>`* | *`<imagemap>`* | *`<image>`* | *`<lines>`* | *`<lq>`* | *`<note>`* | *`<hazardstatement>`* | *`<object>`* | *`<ol>`* | *`<p>`* | *`<pre>`* | *`<simpletable>`* | *`<sl>`* | *`<table>`* | *`<ul>`* | *`<boolean>`* | *`<cite>`* | *`<keyword>`* | *`<ph>`* | *`<b>`* | *`<i>`* | *`<line-through>`* | *`<overline>`* | *`<sup>`* | *`<sub>`* | *`<tt>`* | *`<u>`* | *`<q>`* | *`<term>`* | *`<text>`* | *`<tm>`* | *`<xref>`* | *`<state>`* | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<draft-comment>`* | *`<fn>`* | *`<indextermref>`* | *`<indexterm>`* | *`<required-cleanup>`* | *`<bodydiv>`* | *`<section>`*)*

In this document type	`<bodydiv>` is contained by
Base topic	*`<body>`*, *`<bodydiv>`*

Content models for *`<boolean>`* (331)

In this document type	`<boolean>` contains
Base map, Base topic	EMPTY

In this document type	`<boolean>` is contained by
Base map	*`<b>`*, *`<cite>`*, *`<data>`*, *`<dd>`*, *`<ddhd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<dt>`*, *`<dthd>`*, *`<entry>`*, *`<figgroup>`*, *`<fn>`*, *`<howtoavoid>`*, *`<i>`*, *`<itemgroup>`*, *`<li>`*, *`<line-through>`*, *`<lines>`*, *`<lq>`*, *`<note>`*, *`<overline>`*, *`<p>`*, *`<ph>`*, *`<pre>`*, *`<q>`*, *`<shortdesc>`*, *`<sli>`*, *`<stentry>`*, *`<sub>`*, *`<sup>`*, *`<title>`*, *`<tt>`*, *`<u>`*, *`<xref>`*
Base topic	*`<abstract>`*, *`<b>`*, *`<bodydiv>`*, *`<cite>`*, *`<data>`*, *`<dd>`*, *`<ddhd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<dt>`*, *`<dthd>`*, *`<entry>`*, *`<example>`*, *`<figgroup>`*, *`<fn>`*, *`<howtoavoid>`*, *`<i>`*, *`<itemgroup>`*, *`<li>`*, *`<line-through>`*, *`<lines>`*, *`<linkinfo>`*, *`<lq>`*, *`<note>`*, *`<overline>`*, *`<p>`*, *`<ph>`*, *`<pre>`*, *`<q>`*, *`<section>`*, *`<sectiondiv>`*, *`<shortdesc>`*, *`<sli>`*, *`<stentry>`*, *`<sub>`*, *`<sup>`*, *`<title>`*, *`<tt>`*, *`<u>`*, *`<xref>`*

Content models for *`<brand>`* (258)

In this document type	`<brand>` contains
Base map, Base topic	(text data | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<keyword>`* | *`<term>`* | *`<text>`*)*

In this document type	`<brand>` is contained by
Base map, Base topic	*`<prodinfo>`*

C.3 C elements

Content models for elements beginning with "c".

Content models for *<category>* (258)

In this document type	<category> contains
Base map, Base topic	(text data | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<keyword>* | *<term>* | *<text>*)*

In this document type	<category> is contained by
Base map	*<metadata>*, *<topicmeta>*
Base topic	*<metadata>*

Content models for *<cite>* (192)

In this document type	<cite> contains
Base map, Base topic	(text data | *<boolean>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<draft-comment>* | *<foreign>* | *<unknown>* | *<required-cleanup>*)*

In this document type	<cite> is contained by
Base map	*<b>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<lq>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<shortdesc>*, *<sli>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<u>*, *<xref>*
Base topic	**, *<b>*, *<bodydiv>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<lq>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<section>*, *<sectiondiv>*, *<shortdesc>*, *<sli>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<u>*, *<xref>*

Content models for *<colspec>* (227)

In this document type	<colspec> contains
Base map, Base topic	EMPTY

In this document type	<colspec> is contained by
Base map, Base topic	*<tgroup>*

Content models for *<component>* (259)

In this document type	<component> contains
Base map, Base topic	(text data | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<keyword>* | *<term>* | *<text>*)*

In this document type	`<component>` is contained by
Base map, Base topic	*`<prodinfo>`*

Content models for *`<consequence>`* *(290)*

In this document type	`<consequence>` contains
Base map, Base topic	(text data | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<keyword>`* | *`<term>`* | *`<text>`* | *`<ph>`* | *`<b>`* | *`<i>`* | *`<line-through>`* | *`<overline>`* | *`<sup>`* | *`<sub>`* | *`<tt>`* | *`<u>`* | *`<tm>`*)*

In this document type	`<consequence>` is contained by
Base map, Base topic	*`<messagepanel>`*

Content models for *`<coords>`* *(297)*

In this document type	`<coords>` contains
Base map, Base topic	(text data | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<keyword>`* | *`<term>`* | *`<text>`*)*

In this document type	`<coords>` is contained by
Base map, Base topic	*`<area>`*

Content models for *`<copyrholder>`* *(260)*

In this document type	`<copyrholder>` contains
Base map, Base topic	(text data | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<keyword>`* | *`<term>`* | *`<text>`*)*

In this document type	`<copyrholder>` is contained by
Base map, Base topic	*`<copyright>`*

Content models for *`<copyright>`* *(259)*

In this document type	`<copyright>` contains
Base map, Base topic	*`<copyryear>`*+, *`<copyrholder>`*

In this document type	`<copyright>` is contained by
Base map	*`<topicmeta>`*
Base topic	*`<prolog>`*

Content models for *`<copyryear>`* *(261)*

In this document type	`<copyryear>` contains
Base map, Base topic	EMPTY

In this document type	`<copyryear>` is contained by
Base map, Base topic	*`<copyright>`*

Content models for *`<created>`* (261)

In this document type	`<created>` contains
Base map, Base topic	EMPTY

In this document type	`<created>` is contained by
Base map, Base topic	*`<critdates>`*

Content models for *`<critdates>`* (261)

In this document type	`<critdates>` contains
Base map, Base topic	*`<created>`*?, *`<revised>`**

In this document type	`<critdates>` is contained by
Base map	*`<topicmeta>`*
Base topic	*`<prolog>`*

C.4 D elements

Content models for elements beginning with "d".

Content models for *`<data>`* (331)

In this document type	`<data>` contains
Base map, Base topic	(text data | *`<boolean>`* | *`<cite>`* | *`<keyword>`* | *`<ph>`* | *`<b>`* | *`<i>`* | *`<line-through>`* | *`<overline>`* | *`<sup>`* | *`<sub>`* | *`<tt>`* | *`<u>`* | *`<q>`* | *`<term>`* | *`<text>`* | *`<tm>`* | *`<xref>`* | *`<state>`* | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<draft-comment>`* | *`<foreign>`* | *`<unknown>`* | *`<image>`* | *`<object>`* | *`<required-cleanup>`* | *`<title>`*)*

In this document type	`<data>` is contained by
Base map	*`<alt>`*, *`<anchorref>`*, *`<author>`*, *`<b>`*, *`<brand>`*, *`<category>`*, *`<cite>`*, *`<component>`*, *`<consequence>`*, *`<coords>`*, *`<copyrholder>`*, *`<data>`*, *`<data-about>`*, *`<dd>`*, *`<ddhd>`*, *`<desc>`*, *`<div>`*, *`<dl>`*, *`<draft-comment>`*, *`<dt>`*, *`<dthd>`*, *`<entry>`*, *`<featnum>`*, *`<fig>`*, *`<figgroup>`*, *`<fn>`*, *`<i>`*, *`<index-base>`*, *`<index-see>`*, *`<index-see-also>`*, *`<index-sort-as>`*, *`<indexterm>`*, *`<itemgroup>`*, *`<keydef>`*, *`<li>`*, *`<line-through>`*, *`<lines>`*, *`<linktext>`*, *`<lq>`*, *`<map>`*, *`<mapref>`*, *`<messagepanel>`*, *`<metadata>`*, *`<navtitle>`*, *`<note>`*, *`<ol>`*, *`<overline>`*, *`<p>`*, *`<ph>`*, *`<platform>`*, *`<pre>`*, *`<prodname>`*, *`<prognum>`*, *`<publisher>`*, *`<q>`*, *`<relcell>`*, *`<resourceid>`*, *`<searchtitle>`*, *`<series>`*, *`<shortdesc>`*, *`<sl>`*, *`<sli>`*, *`<source>`*, *`<stentry>`*, *`<sub>`*, *`<sup>`*, *`<title>`*, *`<topicgroup>`*, *`<topichead>`*, *`<topicmeta>`*, *`<topicref>`*, *`<topicset>`*, *`<topicsetref>`*, *`<tt>`*, *`<typeofhazard>`*, *`<u>`*, *`<ul>`*, *`<xref>`*
Base topic	*`<abstract>`*, *`<alt>`*, *`<author>`*, *`<b>`*, *`<body>`*, *`<bodydiv>`*, *`<brand>`*, *`<category>`*, *`<cite>`*, *`<component>`*, *`<consequence>`*, *`<coords>`*, *`<copyrholder>`*, *`<data>`*, *`<data-*

In this document type	`<data>` is contained by
	about, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<dl>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<featnum>*, *<fig>*, *<figgroup>*, *<fn>*, *<i>*, *<index-base>*, *<index-see>*, *<index-see-also>*, *<index-sort-as>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<linktext>*, *<lq>*, *<messagepanel>*, *<metadata>*, *<navtitle>*, *<note>*, * *, *<overline>*, *<p>*, *<ph>*, *<platform>*, *<pre>*, *<prodname>*, *<prognum>*, *<prolog>*, *<publisher>*, *<q>*, *<resourceid>*, *<searchtitle>*, *<section>*, *<sectiondiv>*, *<series>*, *<shortdesc>*, *<sl>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, * *, *<xref>*

Content models for *<data-about>* (333)

In this document type	`<data-about>` contains
Base map, Base topic	*<data>*, *<sort-as>*, (*<data>* | *<sort-as>* | *<data-about>*)*

In this document type	`<data-about>` is contained by
Base map	*<alt>*, *<anchorref>*, *<author>*, *<b>*, *<brand>*, *<category>*, *<cite>*, *<component>*, *<consequence>*, *<coords>*, *<copyrholder>*, *<data>*, *<data-about>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<dl>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<featnum>*, *<fig>*, *<figgroup>*, *<fn>*, *<i>*, *<index-base>*, *<index-see>*, *<index-see-also>*, *<index-sort-as>*, *<indexterm>*, *<itemgroup>*, *<keydef>*, * *, *<line-through>*, *<lines>*, *<linktext>*, *<lq>*, *<map>*, *<mapref>*, *<messagepanel>*, *<metadata>*, *<navtitle>*, *<note>*, * *, *<overline>*, *<p>*, *<ph>*, *<platform>*, *<pre>*, *<prodname>*, *<prognum>*, *<publisher>*, *<q>*, *<relcell>*, *<resourceid>*, *<searchtitle>*, *<series>*, *<shortdesc>*, *<sl>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<topicgroup>*, *<topichead>*, *<topicmeta>*, *<topicref>*, *<topicset>*, *<topicsetref>*, *<tt>*, *<typeofhazard>*, *<u>*, * *, *<xref>*
Base topic	**, *<alt>*, *<author>*, *<b>*, *<body>*, *<bodydiv>*, *<brand>*, *<category>*, *<cite>*, *<component>*, *<consequence>*, *<coords>*, *<copyrholder>*, *<data>*, *<data-about>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<dl>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<featnum>*, *<fig>*, *<figgroup>*, *<fn>*, *<i>*, *<index-base>*, *<index-see>*, *<index-see-also>*, *<index-sort-as>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<linktext>*, *<lq>*, *<messagepanel>*, *<metadata>*, *<navtitle>*, *<note>*, * *, *<overline>*, *<p>*, *<ph>*, *<platform>*, *<pre>*, *<prodname>*, *<prognum>*, *<prolog>*, *<publisher>*, *<q>*, *<resourceid>*, *<searchtitle>*, *<section>*, *<sectiondiv>*, *<series>*, *<shortdesc>*, *<sl>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, * *, *<xref>*

Content models for *<dd>* (192)

In this document type	`<dd>` contains
Base map, Base topic	(text data | *<dl>* | *<div>* | *<fig>* | *<imagemap>* | *<image>* | *<lines>* | *<lq>* | *<note>* | *<hazardstatement>* | *<object>* | * * | *<p>* | *<pre>* | *<simpletable>* | *<sl>* | *<table>* | * * | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* |

In this document type	`<dd>` contains
	`<itemgroup>` | `<draft-comment>` | `<fn>` | `<indextermref>` | `<indexterm>` | `<required-cleanup>`)*

In this document type	`<dd>` is contained by
Base map, Base topic	`<dlentry>`

Content models for *<ddhd>* (193)

In this document type	`<ddhd>` contains
Base map, Base topic	(text data | `<boolean>` | `<keyword>` | `<ph>` | `<b>` | `<i>` | `<line-through>` | `<overline>` | `<sup>` | `<sub>` | `<tt>` | `<u>` | `<q>` | `<term>` | `<text>` | `<tm>` | `<state>` | `<cite>` | `<data>` | `<sort-as>` | `<data-about>` | `<draft-comment>` | `<foreign>` | `<unknown>` | `<required-cleanup>` | `<image>`)*

In this document type	`<ddhd>` is contained by
Base map, Base topic	`<dlhead>`

Content models for *<defaultSubject>* (317)

None of the document type shells distributed with the base package use this element.

Content models for *<desc>* (193)

In this document type	`<desc>` contains
Base map, Base topic	(text data | `<dl>` | `<div>` | `<image>` | `<lines>` | `<lq>` | `<note>` | `<hazardstatement>` | `<ol>` | `<p>` | `<pre>` | `<sl>` | `<ul>` | `<boolean>` | `<cite>` | `<keyword>` | `<ph>` | `<b>` | `<i>` | `<line-through>` | `<overline>` | `<sup>` | `<sub>` | `<tt>` | `<u>` | `<q>` | `<term>` | `<text>` | `<tm>` | `<xref>` | `<state>` | `<data>` | `<sort-as>` | `<data-about>` | `<draft-comment>` | `<foreign>` | `<unknown>` | `<required-cleanup>`)*

In this document type	`<desc>` is contained by
Base map	`<fig>`, `<object>`, `<table>`, `<xref>`
Base topic	`<fig>`, `<link>`, `<linklist>`, `<object>`, `<table>`, `<xref>`

Content models for *<dita>* (191)

None of the document type shells distributed with the base package use this element.

Content models for *<ditavalmeta>* (304)

In this document type	`<ditavalmeta>` contains
Base map	(`<navtitle>`?, (`<dvrResourcePrefix>`?, `<dvrResourceSuffix>`?, `<dvrKeyscopePrefix>`?, `<dvrKeyscopeSuffix>`?`<dvrResourceSuffix>`?, `<dvrKeyscopePrefix>`?, `<dvrKeyscopeSuffix>`?`<dvrKeyscopePrefix>`?, `<dvrKeyscopeSuffix>`?`<dvrKeyscopeSuffix>`?)(`<dvrResourcePrefix>`?, `<dvrResourceSuffix>`?, `<dvrKeyscopePrefix>`?, `<dvrKeyscopeSuffix>`?

In this document type	**`<ditavalmeta>` contains**
	`<dvrResourceSuffix>`?, *`<dvrKeyscopePrefix>`*?, *`<dvrKeyscopeSuffix>`*? *`<dvrKeyscopePrefix>`*?, *`<dvrKeyscopeSuffix>`*?*`<dvrKeyscopeSuffix>`*?))

In this document type	**`<ditavalmeta>` is contained by**
Base map	*`<ditavalref>`*

Content models for *`<ditavalref>`* (302)

In this document type	**`<ditavalref>` contains**
Base map	(*`<ditavalmeta>`*)*

In this document type	**`<ditavalref>` is contained by**
Base map	*`<anchorref>`*, *`<keydef>`*, *`<map>`*, *`<relcell>`*, *`<relcolspec>`*, *`<topicgroup>`*, *`<topichead>`*, *`<topicref>`*, *`<topicset>`*, *`<topicsetref>`*

Content models for *`<div>`* (194)

In this document type	**`<div>` contains**
Base map, Base topic	(text data | *`<dl>`* | *`<div>`* | *`<fig>`* | *`<imagemap>`* | *`<image>`* | *`<lines>`* | *`<lq>`* | *`<note>`* | *`<hazardstatement>`* | *`<object>`* | *`<ol>`* | *`<p>`* | *`<pre>`* | *`<simpletable>`* | *`<sl>`* | *`<table>`* | *`<ul>`* | *`<boolean>`* | *`<cite>`* | *`<keyword>`* | *`<ph>`* | *`<b>`* | *`<i>`* | *`<line-through>`* | *`<overline>`* | *`<sup>`* | *`<sub>`* | *`<tt>`* | *`<u>`* | *`<q>`* | *`<term>`* | *`<text>`* | *`<tm>`* | *`<xref>`* | *`<state>`* | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<draft-comment>`* | *`<fn>`* | *`<indextermref>`* | *`<indexterm>`* | *`<required-cleanup>`*)*

In this document type	**`<div>` is contained by**
Base map	*`<dd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<entry>`*, *`<fig>`*, *`<figgroup>`*, *`<fn>`*, *`<itemgroup>`*, *`<li>`*, *`<lq>`*, *`<note>`*, *`<p>`*, *`<stentry>`*
Base topic	*`<abstract>`*, *`<body>`*, *`<bodydiv>`*, *`<dd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<entry>`*, *`<example>`*, *`<fig>`*, *`<figgroup>`*, *`<fn>`*, *`<itemgroup>`*, *`<li>`*, *`<linkinfo>`*, *`<lq>`*, *`<note>`*, *`<p>`*, *`<section>`*, *`<sectiondiv>`*, *`<stentry>`*

Content models for *`<dl>`* (195)

In this document type	**`<dl>` contains**
Base map, Base topic	(*`<data>`* | *`<sort-as>`* | *`<data-about>`*)*, *`<dlhead>`*?, *`<dlentry>`*+

In this document type	**`<dl>` is contained by**
Base map	*`<dd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<entry>`*, *`<fig>`*, *`<figgroup>`*, *`<fn>`*, *`<itemgroup>`*, *`<li>`*, *`<lq>`*, *`<note>`*, *`<p>`*, *`<stentry>`*
Base topic	*`<abstract>`*, *`<body>`*, *`<bodydiv>`*, *`<dd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<entry>`*, *`<example>`*, *`<fig>`*, *`<figgroup>`*, *`<fn>`*, *`<itemgroup>`*, *`<li>`*, *`<linkinfo>`*, *`<lq>`*, *`<note>`*, *`<p>`*, *`<section>`*, *`<sectiondiv>`*, *`<stentry>`*

Content models for *<dlentry>* (196)

In this document type	<dlentry> contains
Base map, Base topic	*<dt>*+, *<dd>*+

In this document type	<dlentry> is contained by
Base map, Base topic	*<dl>*

Content models for *<dlhead>* (197)

In this document type	<dlhead> contains
Base map, Base topic	*<dthd>*?, *<ddhd>*?

In this document type	<dlhead> is contained by
Base map, Base topic	*<dl>*

Content models for *<draft-comment>* (198)

In this document type	<draft-comment> contains
Base map, Base topic	(text data | *<dl>* | *<div>* | *<fig>* | *<imagemap>* | *<image>* | *<lines>* | *<lq>* | *<note>* | *<hazardstatement>* | *<object>* | * * | *<p>* | *<pre>* | *<simpletable>* | *<sl>* | *<table>* | * * | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>*)*

In this document type	<draft-comment> is contained by
Base map	*<alt>*, *<b>*, *<cite>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<dt>*, *<dthd>*, *<entry>*, *<figgroup>*, *<fn>*, *<i>*, *<itemgroup>*, *<keyword>*, * *, *<line-through>*, *<lines>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<shortdesc>*, *<sli>*, *<stentry>*, *<sub>*, *<sup>*, *<term>*, *<title>*, *<tt>*, *<u>*, *<xref>*
Base topic	**, *<alt>*, *<b>*, *<body>*, *<bodydiv>*, *<cite>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<figgroup>*, *<fn>*, *<i>*, *<itemgroup>*, *<keyword>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<section>*, *<sectiondiv>*, *<shortdesc>*, *<sli>*, *<stentry>*, *<sub>*, *<sup>*, *<term>*, *<title>*, *<tt>*, *<u>*, *<xref>*

Content models for *<dt>* (197)

In this document type	<dt> contains
Base map, Base topic	(text data | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<draft-comment>* | *<foreign>* | *<unknown>* | *<required-cleanup>* | *<image>*)*

In this document type	`<dt>` is contained by
Base map, Base topic	*`<dlentry>`*

Content models for *<dthd>* (199)

In this document type	`<dthd>` contains
Base map, Base topic	(text data | *`<boolean>`* | *`<keyword>`* | *`<ph>`* | *`<b>`* | *`<i>`* | *`<line-through>`* | *`<overline>`* | *`<sup>`* | *`<sub>`* | *`<tt>`* | *`<u>`* | *`<q>`* | *`<term>`* | *`<text>`* | *`<tm>`* | *`<state>`* | *`<cite>`* | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<draft-comment>`* | *`<foreign>`* | *`<unknown>`* | *`<required-cleanup>`* | *`<image>`*)*

In this document type	`<dthd>` is contained by
Base map, Base topic	*`<dlhead>`*

Content models for *<dvrKeyscopePrefix>* (307)

In this document type	`<dvrKeyscopePrefix>` contains
Base map	(text data | *`<text>`*)*

In this document type	`<dvrKeyscopePrefix>` is contained by
Base map	*`<ditavalmeta>`*

Content models for *<dvrKeyscopeSuffix>* (307)

In this document type	`<dvrKeyscopeSuffix>` contains
Base map	(text data | *`<text>`*)*

In this document type	`<dvrKeyscopeSuffix>` is contained by
Base map	*`<ditavalmeta>`*

Content models for *<dvrResourcePrefix>* (305)

In this document type	`<dvrResourcePrefix>` contains
Base map	(text data | *`<text>`*)*

In this document type	`<dvrResourcePrefix>` is contained by
Base map	*`<ditavalmeta>`*

Content models for *<dvrResourceSuffix>* (306)

In this document type	`<dvrResourceSuffix>` contains
Base map	(text data | *`<text>`*)*

In this document type	`<dvrResourceSuffix>` is contained by
Base map	*`<ditavalmeta>`*

C.5 E elements

Content models for elements beginning with "e".

Content models for *<elementdef>* *(314)*

None of the document type shells distributed with the base package use this element.

Content models for *<entry>* *(229)*

In this document type	`<entry>` contains
Base map, Base topic	(text data | *`<dl>`* | *`<div>`* | *`<fig>`* | *`<imagemap>`* | *`<image>`* | *`<lines>`* | *`<lq>`* | *`<note>`* | *`<hazardstatement>`* | *`<object>`* | *`<ol>`* | *`<p>`* | *`<pre>`* | *`<sl>`* | *`<ul>`* | *`<boolean>`* | *`<cite>`* | *`<keyword>`* | *`<ph>`* | *`<b>`* | *`<i>`* | *`<line-through>`* | *`<overline>`* | *`<sup>`* | *`<sub>`* | *`<tt>`* | *`<u>`* | *`<q>`* | *`<term>`* | *`<text>`* | *`<tm>`* | *`<xref>`* | *`<state>`* | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<draft-comment>`* | *`<fn>`* | *`<indextermref>`* | *`<indexterm>`* | *`<required-cleanup>`*)*

In this document type	`<entry>` is contained by
Base map, Base topic	*`<row>`*

Content models for *<enumerationdef>* *(313)*

None of the document type shells distributed with the base package use this element.

Content models for *<example>* *(199)*

In this document type	`<example>` contains
Base topic	(text data | *`<dl>`* | *`<div>`* | *`<fig>`* | *`<imagemap>`* | *`<image>`* | *`<lines>`* | *`<lq>`* | *`<note>`* | *`<hazardstatement>`* | *`<object>`* | *`<ol>`* | *`<p>`* | *`<pre>`* | *`<simpletable>`* | *`<sl>`* | *`<table>`* | *`<ul>`* | *`<boolean>`* | *`<cite>`* | *`<keyword>`* | *`<ph>`* | *`<b>`* | *`<i>`* | *`<line-through>`* | *`<overline>`* | *`<sup>`* | *`<sub>`* | *`<tt>`* | *`<u>`* | *`<q>`* | *`<term>`* | *`<text>`* | *`<tm>`* | *`<xref>`* | *`<state>`* | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<title>`* | *`<draft-comment>`* | *`<fn>`* | *`<indextermref>`* | *`<indexterm>`* | *`<required-cleanup>`*)*

In this document type	`<example>` is contained by
Base topic	*`<body>`*

Content models for *<exportanchors>* *(280)*

In this document type	`<exportanchors>` contains
Base map	(*`<anchorid>`* | *`<anchorkey>`*)*

In this document type	`<exportanchors>` is contained by
Base map	*`<metadata>`*, *`<topicmeta>`*

C.6 F elements

Content models for elements beginning with "f".

Content models for *<featnum>* (262)

In this document type	<featnum> contains
Base map, Base topic	(text data | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<keyword>* | *<term>* | *<text>*)*

In this document type	<featnum> is contained by
Base map, Base topic	*<prodinfo>*

Content models for *<fig>* (200)

In this document type	<fig> contains
Base map, Base topic	*<title>*?, *<desc>*?, (*<figgroup>* | *<dl>* | *<div>* | *<image>* | *<lines>* | *<lq>* | *<note>* | *<hazardstatement>* | *<object>* | * * | *<p>* | *<pre>* | *<sl>* | * * | *<data>* | *<sort-as>* | *<data-about>* | *<fn>* | *<foreign>* | *<unknown>* | *<simpletable>* | *<xref>*)*

In this document type	<fig> is contained by
Base map	*<dd>*, *<div>*, *<draft-comment>*, *<entry>*, *<fn>*, *<itemgroup>*, * *, *<lq>*, *<note>*, *<p>*, *<stentry>*
Base topic	**, *<body>*, *<bodydiv>*, *<dd>*, *<div>*, *<draft-comment>*, *<entry>*, *<example>*, *<fn>*, *<itemgroup>*, * *, *<lq>*, *<note>*, *<p>*, *<section>*, *<sectiondiv>*, *<stentry>*

Content models for *<figgroup>* (200)

In this document type	<figgroup> contains
Base map, Base topic	*<title>*?, (*<figgroup>* | *<dl>* | *<div>* | *<image>* | *<lines>* | *<lq>* | *<note>* | *<hazardstatement>* | *<object>* | * * | *<p>* | *<pre>* | *<sl>* | * * | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<draft-comment>* | *<fn>* | *<foreign>* | *<unknown>* | *<required-cleanup>*)*

In this document type	<figgroup> is contained by
Base map, Base topic	*<fig>*, *<figgroup>*

Content models for *<fn>* (201)

In this document type	<fn> contains
Base map, Base topic	(text data | *<dl>* | *<div>* | *<fig>* | *<imagemap>* | *<image>* | *<lines>* | *<lq>* | *<note>* | *<hazardstatement>* | *<object>* | * * | *<p>* | *<pre>* | *<sl>* | * * | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-*

In this document type	`<fn>` contains
	*`as>`	`<data-about>`	`<draft-comment>`	`<foreign>`	`<unknown>`	`<required-cleanup>`)**

In this document type	`<fn>` is contained by
Base map	*`<dd>`, `<div>`, `<entry>`, `<fig>`, `<figgroup>`, `<itemgroup>`, `<li>`, `<lines>`, `<lq>`, `<note>`, `<p>`, `<ph>`, `<pre>`, `<sli>`, `<stentry>`*
Base topic	*`<abstract>`, `<bodydiv>`, `<dd>`, `<div>`, `<entry>`, `<example>`, `<fig>`, `<figgroup>`, `<itemgroup>`, `<li>`, `<lines>`, `<lq>`, `<note>`, `<p>`, `<ph>`, `<pre>`, `<section>`, `<sectiondiv>`, `<sli>`, `<stentry>`*

Content models for *<foreign>* (334)

In this document type	`<foreign>` contains
Base map, Base topic	Any defined element

In this document type	`<foreign>` is contained by
Base map	*`<alt>`, `<author>`, `<b>`, `<brand>`, `<category>`, `<cite>`, `<component>`, `<consequence>`, `<coords>`, `<copyrholder>`, `<data>`, `<dd>`, `<ddhd>`, `<desc>`, `<div>`, `<draft-comment>`, `<dt>`, `<dthd>`, `<entry>`, `<featnum>`, `<fig>`, `<figgroup>`, `<fn>`, `<i>`, `<index-base>`, `<index-see>`, `<index-see-also>`, `<index-sort-as>`, `<indexterm>`, `<itemgroup>`, `<li>`, `<line-through>`, `<lines>`, `<linktext>`, `<lq>`, `<metadata>`, `<navtitle>`, `<note>`, `<object>`, `<overline>`, `<p>`, `<ph>`, `<platform>`, `<pre>`, `<prodname>`, `<prognum>`, `<publisher>`, `<q>`, `<searchtitle>`, `<series>`, `<shortdesc>`, `<sli>`, `<source>`, `<stentry>`, `<sub>`, `<sup>`, `<title>`, `<topicmeta>`, `<tt>`, `<typeofhazard>`, `<u>`, `<xref>`*
Base topic	*`<abstract>`, `<alt>`, `<author>`, `<b>`, `<body>`, `<bodydiv>`, `<brand>`, `<category>`, `<cite>`, `<component>`, `<consequence>`, `<coords>`, `<copyrholder>`, `<data>`, `<dd>`, `<ddhd>`, `<desc>`, `<div>`, `<draft-comment>`, `<dt>`, `<dthd>`, `<entry>`, `<example>`, `<featnum>`, `<fig>`, `<figgroup>`, `<fn>`, `<i>`, `<index-base>`, `<index-see>`, `<index-see-also>`, `<index-sort-as>`, `<indexterm>`, `<itemgroup>`, `<li>`, `<line-through>`, `<lines>`, `<linkinfo>`, `<linktext>`, `<lq>`, `<metadata>`, `<navtitle>`, `<note>`, `<object>`, `<overline>`, `<p>`, `<ph>`, `<platform>`, `<pre>`, `<prodname>`, `<prognum>`, `<prolog>`, `<publisher>`, `<q>`, `<searchtitle>`, `<section>`, `<sectiondiv>`, `<series>`, `<shortdesc>`, `<sli>`, `<source>`, `<stentry>`, `<sub>`, `<sup>`, `<title>`, `<tt>`, `<typeofhazard>`, `<u>`, `<xref>`*

C.7 H elements

Content models for elements beginning with "h".

Content models for *<hasInstance>* (310)

None of the document type shells distributed with the base package use this element.

Content models for *<hasKind>* (311)

None of the document type shells distributed with the base package use this element.

Content models for *<hasNarrower>* *(311)*

None of the document type shells distributed with the base package use this element.

Content models for *<hasPart>* *(312)*

None of the document type shells distributed with the base package use this element.

Content models for *<hasRelated>* *(313)*

None of the document type shells distributed with the base package use this element.

Content models for *<hazardstatement>* *(289)*

In this document type	`<hazardstatement>` contains
Base map, Base topic	*<messagepanel>*+, *<hazardsymbol>**

In this document type	`<hazardstatement>` is contained by
Base map	*<dd>*, *<desc>*, *<div>*, *<draft-comment>*, *<entry>*, *<fig>*, *<figgroup>*, *<fn>*, *<itemgroup>*, * *, *<lq>*, *<p>*, *<stentry>*
Base topic	**, *<body>*, *<bodydiv>*, *<dd>*, *<desc>*, *<div>*, *<draft-comment>*, *<entry>*, *<example>*, *<fig>*, *<figgroup>*, *<fn>*, *<itemgroup>*, * *, *<linkinfo>*, *<lq>*, *<p>*, *<section>*, *<sectiondiv>*, *<stentry>*

Content models for *<hazardsymbol>* *(290)*

In this document type	`<hazardsymbol>` contains
Base map, Base topic	*<alt>*?, *<longdescref>*?

In this document type	`<hazardsymbol>` is contained by
Base map, Base topic	*<hazardstatement>*

Content models for *<howtoavoid>* *(292)*

In this document type	`<howtoavoid>` contains
Base map, Base topic	(text data | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<sl>* | *<simpletable>*)*

In this document type	`<howtoavoid>` is contained by
Base map, Base topic	*<messagepanel>*

C.8 I elements

Content models for elements beginning with "i".

Content models for *<i>* (294)

In this document type	<i> contains
Base map, Base topic	(text data | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<draft-comment>* | *<foreign>* | *<unknown>* | *<required-cleanup>*)*

In this document type	<i> is contained by
Base map	*<alt>*, *<b>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*
Base topic	**, *<alt>*, *<b>*, *<bodydiv>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<searchtitle>*, *<section>*, *<sectiondiv>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*

Content models for *<image>* (203)

In this document type	<image> contains
Base map, Base topic	*<alt>*?, *<longdescref>*?

In this document type	<image> is contained by
Base map	*<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<fig>*, *<figgroup>*, *<fn>*, *<imagemap>*, *<itemgroup>*, * *, *<lq>*, *<note>*, *<p>*, *<ph>*, *<shortdesc>*, *<sli>*, *<stentry>*, *<title>*, *<xref>*
Base topic	**, *<body>*, *<bodydiv>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<fig>*, *<figgroup>*, *<fn>*, *<imagemap>*, *<itemgroup>*, * *, *<linkinfo>*, *<lq>*, *<note>*, *<p>*, *<ph>*, *<section>*, *<sectiondiv>*, *<shortdesc>*, *<sli>*, *<stentry>*, *<title>*, *<xref>*

Content models for *<imagemap>* (298)

In this document type	<imagemap> contains
Base map, Base topic	*<image>*, *<area>*+

In this document type	**`<imagemap>` is contained by**
Base map	*`<dd>`*, *`<div>`*, *`<draft-comment>`*, *`<entry>`*, *`<fn>`*, *`<itemgroup>`*, *`<li>`*, *`<lq>`*, *`<note>`*, *`<p>`*, *`<stentry>`*
Base topic	*`<abstract>`*, *`<body>`*, *`<bodydiv>`*, *`<dd>`*, *`<div>`*, *`<draft-comment>`*, *`<entry>`*, *`<example>`*, *`<fn>`*, *`<itemgroup>`*, *`<li>`*, *`<lq>`*, *`<note>`*, *`<p>`*, *`<section>`*, *`<sectiondiv>`*, *`<stentry>`*

Content models for *<index-base>* (335)

In this document type	**`<index-base>` contains**
Base map, Base topic	(text data | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<keyword>`* | *`<term>`* | *`<text>`* | *`<indexterm>`*)*

In this document type	**`<index-base>` is contained by**
Base map, Base topic	*`<indexterm>`*

Content models for *<index-see>* (276)

In this document type	**`<index-see>` contains**
Base map, Base topic	(text data | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<keyword>`* | *`<term>`* | *`<text>`* | *`<ph>`* | *`<b>`* | *`<i>`* | *`<line-through>`* | *`<overline>`* | *`<sup>`* | *`<sub>`* | *`<tt>`* | *`<u>`* | *`<indexterm>`*)*

In this document type	**`<index-see>` is contained by**
Base map, Base topic	*`<indexterm>`*

Content models for *<index-see-also>* (278)

In this document type	**`<index-see-also>` contains**
Base map, Base topic	(text data | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<keyword>`* | *`<term>`* | *`<text>`* | *`<ph>`* | *`<b>`* | *`<i>`* | *`<line-through>`* | *`<overline>`* | *`<sup>`* | *`<sub>`* | *`<tt>`* | *`<u>`* | *`<indexterm>`*)*

In this document type	**`<index-see-also>` is contained by**
Base map, Base topic	*`<indexterm>`*

Content models for *<index-sort-as>* (279)

In this document type	**`<index-sort-as>` contains**
Base map, Base topic	(text data | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<keyword>`* | *`<term>`* | *`<text>`*)*

In this document type	**`<index-sort-as>` is contained by**
Base map, Base topic	*`<indexterm>`*

Content models for *<indexterm>* *(272)*

In this document type	**<indexterm>** contains
Base map, Base topic	(text data | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<keyword>* | *<term>* | *<text>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<indexterm>* | *<index-base>* | *<index-see>* | *<index-see-also>* | *<index-sort-as>*)*

In this document type	**<indexterm>** is contained by
Base map	*<dd>*, *<div>*, *<entry>*, *<index-base>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, *<keywords>*, * *, *<lines>*, *<lq>*, *<note>*, *<p>*, *<ph>*, *<pre>*, *<sli>*, *<stentry>*
Base topic	**, *<bodydiv>*, *<dd>*, *<div>*, *<entry>*, *<example>*, *<index-base>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, *<keywords>*, * *, *<lines>*, *<lq>*, *<note>*, *<p>*, *<ph>*, *<pre>*, *<section>*, *<sectiondiv>*, *<sli>*, *<stentry>*

Content models for *<indextermref>* *(276)*

In this document type	**<indextermref>** contains
Base map, Base topic	EMPTY

In this document type	**<indextermref>** is contained by
Base map	*<dd>*, *<div>*, *<entry>*, *<itemgroup>*, * *, *<lines>*, *<lq>*, *<note>*, *<p>*, *<ph>*, *<pre>*, *<sli>*, *<stentry>*
Base topic	**, *<bodydiv>*, *<dd>*, *<div>*, *<entry>*, *<example>*, *<itemgroup>*, * *, *<lines>*, *<lq>*, *<note>*, *<p>*, *<ph>*, *<pre>*, *<section>*, *<sectiondiv>*, *<sli>*, *<stentry>*

Content models for *<itemgroup>* *(336)*

In this document type	**<itemgroup>** contains
Base map, Base topic	(text data | *<dl>* | *<div>* | *<fig>* | *<imagemap>* | *<image>* | *<lines>* | *<lq>* | *<note>* | *<hazardstatement>* | *<object>* | * * | *<p>* | *<pre>* | *<simpletable>* | *<sl>* | *<table>* | * * | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<draft-comment>* | *<fn>* | *<indextermref>* | *<indexterm>* | *<required-cleanup>*)*

In this document type	**<itemgroup>** is contained by
Base map, Base topic	*<dd>*, * *

C.9 K elements

Content models for elements beginning with "k".

Content models for *<keydef>* (250)

In this document type	**<keydef> contains**
Base map	*<topicmeta>*?, (*<anchor>* | *<data>* | *<sort-as>* | *<data-about>* | *<navref>* | *<topicref>* | *<anchorref>* | *<keydef>* | *<mapref>* | *<topicgroup>* | *<topichead>* | *<topicset>* | *<topicsetref>* | *<ditavalref>*)*

In this document type	**<keydef> is contained by**
Base map	*<anchorref>*, *<keydef>*, *<map>*, *<relcell>*, *<relcolspec>*, *<topicgroup>*, *<topichead>*, *<topicref>*, *<topicset>*, *<topicsetref>*

Content models for *<keyword>* (204)

In this document type	**<keyword> contains**
Base map, Base topic	(text data | *<draft-comment>* | *<required-cleanup>* | *<text>* | *<tm>*)*

In this document type	**<keyword> is contained by**
Base map	*<alt>*, *<author>*, *<b>*, *<brand>*, *<category>*, *<cite>*, *<component>*, *<consequence>*, *<coords>*, *<copyrholder>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<featnum>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-base>*, *<index-see>*, *<index-see-also>*, *<index-sort-as>*, *<indexterm>*, *<itemgroup>*, *<keywords>*, * *, *<line-through>*, *<lines>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<platform>*, *<pre>*, *<prodname>*, *<prognum>*, *<publisher>*, *<q>*, *<searchtitle>*, *<series>*, *<shortdesc>*, *<sli>*, *<sort-as>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*
Base topic	**, *<alt>*, *<author>*, *<b>*, *<bodydiv>*, *<brand>*, *<category>*, *<cite>*, *<component>*, *<consequence>*, *<coords>*, *<copyrholder>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<featnum>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-base>*, *<index-see>*, *<index-see-also>*, *<index-sort-as>*, *<indexterm>*, *<itemgroup>*, *<keywords>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<platform>*, *<pre>*, *<prodname>*, *<prognum>*, *<publisher>*, *<q>*, *<searchtitle>*, *<section>*, *<sectiondiv>*, *<series>*, *<shortdesc>*, *<sli>*, *<sort-as>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*

Content models for *<keywords>* (262)

In this document type	**<keywords> contains**
Base map, Base topic	(*<indexterm>* | *<keyword>*)*

In this document type	`<keywords>` is contained by
Base map	*`<metadata>`*, *`<topicmeta>`*
Base topic	*`<metadata>`*

C.10 L elements

Content models for elements beginning with "l".

Content models for *`<li>`* (205)

In this document type	`<li>` contains
Base map, Base topic	(text data | *`<dl>`* | *`<div>`* | *`<fig>`* | *`<imagemap>`* | *`<image>`* | *`<lines>`* | *`<lq>`* | *`<note>`* | *`<hazardstatement>`* | *`<object>`* | *`<ol>`* | *`<p>`* | *`<pre>`* | *`<simpletable>`* | *`<sl>`* | *`<table>`* | *`<ul>`* | *`<boolean>`* | *`<cite>`* | *`<keyword>`* | *`<ph>`* | *`<b>`* | *`<i>`* | *`<line-through>`* | *`<overline>`* | *`<sup>`* | *`<sub>`* | *`<tt>`* | *`<u>`* | *`<q>`* | *`<term>`* | *`<text>`* | *`<tm>`* | *`<xref>`* | *`<state>`* | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<itemgroup>`* | *`<draft-comment>`* | *`<fn>`* | *`<indextermref>`* | *`<indexterm>`* | *`<required-cleanup>`*)*

In this document type	`<li>` is contained by
Base map, Base topic	*`<ol>`*, *`<ul>`*

Content models for *`<line-through>`* (296)

In this document type	`<line-through>` contains
Base map, Base topic	(text data | *`<boolean>`* | *`<cite>`* | *`<keyword>`* | *`<ph>`* | *`<b>`* | *`<i>`* | *`<line-through>`* | *`<overline>`* | *`<sup>`* | *`<sub>`* | *`<tt>`* | *`<u>`* | *`<q>`* | *`<term>`* | *`<text>`* | *`<tm>`* | *`<xref>`* | *`<state>`* | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<draft-comment>`* | *`<foreign>`* | *`<unknown>`* | *`<required-cleanup>`*)*

In this document type	`<line-through>` is contained by
Base map	*`<alt>`*, *`<b>`*, *`<cite>`*, *`<consequence>`*, *`<data>`*, *`<dd>`*, *`<ddhd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<dt>`*, *`<dthd>`*, *`<entry>`*, *`<figgroup>`*, *`<fn>`*, *`<howtoavoid>`*, *`<i>`*, *`<index-see>`*, *`<index-see-also>`*, *`<indexterm>`*, *`<itemgroup>`*, *`<li>`*, *`<line-through>`*, *`<lines>`*, *`<linktext>`*, *`<lq>`*, *`<navtitle>`*, *`<note>`*, *`<overline>`*, *`<p>`*, *`<ph>`*, *`<pre>`*, *`<q>`*, *`<shortdesc>`*, *`<sli>`*, *`<source>`*, *`<stentry>`*, *`<sub>`*, *`<sup>`*, *`<title>`*, *`<tt>`*, *`<typeofhazard>`*, *`<u>`*, *`<xref>`*
Base topic	*`<abstract>`*, *`<alt>`*, *`<b>`*, *`<bodydiv>`*, *`<cite>`*, *`<consequence>`*, *`<data>`*, *`<dd>`*, *`<ddhd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<dt>`*, *`<dthd>`*, *`<entry>`*, *`<example>`*, *`<figgroup>`*, *`<fn>`*, *`<howtoavoid>`*, *`<i>`*, *`<index-see>`*, *`<index-see-also>`*, *`<indexterm>`*, *`<itemgroup>`*, *`<li>`*, *`<line-through>`*, *`<lines>`*, *`<linkinfo>`*, *`<linktext>`*, *`<lq>`*, *`<navtitle>`*, *`<note>`*, *`<overline>`*, *`<p>`*, *`<ph>`*, *`<pre>`*, *`<q>`*, *`<searchtitle>`*, *`<section>`*, *`<sectiondiv>`*, *`<shortdesc>`*, *`<sli>`*, *`<source>`*, *`<stentry>`*, *`<sub>`*, *`<sup>`*, *`<title>`*, *`<tt>`*, *`<typeofhazard>`*, *`<u>`*, *`<xref>`*

Content models for *<lines>* (205)

In this document type	`<lines>` contains
Base map, Base topic	(text data | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<draft-comment>* | *<fn>* | *<indextermref>* | *<indexterm>* | *<required-cleanup>*)*

In this document type	`<lines>` is contained by
Base map	*<dd>*, *<desc>*, *<div>*, *<draft-comment>*, *<entry>*, *<fig>*, *<figgroup>*, *<fn>*, *<itemgroup>*, * *, *<lq>*, *<note>*, *<p>*, *<stentry>*
Base topic	**, *<body>*, *<bodydiv>*, *<dd>*, *<desc>*, *<div>*, *<draft-comment>*, *<entry>*, *<example>*, *<fig>*, *<figgroup>*, *<fn>*, *<itemgroup>*, * *, *<linkinfo>*, *<lq>*, *<note>*, *<p>*, *<section>*, *<sectiondiv>*, *<stentry>*

Content models for *<link>* (232)

In this document type	`<link>` contains
Base topic	*<linktext>*?, *<desc>*?

In this document type	`<link>` is contained by
Base topic	*<linklist>*, *<linkpool>*, *<related-links>*

Content models for *<linkinfo>* (236)

In this document type	`<linkinfo>` contains
Base topic	(text data | *<dl>* | *<div>* | *<image>* | *<lines>* | *<lq>* | *<note>* | *<hazardstatement>* | * * | *<p>* | *<pre>* | *<sl>* | * * | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<draft-comment>* | *<foreign>* | *<unknown>* | *<required-cleanup>*)*

In this document type	`<linkinfo>` is contained by
Base topic	*<linklist>*

Content models for *<linklist>* (233)

In this document type	`<linklist>` contains
Base topic	*<title>*?, *<desc>*?, (*<linklist>* | *<link>*)*, *<linkinfo>*?

In this document type	`<linklist>` is contained by
Base topic	*<linklist>*, *<related-links>*

Content models for *<linkpool>* (235)

In this document type	`<linkpool>` contains
Base topic	(*<linkpool>* | *<link>*)*

In this document type	`<linkpool>` is contained by
Base topic	*<linkpool>*, *<related-links>*

Content models for *<linktext>* (236)

In this document type	`<linktext>` contains
Base map, Base topic	(text data | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<keyword>* | *<term>* | *<text>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>*)*

In this document type	`<linktext>` is contained by
Base map	*<topicmeta>*
Base topic	*<link>*

Content models for *<longdescref>* (206)

In this document type	`<longdescref>` contains
Base map, Base topic	EMPTY

In this document type	`<longdescref>` is contained by
Base map, Base topic	*<hazardsymbol>*, *<image>*, *<object>*

Content models for *<longquoteref>* (207)

In this document type	`<longquoteref>` contains
Base map, Base topic	EMPTY

In this document type	`<longquoteref>` is contained by
Base map, Base topic	*<lq>*

Content models for *<lq>* (207)

In this document type	`<lq>` contains
Base map, Base topic	(text data | *<dl>* | *<div>* | *<fig>* | *<imagemap>* | *<image>* | *<lines>* | *<note>* | *<hazardstatement>* | *<object>* | * * | *<p>* | *<pre>* | *<simpletable>* | *<sl>* | *<table>* | * * | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* |

In this document type	`<lq>` contains
	`<longquoteref>` | *`<draft-comment>`* | *`<fn>`* | *`<indextermref>`* | *`<indexterm>`* | *`<required-cleanup>`*)*

In this document type	`<lq>` is contained by
Base map	*`<dd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<entry>`*, *`<fig>`*, *`<figgroup>`*, *`<fn>`*, *`<itemgroup>`*, *`<li>`*, *`<note>`*, *`<p>`*, *`<stentry>`*
Base topic	*`<abstract>`*, *`<body>`*, *`<bodydiv>`*, *`<dd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<entry>`*, *`<example>`*, *`<fig>`*, *`<figgroup>`*, *`<fn>`*, *`<itemgroup>`*, *`<li>`*, *`<linkinfo>`*, *`<note>`*, *`<p>`*, *`<section>`*, *`<sectiondiv>`*, *`<stentry>`*

C.11 M elements

Content models for elements beginning with "m".

Content models for *<map>* (237)

In this document type	`<map>` contains
Base map	*`<title>`*?, *`<topicmeta>`*?, (*`<anchor>`* | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<navref>`* | *`<reltable>`* | *`<topicref>`* | *`<anchorref>`* | *`<keydef>`* | *`<mapref>`* | *`<topicgroup>`* | *`<topichead>`* | *`<topicset>`* | *`<topicsetref>`* | *`<ditavalref>`*)*

In this document type	`<map>` is contained by
Base map	*`<foreign>`*, *`<required-cleanup>`*, *`<unknown>`*

Content models for *<mapref>* (251)

In this document type	`<mapref>` contains
Base map	*`<topicmeta>`*?, (*`<data>`* | *`<sort-as>`* | *`<data-about>`*)*

In this document type	`<mapref>` is contained by
Base map	*`<anchorref>`*, *`<keydef>`*, *`<map>`*, *`<relcell>`*, *`<relcolspec>`*, *`<topicgroup>`*, *`<topichead>`*, *`<topicref>`*, *`<topicset>`*, *`<topicsetref>`*

Content models for *<messagepanel>* (292)

In this document type	`<messagepanel>` contains
Base map, Base topic	(*`<data>`* | *`<sort-as>`* | *`<data-about>`*)*, *`<typeofhazard>`*, *`<consequence>`**, *`<howtoavoid>`*+

In this document type	`<messagepanel>` is contained by
Base map, Base topic	*`<hazardstatement>`*

Content models for *<metadata>* (263)

In this document type	`<metadata>` contains
Base map	*<audience>**, *<category>**, *<keywords>*, *<exportanchors>* *<exportanchors>**, *<prodinfo>**, *<othermeta>**, (*<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>*)*
Base topic	*<audience>**, *<category>**, *<keywords>**, *<prodinfo>**, *<othermeta>**, (*<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>*)*

In this document type	`<metadata>` is contained by
Base map	*<topicmeta>*
Base topic	*<prolog>*

C.12 N elements

Content models for elements beginning with "n".

Content models for *<navref>* (241)

In this document type	`<navref>` contains
Base map	EMPTY

In this document type	`<navref>` is contained by
Base map	*<keydef>*, *<map>*, *<topicgroup>*, *<topichead>*, *<topicref>*, *<topicset>*

Content models for *<navtitle>* (184)

In this document type	`<navtitle>` contains
Base map, Base topic	(text data | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<keyword>* | *<term>* | *<text>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<draft-comment>* | *<required-cleanup>*)*

In this document type	`<navtitle>` is contained by
Base map	*<ditavalmeta>*, *<topicmeta>*
Base topic	*<titlealts>*

Content models for *<no-topic-nesting>* (336)

In this document type	`<no-topic-nesting>` contains
Base topic	EMPTY

In this document type	`<no-topic-nesting>` is contained by
Base topic	No parent element

Content models for *<note>* (208)

In this document type	`<note>` contains
Base map, Base topic	(text data | *<dl>* | *<div>* | *<fig>* | *<imagemap>* | *<image>* | *<lines>* | *<lq>* | *<object>* | * * | *<p>* | *<pre>* | *<simpletable>* | *<sl>* | *<table>* | * * | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<draft-comment>* | *<fn>* | *<indextermref>* | *<indexterm>* | *<required-cleanup>*)*

In this document type	`<note>` is contained by
Base map	*<dd>*, *<desc>*, *<div>*, *<draft-comment>*, *<entry>*, *<fig>*, *<figgroup>*, *<fn>*, *<itemgroup>*, * *, *<lq>*, *<p>*, *<stentry>*
Base topic	**, *<body>*, *<bodydiv>*, *<dd>*, *<desc>*, *<div>*, *<draft-comment>*, *<entry>*, *<example>*, *<fig>*, *<figgroup>*, *<fn>*, *<itemgroup>*, * *, *<linkinfo>*, *<lq>*, *<p>*, *<section>*, *<sectiondiv>*, *<stentry>*

C.13 O elements

Content models for elements beginning with "o".

Content models for *<object>* (209)

In this document type	`<object>` contains
Base map, Base topic	*<desc>*?, *<longdescref>*?, *<param>**, (*<foreign>* | *<unknown>*)*

In this document type	`<object>` is contained by
Base map	*<data>*, *<dd>*, *<div>*, *<draft-comment>*, *<entry>*, *<fig>*, *<figgroup>*, *<fn>*, *<itemgroup>*, * *, *<lq>*, *<note>*, *<p>*, *<stentry>*
Base topic	**, *<body>*, *<bodydiv>*, *<data>*, *<dd>*, *<div>*, *<draft-comment>*, *<entry>*, *<example>*, *<fig>*, *<figgroup>*, *<fn>*, *<itemgroup>*, * *, *<lq>*, *<note>*, *<p>*, *<section>*, *<sectiondiv>*, *<stentry>*

Content models for *<ol>* (212)

In this document type	`<ol>` contains
Base map, Base topic	(*<data>* | *<sort-as>* | *<data-about>*)*, * *+

In this document type	`<ol>` is contained by
Base map	*<dd>*, *<desc>*, *<div>*, *<draft-comment>*, *<entry>*, *<fig>*, *<figgroup>*, *<fn>*, *<itemgroup>*, * *, *<lq>*, *<note>*, *<p>*, *<stentry>*
Base topic	**, *<body>*, *<bodydiv>*, *<dd>*, *<desc>*, *<div>*, *<draft-comment>*, *<entry>*, *<example>*, *<fig>*, *<figgroup>*, *<fn>*, *<itemgroup>*, * *, *<linkinfo>*, *<lq>*, *<note>*, *<p>*, *<section>*, *<sectiondiv>*, *<stentry>*

Content models for *<othermeta>* (264)

In this document type	`<othermeta>` contains
Base map, Base topic	EMPTY

In this document type	`<othermeta>` is contained by
Base map	*<metadata>*, *<topicmeta>*
Base topic	*<metadata>*

Content models for *<overline>* (297)

In this document type	`<overline>` contains
Base map, Base topic	(text data | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<draft-comment>* | *<foreign>* | *<unknown>* | *<required-cleanup>*)*

In this document type	`<overline>` is contained by
Base map	*<alt>*, *<b>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*
Base topic	**, *<alt>*, *<b>*, *<bodydiv>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<searchtitle>*, *<section>*, *<sectiondiv>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*

C.14 P elements

Content models for elements beginning with "p".

Content models for *<p>* (213)

In this document type	`<p>` contains
Base map, Base topic	(text data | *<dl>* | *<div>* | *<fig>* | *<imagemap>* | *<image>* | *<lines>* | *<lq>* | *<note>* | *<hazardstatement>* | *<object>* | * * | *<pre>* | *<simpletable>* | *<sl>* | *<table>* | * * | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<draft-comment>* | *<fn>* | *<indextermref>* | *<indexterm>* | *<required-cleanup>*)*

In this document type	`<p>` is contained by
Base map	*<dd>*, *<desc>*, *<div>*, *<draft-comment>*, *<entry>*, *<fig>*, *<figgroup>*, *<fn>*, *<itemgroup>*, * *, *<lq>*, *<note>*, *<stentry>*
Base topic	**, *<body>*, *<bodydiv>*, *<dd>*, *<desc>*, *<div>*, *<draft-comment>*, *<entry>*, *<example>*, *<fig>*, *<figgroup>*, *<fn>*, *<itemgroup>*, * *, *<linkinfo>*, *<lq>*, *<note>*, *<section>*, *<sectiondiv>*, *<stentry>*

Content models for *<param>* *(213)*

In this document type	`<param>` contains
Base map, Base topic	EMPTY

In this document type	`<param>` is contained by
Base map, Base topic	*<object>*

Content models for *<permissions>* *(264)*

In this document type	`<permissions>` contains
Base map, Base topic	EMPTY

In this document type	`<permissions>` is contained by
Base map	*<topicmeta>*
Base topic	*<prolog>*

Content models for *<ph>* *(214)*

In this document type	`<ph>` contains
Base map, Base topic	(text data | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<image>* | *<draft-comment>* | *<fn>* | *<indextermref>* | *<indexterm>* | *<required-cleanup>*)*

In this document type	`<ph>` is contained by
Base map	*<alt>*, *<b>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*
Base topic	**, *<alt>*, *<b>*, *<bodydiv>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<searchtitle>*,

In this document type	`<ph>` is contained by
	`<section>`, *`<sectiondiv>`*, *`<shortdesc>`*, *`<sli>`*, *`<source>`*, *`<stentry>`*, *`<sub>`*, *`<sup>`*, *`<title>`*, *`<tt>`*, *`<typeofhazard>`*, *`<u>`*, *`<xref>`*

Content models for *<platform>* (265)

In this document type	`<platform>` contains
Base map, Base topic	(text data | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<keyword>`* | *`<term>`* | *`<text>`*)*

In this document type	`<platform>` is contained by
Base map, Base topic	*`<prodinfo>`*

Content models for *<pre>* (215)

In this document type	`<pre>` contains
Base map, Base topic	(text data | *`<boolean>`* | *`<cite>`* | *`<keyword>`* | *`<ph>`* | *`<b>`* | *`<i>`* | *`<line-through>`* | *`<overline>`* | *`<sup>`* | *`<sub>`* | *`<tt>`* | *`<u>`* | *`<q>`* | *`<term>`* | *`<text>`* | *`<tm>`* | *`<xref>`* | *`<state>`* | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<draft-comment>`* | *`<fn>`* | *`<indextermref>`* | *`<indexterm>`* | *`<required-cleanup>`*)*

In this document type	`<pre>` is contained by
Base map	*`<dd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<entry>`*, *`<fig>`*, *`<figgroup>`*, *`<fn>`*, *`<itemgroup>`*, *`<li>`*, *`<lq>`*, *`<note>`*, *`<p>`*, *`<stentry>`*
Base topic	*`<abstract>`*, *`<body>`*, *`<bodydiv>`*, *`<dd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<entry>`*, *`<example>`*, *`<fig>`*, *`<figgroup>`*, *`<fn>`*, *`<itemgroup>`*, *`<li>`*, *`<linkinfo>`*, *`<lq>`*, *`<note>`*, *`<p>`*, *`<section>`*, *`<sectiondiv>`*, *`<stentry>`*

Content models for *<prodinfo>* (265)

In this document type	`<prodinfo>` contains
Base map, Base topic	*`<prodname>`*, *`<vrmlist>`*?, (*`<brand>`* | *`<component>`* | *`<featnum>`* | *`<platform>`* | *`<prognum>`* | *`<series>`*)*

In this document type	`<prodinfo>` is contained by
Base map	*`<metadata>`*, *`<topicmeta>`*
Base topic	*`<metadata>`*

Content models for *<prodname>* (266)

In this document type	`<prodname>` contains
Base map, Base topic	(text data | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<keyword>`* | *`<term>`* | *`<text>`*)*

In this document type	`<prodname>` is contained by
Base map, Base topic	*`<prodinfo>`*

Content models for *`<prognum>`* (266)

In this document type	`<prognum>` contains
Base map, Base topic	(text data | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<keyword>`* | *`<term>`* | *`<text>`*)*

In this document type	`<prognum>` is contained by
Base map, Base topic	*`<prodinfo>`*

Content models for *`<prolog>`* (256)

In this document type	`<prolog>` contains
Base topic	*`<author>`** , *`<source>`*?, *`<publisher>`*?, *`<copyright>`** , *`<critdates>`*?, *`<permissions>`*?, *`<metadata>`** , *`<resourceid>`** , (*`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`*)*

In this document type	`<prolog>` is contained by
Base topic	*`<topic>`*

Content models for *`<publisher>`* (267)

In this document type	`<publisher>` contains
Base map, Base topic	(text data | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<keyword>`* | *`<term>`* | *`<text>`*)*

In this document type	`<publisher>` is contained by
Base map	*`<topicmeta>`*
Base topic	*`<prolog>`*

C.15 Q elements

Content models for elements beginning with "q".

Content models for *`<q>`* (215)

In this document type	`<q>` contains
Base map, Base topic	(text data | *`<boolean>`* | *`<cite>`* | *`<keyword>`* | *`<ph>`* | *`<b>`* | *`<i>`* | *`<line-through>`* | *`<overline>`* | *`<sup>`* | *`<sub>`* | *`<tt>`* | *`<u>`* | *`<q>`* | *`<term>`* | *`<text>`* | *`<tm>`* | *`<xref>`* | *`<state>`* | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<draft-comment>`* | *`<foreign>`* | *`<unknown>`* | *`<required-cleanup>`*)*

In this document type	<q> is contained by
Base map	*<b>*, *<cite>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<lq>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<shortdesc>*, *<sli>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<u>*, *<xref>*
Base topic	**, *<b>*, *<bodydiv>*, *<cite>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<lq>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<section>*, *<sectiondiv>*, *<shortdesc>*, *<sli>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<u>*, *<xref>*

C.16 R elements

Content models for elements beginning with "r".

Content models for *<related-links>* (190)

In this document type	<related-links> contains
Base topic	(*<link>* | *<linklist>* | *<linkpool>*)*

In this document type	<related-links> is contained by
Base topic	*<topic>*

Content models for *<relatedSubjects>* (320)

None of the document type shells distributed with the base package use this element.

Content models for *<relcell>* (244)

In this document type	<relcell> contains
Base map	(*<topicref>* | *<anchorref>* | *<keydef>* | *<mapref>* | *<topicgroup>* | *<topichead>* | *<topicset>* | *<topicsetref>* | *<ditavalref>* | *<data>* | *<sort-as>* | *<data-about>*)*

In this document type	<relcell> is contained by
Base map	*<relrow>*

Content models for *<relcolspec>* (245)

In this document type	<relcolspec> contains
Base map	*<title>*?, *<topicmeta>*?, *<topicref>*, (*<anchorref>* | *<keydef>* | *<mapref>* | *<topicgroup>* | *<topichead>* | *<topicset>* | *<topicsetref>*), (*<ditavalref>*)(*<anchorref>* | *<keydef>* | *<mapref>* | *<topicgroup>* | *<topichead>* | *<topicset>* | *<topicsetref>*), (*<ditavalref>*)(*<ditavalref>*)*

In this document type	<relcolspec> is contained by
Base map	*<relheader>*

Content models for *‹relheader›* *(244)*

In this document type	**‹relheader› contains**
Base map	*‹relcolspec›*+

In this document type	**‹relheader› is contained by**
Base map	*‹reltable›*

Content models for *‹relrow›* *(244)*

In this document type	**‹relrow› contains**
Base map	*‹relcell›**

In this document type	**‹relrow› is contained by**
Base map	*‹reltable›*

Content models for *‹reltable›* *(242)*

In this document type	**‹reltable› contains**
Base map	*‹title›*?, *‹topicmeta›*?, *‹relheader›*?, *‹relrow›*+

In this document type	**‹reltable› is contained by**
Base map	*‹map›*

Content models for *‹required-cleanup›* *(338)*

In this document type	**‹required-cleanup› contains**
Base map, Base topic	Any defined element

In this document type	**‹required-cleanup› is contained by**
Base map	*‹alt›*, *‹b›*, *‹cite›*, *‹data›*, *‹dd›*, *‹ddhd›*, *‹desc›*, *‹div›*, *‹dt›*, *‹dthd›*, *‹entry›*, *‹figgroup›*, *‹fn›*, *‹i›*, *‹itemgroup›*, *‹keyword›*, *‹li›*, *‹line-through›*, *‹lines›*, *‹lq›*, *‹navtitle›*, *‹note›*, *‹overline›*, *‹p›*, *‹ph›*, *‹pre›*, *‹q›*, *‹shortdesc›*, *‹sli›*, *‹stentry›*, *‹sub›*, *‹sup›*, *‹term›*, *‹title›*, *‹tt›*, *‹u›*, *‹xref›*
Base topic	*‹abstract›*, *‹alt›*, *‹b›*, *‹body›*, *‹bodydiv›*, *‹cite›*, *‹data›*, *‹dd›*, *‹ddhd›*, *‹desc›*, *‹div›*, *‹dt›*, *‹dthd›*, *‹entry›*, *‹example›*, *‹figgroup›*, *‹fn›*, *‹i›*, *‹itemgroup›*, *‹keyword›*, *‹li›*, *‹line-through›*, *‹lines›*, *‹linkinfo›*, *‹lq›*, *‹navtitle›*, *‹note›*, *‹overline›*, *‹p›*, *‹ph›*, *‹pre›*, *‹q›*, *‹section›*, *‹sectiondiv›*, *‹shortdesc›*, *‹sli›*, *‹stentry›*, *‹sub›*, *‹sup›*, *‹term›*, *‹title›*, *‹tt›*, *‹u›*, *‹xref›*

Content models for *‹resourceid›* *(267)*

In this document type	**‹resourceid› contains**
Base map, Base topic	(*‹data›* | *‹sort-as›* | *‹data-about›*)*

In this document type	`<resourceid>` is contained by
Base map	*`<topicmeta>`*
Base topic	*`<prolog>`*

Content models for *<revised>* (269)

In this document type	`<revised>` contains
Base map, Base topic	EMPTY

In this document type	`<revised>` is contained by
Base map, Base topic	*`<critdates>`*

Content models for *<row>* (228)

In this document type	`<row>` contains
Base map, Base topic	*`<entry>`*+

In this document type	`<row>` is contained by
Base map, Base topic	*`<tbody>`*, *`<thead>`*

C.17 S elements

Content models for elements beginning with "s".

Content models for *<schemeref>* (309)

None of the document type shells distributed with the base package use this element.

Content models for *<searchtitle>* (183)

In this document type	`<searchtitle>` contains
Base map	(text data | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<keyword>`* | *`<term>`* | *`<text>`*)*
Base topic	(text data | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<keyword>`* | *`<term>`* | *`<text>`* | *`<ph>`* | *`<b>`* | *`<i>`* | *`<line-through>`* | *`<overline>`* | *`<sup>`* | *`<sub>`* | *`<tt>`* | *`<u>`*)*

In this document type	`<searchtitle>` is contained by
Base map	*`<topicmeta>`*
Base topic	*`<titlealts>`*

Content models for *<section>* (216)

In this document type	`<section>` contains
Base topic	(text data | *`<dl>`* | *`<div>`* | *`<fig>`* | *`<imagemap>`* | *`<image>`* | *`<lines>`* | *`<lq>`* | *`<note>`* | *`<hazardstatement>`* | *`<object>`* | *`<ol>`* | *`<p>`* | *`<pre>`* | *`<simpletable>`* | *`<sl>`* |

In this document type	`<section>` contains
	`<table>` | `<ul>` | `<boolean>` | `<cite>` | `<keyword>` | `<ph>` | `<b>` | `<i>` | `<line-through>` | `<overline>` | `<sup>` | `<sub>` | `<tt>` | `<u>` | `<q>` | `<term>` | `<text>` | `<tm>` | `<xref>` | `<state>` | `<data>` | `<sort-as>` | `<data-about>` | `<foreign>` | `<unknown>` | `<sectiondiv>` | `<title>` | `<draft-comment>` | `<fn>` | `<indextermref>` | `<indexterm>` | `<required-cleanup>`)*

In this document type	`<section>` is contained by
Base topic	`<body>`, `<bodydiv>`

Content models for `<sectiondiv>` *(216)*

In this document type	`<sectiondiv>` contains
Base topic	(text data | `<dl>` | `<div>` | `<fig>` | `<imagemap>` | `<image>` | `<lines>` | `<lq>` | `<note>` | `<hazardstatement>` | `<object>` | `<ol>` | `<p>` | `<pre>` | `<simpletable>` | `<sl>` | `<table>` | `<ul>` | `<boolean>` | `<cite>` | `<keyword>` | `<ph>` | `<b>` | `<i>` | `<line-through>` | `<overline>` | `<sup>` | `<sub>` | `<tt>` | `<u>` | `<q>` | `<term>` | `<text>` | `<tm>` | `<xref>` | `<state>` | `<data>` | `<sort-as>` | `<data-about>` | `<foreign>` | `<unknown>` | `<draft-comment>` | `<fn>` | `<indextermref>` | `<indexterm>` | `<required-cleanup>` | `<sectiondiv>`)*

In this document type	`<sectiondiv>` is contained by
Base topic	`<section>`, `<sectiondiv>`

Content models for `<series>` *(269)*

In this document type	`<series>` contains
Base map, Base topic	(text data | `<data>` | `<sort-as>` | `<data-about>` | `<foreign>` | `<unknown>` | `<keyword>` | `<term>` | `<text>`)*

In this document type	`<series>` is contained by
Base map, Base topic	`<prodinfo>`

Content models for `<shape>` *(300)*

In this document type	`<shape>` contains
Base map, Base topic	(text data | `<text>`)*

In this document type	`<shape>` is contained by
Base map, Base topic	`<area>`

Content models for *<shortdesc>* (185)

In this document type	`<shortdesc>` contains
Base map, Base topic	(text data | *<boolean>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<state>* | *<cite>* | *<data>* | *<sort-as>* | *<data-about>* | *<draft-comment>* | *<foreign>* | *<unknown>* | *<required-cleanup>* | *<image>* | *<xref>*)*

In this document type	`<shortdesc>` is contained by
Base map	*<topicmeta>*
Base topic	**, *<topic>*

Content models for *<simpletable>* (230)

In this document type	`<simpletable>` contains
Base map, Base topic	*<sthead>*?, *<strow>*+

In this document type	`<simpletable>` is contained by
Base map	*<dd>*, *<div>*, *<draft-comment>*, *<fig>*, *<howtoavoid>*, *<itemgroup>*, * *, *<lq>*, *<note>*, *<p>*
Base topic	**, *<body>*, *<bodydiv>*, *<dd>*, *<div>*, *<draft-comment>*, *<example>*, *<fig>*, *<howtoavoid>*, *<itemgroup>*, * *, *<lq>*, *<note>*, *<p>*, *<section>*, *<sectiondiv>*

Content models for *<sl>* (217)

In this document type	`<sl>` contains
Base map, Base topic	(*<data>* | *<sort-as>* | *<data-about>*)*, *<sli>*+

In this document type	`<sl>` is contained by
Base map	*<dd>*, *<desc>*, *<div>*, *<draft-comment>*, *<entry>*, *<fig>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<itemgroup>*, * *, *<lq>*, *<note>*, *<p>*, *<stentry>*
Base topic	**, *<body>*, *<bodydiv>*, *<dd>*, *<desc>*, *<div>*, *<draft-comment>*, *<entry>*, *<example>*, *<fig>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<itemgroup>*, * *, *<linkinfo>*, *<lq>*, *<note>*, *<p>*, *<section>*, *<sectiondiv>*, *<stentry>*

Content models for *<sli>* (217)

In this document type	`<sli>` contains
Base map, Base topic	(text data | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<image>* | *<draft-comment>* | *<fn>* | *<indextermref>* | *<indexterm>* | *<required-cleanup>*)*

In this document type	`<sli>` is contained by
Base map, Base topic	*<sl>*

Content models for *<sort-as>* (301)

In this document type	`<sort-as>` contains
Base map, Base topic	(text data | *<text>* | *<keyword>*)*

In this document type	`<sort-as>` is contained by
Base map	*<alt>*, *<anchorref>*, *<author>*, *<b>*, *<brand>*, *<category>*, *<cite>*, *<component>*, *<consequence>*, *<coords>*, *<copyrholder>*, *<data>*, *<data-about>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<dl>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<featnum>*, *<fig>*, *<figgroup>*, *<fn>*, *<i>*, *<index-base>*, *<index-see>*, *<index-see-also>*, *<index-sort-as>*, *<indexterm>*, *<itemgroup>*, *<keydef>*, * *, *<line-through>*, *<lines>*, *<linktext>*, *<lq>*, *<map>*, *<mapref>*, *<messagepanel>*, *<metadata>*, *<navtitle>*, *<note>*, * *, *<overline>*, *<p>*, *<ph>*, *<platform>*, *<pre>*, *<prodname>*, *<prognum>*, *<publisher>*, *<q>*, *<relcell>*, *<resourceid>*, *<searchtitle>*, *<series>*, *<shortdesc>*, *<sl>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<topicgroup>*, *<topichead>*, *<topicmeta>*, *<topicref>*, *<topicset>*, *<topicsetref>*, *<tt>*, *<typeofhazard>*, *<u>*, * *, *<xref>*
Base topic	**, *<alt>*, *<author>*, *<b>*, *<body>*, *<bodydiv>*, *<brand>*, *<category>*, *<cite>*, *<component>*, *<consequence>*, *<coords>*, *<copyrholder>*, *<data>*, *<data-about>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<dl>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<featnum>*, *<fig>*, *<figgroup>*, *<fn>*, *<i>*, *<index-base>*, *<index-see>*, *<index-see-also>*, *<index-sort-as>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<linktext>*, *<lq>*, *<messagepanel>*, *<metadata>*, *<navtitle>*, *<note>*, * *, *<overline>*, *<p>*, *<ph>*, *<platform>*, *<pre>*, *<prodname>*, *<prognum>*, *<prolog>*, *<publisher>*, *<q>*, *<resourceid>*, *<searchtitle>*, *<section>*, *<sectiondiv>*, *<series>*, *<shortdesc>*, *<sl>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, * *, *<xref>*

Content models for *<source>* (270)

In this document type	`<source>` contains
Base map, Base topic	(text data | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<keyword>* | *<term>* | *<text>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>*)*

In this document type	`<source>` is contained by
Base map	*<topicmeta>*
Base topic	*<prolog>*

Content models for *<state>* (337)

In this document type	`<state>` contains
Base map, Base topic	EMPTY

In this document type	`<state>` is contained by
Base map	*`<b>`*, *`<cite>`*, *`<data>`*, *`<dd>`*, *`<ddhd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<dt>`*, *`<dthd>`*, *`<entry>`*, *`<figgroup>`*, *`<fn>`*, *`<howtoavoid>`*, *`<i>`*, *`<itemgroup>`*, *`<li>`*, *`<line-through>`*, *`<lines>`*, *`<lq>`*, *`<note>`*, *`<overline>`*, *`<p>`*, *`<ph>`*, *`<pre>`*, *`<q>`*, *`<shortdesc>`*, *`<sli>`*, *`<stentry>`*, *`<sub>`*, *`<sup>`*, *`<title>`*, *`<tt>`*, *`<u>`*, *`<xref>`*
Base topic	*`<abstract>`*, *`<b>`*, *`<bodydiv>`*, *`<cite>`*, *`<data>`*, *`<dd>`*, *`<ddhd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<dt>`*, *`<dthd>`*, *`<entry>`*, *`<example>`*, *`<figgroup>`*, *`<fn>`*, *`<howtoavoid>`*, *`<i>`*, *`<itemgroup>`*, *`<li>`*, *`<line-through>`*, *`<lines>`*, *`<linkinfo>`*, *`<lq>`*, *`<note>`*, *`<overline>`*, *`<p>`*, *`<ph>`*, *`<pre>`*, *`<q>`*, *`<section>`*, *`<sectiondiv>`*, *`<shortdesc>`*, *`<sli>`*, *`<stentry>`*, *`<sub>`*, *`<sup>`*, *`<title>`*, *`<tt>`*, *`<u>`*, *`<xref>`*

Content models for *`<stentry>`* *(232)*

In this document type	`<stentry>` contains
Base map, Base topic	(text data | *`<dl>`* | *`<div>`* | *`<fig>`* | *`<imagemap>`* | *`<image>`* | *`<lines>`* | *`<lq>`* | *`<note>`* | *`<hazardstatement>`* | *`<object>`* | *`<ol>`* | *`<p>`* | *`<pre>`* | *`<sl>`* | *`<ul>`* | *`<boolean>`* | *`<cite>`* | *`<keyword>`* | *`<ph>`* | *`<b>`* | *`<i>`* | *`<line-through>`* | *`<overline>`* | *`<sup>`* | *`<sub>`* | *`<tt>`* | *`<u>`* | *`<q>`* | *`<term>`* | *`<text>`* | *`<tm>`* | *`<xref>`* | *`<state>`* | *`<data>`* | *`<sort-as>`* | *`<data-about>`* | *`<foreign>`* | *`<unknown>`* | *`<draft-comment>`* | *`<fn>`* | *`<indextermref>`* | *`<indexterm>`* | *`<required-cleanup>`*)*

In this document type	`<stentry>` is contained by
Base map, Base topic	*`<sthead>`*, *`<strow>`*

Content models for *`<sthead>`* *(231)*

In this document type	`<sthead>` contains
Base map, Base topic	*`<stentry>`*+

In this document type	`<sthead>` is contained by
Base map, Base topic	*`<simpletable>`*

Content models for *`<strow>`* *(232)*

In this document type	`<strow>` contains
Base map, Base topic	*`<stentry>`**

In this document type	`<strow>` is contained by
Base map, Base topic	*`<simpletable>`*

Content models for *`<sub>`* *(295)*

In this document type	`<sub>` contains
Base map, Base topic	(text data	*`<boolean>`*	*`<cite>`*	*`<keyword>`*	*`<ph>`*	*`<b>`*	*`<i>`*	*`<line-through>`*	*`<overline>`*	*`<sup>`*	*`<sub>`*	*`<tt>`*	*`<u>`*	*`<q>`*	*`<term>`*	*`<text>`*	*`<tm>`*	*`<xref>`*	

In this document type	`<sub>` contains
	<state> | *<data>* | *<sort-as>* | *<data-about>* | *<draft-comment>* | *<foreign>* | *<unknown>* | *<required-cleanup>*)*

In this document type	`<sub>` is contained by
Base map	*<alt>*, *<b>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*
Base topic	**, *<alt>*, *<b>*, *<bodydiv>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<searchtitle>*, *<section>*, *<sectiondiv>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*

Content models for *<subjectCell>* (330)

None of the document type shells distributed with the base package use this element.

Content models for *<subjectHead>* (317)

None of the document type shells distributed with the base package use this element.

Content models for *<subjectHeadMeta>* (318)

None of the document type shells distributed with the base package use this element.

Content models for *<subjectRel>* (323)

None of the document type shells distributed with the base package use this element.

Content models for *<subjectRelHeader>* (322)

None of the document type shells distributed with the base package use this element.

Content models for *<subjectRelTable>* (321)

None of the document type shells distributed with the base package use this element.

Content models for *<subjectRole>* (323)

None of the document type shells distributed with the base package use this element.

Content models for *<subjectScheme>* (308)

None of the document type shells distributed with the base package use this element.

Content models for *<subjectdef>* (319)

None of the document type shells distributed with the base package use this element.

Content models for *<subjectref>* (324)

None of the document type shells distributed with the base package use this element.

Content models for *<sup>* (294)

In this document type	<sup> contains
Base map, Base topic	(text data | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<draft-comment>* | *<foreign>* | *<unknown>* | *<required-cleanup>*)*

In this document type	<sup> is contained by
Base map	*<alt>*, *<b>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*
Base topic	**, *<alt>*, *<b>*, *<bodydiv>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<searchtitle>*, *<section>*, *<sectiondiv>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*

C.18 T elements

Content models for elements beginning with "t".

Content models for *<table>* (222)

In this document type	<table> contains
Base map, Base topic	*<title>*?, *<desc>*?, *<tgroup>*+

In this document type	<table> is contained by
Base map	*<dd>*, *<div>*, *<draft-comment>*, *<itemgroup>*, * *, *<lq>*, *<note>*, *<p>*
Base topic	**, *<body>*, *<bodydiv>*, *<dd>*, *<div>*, *<draft-comment>*, *<example>*, *<itemgroup>*, * *, *<lq>*, *<note>*, *<p>*, *<section>*, *<sectiondiv>*

Content models for *<tbody>* (228)

In this document type	<tbody> contains
Base map, Base topic	*<row>*+

In this document type	<tbody> is contained by
Base map, Base topic	*<tgroup>*

Content models for *<term>* *(218)*

In this document type	`<term>` contains
Base map, Base topic	(text data | *<draft-comment>* | *<required-cleanup>* | *<text>* | *<tm>*)*

In this document type	`<term>` is contained by
Base map	*<alt>*, *<author>*, *<b>*, *<brand>*, *<category>*, *<cite>*, *<component>*, *<consequence>*, *<coords>*, *<copyrholder>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<featnum>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-base>*, *<index-see>*, *<index-see-also>*, *<index-sort-as>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<platform>*, *<pre>*, *<prodname>*, *<prognum>*, *<publisher>*, *<q>*, *<searchtitle>*, *<series>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*
Base topic	**, *<alt>*, *<author>*, *<b>*, *<bodydiv>*, *<brand>*, *<category>*, *<cite>*, *<component>*, *<consequence>*, *<coords>*, *<copyrholder>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<featnum>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-base>*, *<index-see>*, *<index-see-also>*, *<index-sort-as>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<platform>*, *<pre>*, *<prodname>*, *<prognum>*, *<publisher>*, *<q>*, *<searchtitle>*, *<section>*, *<sectiondiv>*, *<series>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*

Content models for *<text>* *(218)*

In this document type	`<text>` contains
Base map, Base topic	(text data | *<text>*)*

In this document type	`<text>` is contained by
Base map	*<alt>*, *<author>*, *<b>*, *<brand>*, *<category>*, *<cite>*, *<component>*, *<consequence>*, *<coords>*, *<copyrholder>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<dvrKeyscopePrefix>*, *<dvrKeyscopeSuffix>*, *<dvrResourcePrefix>*, *<dvrResourceSuffix>*, *<entry>*, *<featnum>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-base>*, *<index-see>*, *<index-see-also>*, *<index-sort-as>*, *<indexterm>*, *<itemgroup>*, *<keyword>*, * *, *<line-through>*, *<lines>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<platform>*, *<pre>*, *<prodname>*, *<prognum>*, *<publisher>*, *<q>*, *<searchtitle>*, *<series>*, *<shape>*, *<shortdesc>*, *<sli>*, *<sort-as>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<term>*, *<text>*, *<title>*, *<tm>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*
Base topic	**, *<alt>*, *<author>*, *<b>*, *<bodydiv>*, *<brand>*, *<category>*, *<cite>*, *<component>*, *<consequence>*, *<coords>*, *<copyrholder>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<featnum>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-base>*, *<index-see>*, *<index-see-also>*, *<index-sort-as>*, *<indexterm>*, *<itemgroup>*, *<keyword>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<platform>*, *<pre>*, *<prodname>*, *<prognum>*, *<publisher>*,

In this document type	`<text>` is contained by
	<q>, *<searchtitle>*, *<section>*, *<sectiondiv>*, *<series>*, *<shape>*, *<shortdesc>*, *<sli>*, *<sort-as>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<term>*, *<text>*, *<title>*, *<tm>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*

Content models for *<tgroup>* (227)

In this document type	`<tgroup>` contains
Base map, Base topic	*<colspec>**, *<thead>*?, *<tbody>*

In this document type	`<tgroup>` is contained by
Base map, Base topic	*<table>*

Content models for *<thead>* (228)

In this document type	`<thead>` contains
Base map, Base topic	*<row>*+

In this document type	`<thead>` is contained by
Base map, Base topic	*<tgroup>*

Content models for *<title>* (182)

In this document type	`<title>` contains
Base map, Base topic	(text data | *<boolean>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<state>* | *<cite>* | *<data>* | *<sort-as>* | *<data-about>* | *<draft-comment>* | *<foreign>* | *<unknown>* | *<required-cleanup>* | *<image>*)*

In this document type	`<title>` is contained by
Base map	*<data>*, *<fig>*, *<figgroup>*, *<map>*, *<relcolspec>*, *<reltable>*, *<table>*
Base topic	*<data>*, *<example>*, *<fig>*, *<figgroup>*, *<linklist>*, *<section>*, *<table>*, *<topic>*

Content models for *<titlealts>* (183)

In this document type	`<titlealts>` contains
Base topic	*<navtitle>*?, *<searchtitle>*?

In this document type	`<titlealts>` is contained by
Base topic	*<topic>*

Content models for *<tm>* (219)

In this document type	`<tm>` contains
Base map, Base topic	(text data | *<text>* | *<tm>*)*

In this document type	`<tm>` is contained by
Base map	*<b>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<itemgroup>*, *<keyword>*, * *, *<line-through>*, *<lines>*, *<lq>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<shortdesc>*, *<sli>*, *<stentry>*, *<sub>*, *<sup>*, *<term>*, *<title>*, *<tm>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*
Base topic	**, *<b>*, *<bodydiv>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<itemgroup>*, *<keyword>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<lq>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<section>*, *<sectiondiv>*, *<shortdesc>*, *<sli>*, *<stentry>*, *<sub>*, *<sup>*, *<term>*, *<title>*, *<tm>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*

Content models for *<topic>* (182)

In this document type	`<topic>` contains
Base topic	*<title>*, *<titlealts>*?, (*<shortdesc>* | **)?, *<prolog>*?, *<body>*?, *<related-links>*?, *<topic>**

In this document type	`<topic>` is contained by
Base topic	*<foreign>*, *<required-cleanup>*, *<topic>*, *<unknown>*

Content models for *<topicapply>* (325)

None of the document type shells distributed with the base package use this element.

Content models for *<topicCell>* (330)

None of the document type shells distributed with the base package use this element.

Content models for *<topicgroup>* (252)

In this document type	`<topicgroup>` contains
Base map	*<topicmeta>*?, (*<anchor>* | *<data>* | *<sort-as>* | *<data-about>* | *<navref>* | *<topicref>* | *<anchorref>* | *<keydef>* | *<mapref>* | *<topicgroup>* | *<topichead>* | *<topicset>* | *<topicsetref>* | *<ditavalref>*)*

In this document type	`<topicgroup>` is contained by
Base map	*<anchorref>*, *<keydef>*, *<map>*, *<relcell>*, *<relcolspec>*, *<topicgroup>*, *<topichead>*, *<topicref>*, *<topicset>*, *<topicsetref>*

Content models for *<topichead>* *(253)*

In this document type	**<topichead> contains**
Base map	*<topicmeta>*?, (*<anchor>* | *<data>* | *<sort-as>* | *<data-about>* | *<navref>* | *<topicref>* | *<anchorref>* | *<keydef>* | *<mapref>* | *<topicgroup>* | *<topichead>* | *<topicset>* | *<topicsetref>* | *<ditavalref>*)*

In this document type	**<topichead> is contained by**
Base map	*<anchorref>*, *<keydef>*, *<map>*, *<relcell>*, *<relcolspec>*, *<topicgroup>*, *<topichead>*, *<topicref>*, *<topicset>*, *<topicsetref>*

Content models for *<topicmeta>* *(239)*

In this document type	**<topicmeta> contains**
Base map	*<navtitle>*?, *<linktext>*?, *<searchtitle>*?, *<shortdesc>*?, *<author>**, *<source>*?, *<publisher>*?, *<copyright>**, *<critdates>*?, *<permissions>*?, *<metadata>**, *<audience>**, *<category>**, *<keywords>*, *<exportanchors>* *<exportanchors>**, *<prodinfo>**, *<othermeta>**, *<resourceid>**, *<ux-window>**, (*<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>*)*

In this document type	**<topicmeta> is contained by**
Base map	*<anchorref>*, *<keydef>*, *<map>*, *<mapref>*, *<relcolspec>*, *<reltable>*, *<topicgroup>*, *<topichead>*, *<topicref>*, *<topicset>*, *<topicsetref>*

Content models for *<topicref>* *(238)*

In this document type	**<topicref> contains**
Base map	*<topicmeta>*?, (*<anchor>* | *<data>* | *<sort-as>* | *<data-about>* | *<navref>* | *<topicref>* | *<anchorref>* | *<keydef>* | *<mapref>* | *<topicgroup>* | *<topichead>* | *<topicset>* | *<topicsetref>* | *<ditavalref>*)*

In this document type	**<topicref> is contained by**
Base map	*<anchorref>*, *<keydef>*, *<map>*, *<relcell>*, *<relcolspec>*, *<topicgroup>*, *<topichead>*, *<topicref>*, *<topicset>*, *<topicsetref>*

Content models for *<topicset>* *(253)*

In this document type	**<topicset> contains**
Base map	*<topicmeta>*?, (*<anchor>* | *<data>* | *<sort-as>* | *<data-about>* | *<navref>* | *<topicref>* | *<anchorref>* | *<keydef>* | *<mapref>* | *<topicgroup>* | *<topichead>* | *<topicset>* | *<topicsetref>* | *<ditavalref>*)*

In this document type	**<topicset> is contained by**
Base map	*<anchorref>*, *<keydef>*, *<map>*, *<relcell>*, *<relcolspec>*, *<topicgroup>*, *<topichead>*, *<topicref>*, *<topicset>*, *<topicsetref>*

Content models for *<topicsetref>* *(254)*

In this document type	*<topicsetref>* contains
Base map	*<topicmeta>*?, (*<data>* | *<sort-as>* | *<data-about>* | *<topicref>* | *<anchorref>* | *<keydef>* | *<mapref>* | *<topicgroup>* | *<topichead>* | *<topicset>* | *<topicsetref>* | *<ditavalref>*)*

In this document type	*<topicsetref>* is contained by
Base map	*<anchorref>*, *<keydef>*, *<map>*, *<relcell>*, *<relcolspec>*, *<topicgroup>*, *<topichead>*, *<topicref>*, *<topicset>*, *<topicsetref>*

Content models for *<topicsubject>* *(326)*

None of the document type shells distributed with the base package use this element.

Content models for *<topicSubjectHeader>* *(329)*

None of the document type shells distributed with the base package use this element.

Content models for *<topicSubjectRow>* *(329)*

None of the document type shells distributed with the base package use this element.

Content models for *<topicSubjectTable>* *(327)*

None of the document type shells distributed with the base package use this element.

Content models for *<tt>* *(295)*

In this document type	*<tt>* contains
Base map, Base topic	(text data | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<draft-comment>* | *<foreign>* | *<unknown>* | *<required-cleanup>*)*

In this document type	*<tt>* is contained by
Base map	*<alt>*, *<b>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*
Base topic	**, *<alt>*, *<b>*, *<bodydiv>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<searchtitle>*, *<section>*, *<sectiondiv>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*

Content models for *<typeofhazard>* (293)

In this document type	`<typeofhazard>` contains
Base map, Base topic	(text data | *<data>* | *<sort-as>* | *<data-about>* | *<foreign>* | *<unknown>* | *<keyword>* | *<term>* | *<text>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<tm>*)*

In this document type	`<typeofhazard>` is contained by
Base map, Base topic	*<messagepanel>*

C.19 U elements

Content models for elements beginning with "u".

Content models for *<u>* (296)

In this document type	`<u>` contains
Base map, Base topic	(text data | *<boolean>* | *<cite>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<xref>* | *<state>* | *<data>* | *<sort-as>* | *<data-about>* | *<draft-comment>* | *<foreign>* | *<unknown>* | *<required-cleanup>*)*

In this document type	`<u>` is contained by
Base map	*<alt>*, *<b>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*
Base topic	**, *<alt>*, *<b>*, *<bodydiv>*, *<cite>*, *<consequence>*, *<data>*, *<dd>*, *<ddhd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<dthd>*, *<entry>*, *<example>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<index-see>*, *<index-see-also>*, *<indexterm>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<linktext>*, *<lq>*, *<navtitle>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<searchtitle>*, *<section>*, *<sectiondiv>*, *<shortdesc>*, *<sli>*, *<source>*, *<stentry>*, *<sub>*, *<sup>*, *<title>*, *<tt>*, *<typeofhazard>*, *<u>*, *<xref>*

Content models for *<ul>* (219)

In this document type	`<ul>` contains
Base map, Base topic	(*<data>* | *<sort-as>* | *<data-about>*)*, * *+

In this document type	`<ul>` is contained by
Base map	*<dd>*, *<desc>*, *<div>*, *<draft-comment>*, *<entry>*, *<fig>*, *<figgroup>*, *<fn>*, *<itemgroup>*, * *, *<lq>*, *<note>*, *<p>*, *<stentry>*

In this document type	`<ul>` is contained by
Base topic	*`<abstract>`*, *`<body>`*, *`<bodydiv>`*, *`<dd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<entry>`*, *`<example>`*, *`<fig>`*, *`<figgroup>`*, *`<fn>`*, *`<itemgroup>`*, *`<li>`*, *`<linkinfo>`*, *`<lq>`*, *`<note>`*, *`<p>`*, *`<section>`*, *`<sectiondiv>`*, *`<stentry>`*

Content models for *`<unknown>`* *(337)*

In this document type	`<unknown>` contains
Base map, Base topic	Any defined element

In this document type	`<unknown>` is contained by
Base map	*`<alt>`*, *`<author>`*, *`<b>`*, *`<brand>`*, *`<category>`*, *`<cite>`*, *`<component>`*, *`<consequence>`*, *`<coords>`*, *`<copyrholder>`*, *`<data>`*, *`<dd>`*, *`<ddhd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<dt>`*, *`<dthd>`*, *`<entry>`*, *`<featnum>`*, *`<fig>`*, *`<figgroup>`*, *`<fn>`*, *`<i>`*, *`<index-base>`*, *`<index-see>`*, *`<index-see-also>`*, *`<index-sort-as>`*, *`<indexterm>`*, *`<itemgroup>`*, *`<li>`*, *`<line-through>`*, *`<lines>`*, *`<linktext>`*, *`<lq>`*, *`<metadata>`*, *`<navtitle>`*, *`<note>`*, *`<object>`*, *`<overline>`*, *`<p>`*, *`<ph>`*, *`<platform>`*, *`<pre>`*, *`<prodname>`*, *`<prognum>`*, *`<publisher>`*, *`<q>`*, *`<searchtitle>`*, *`<series>`*, *`<shortdesc>`*, *`<sli>`*, *`<source>`*, *`<stentry>`*, *`<sub>`*, *`<sup>`*, *`<title>`*, *`<topicmeta>`*, *`<tt>`*, *`<typeofhazard>`*, *`<u>`*, *`<xref>`*
Base topic	*`<abstract>`*, *`<alt>`*, *`<author>`*, *`<b>`*, *`<body>`*, *`<bodydiv>`*, *`<brand>`*, *`<category>`*, *`<cite>`*, *`<component>`*, *`<consequence>`*, *`<coords>`*, *`<copyrholder>`*, *`<data>`*, *`<dd>`*, *`<ddhd>`*, *`<desc>`*, *`<div>`*, *`<draft-comment>`*, *`<dt>`*, *`<dthd>`*, *`<entry>`*, *`<example>`*, *`<featnum>`*, *`<fig>`*, *`<figgroup>`*, *`<fn>`*, *`<i>`*, *`<index-base>`*, *`<index-see>`*, *`<index-see-also>`*, *`<index-sort-as>`*, *`<indexterm>`*, *`<itemgroup>`*, *`<li>`*, *`<line-through>`*, *`<lines>`*, *`<linkinfo>`*, *`<linktext>`*, *`<lq>`*, *`<metadata>`*, *`<navtitle>`*, *`<note>`*, *`<object>`*, *`<overline>`*, *`<p>`*, *`<ph>`*, *`<platform>`*, *`<pre>`*, *`<prodname>`*, *`<prognum>`*, *`<prolog>`*, *`<publisher>`*, *`<q>`*, *`<searchtitle>`*, *`<section>`*, *`<sectiondiv>`*, *`<series>`*, *`<shortdesc>`*, *`<sli>`*, *`<source>`*, *`<stentry>`*, *`<sub>`*, *`<sup>`*, *`<title>`*, *`<tt>`*, *`<typeofhazard>`*, *`<u>`*, *`<xref>`*

Content models for *`<ux-window>`* *(247)*

In this document type	`<ux-window>` contains
Base map	EMPTY

In this document type	`<ux-window>` is contained by
Base map	*`<topicmeta>`*

C.20 V elements

Content models for elements beginning with "v".

Content models for *`<vrm>`* *(271)*

In this document type	`<vrm>` contains
Base map, Base topic	EMPTY

In this document type	`<vrm>` is contained by
Base map, Base topic	*<vrmlist>*

Content models for *<vrmlist>* (270)

In this document type	`<vrmlist>` contains
Base map, Base topic	*<vrm>*+

In this document type	`<vrmlist>` is contained by
Base map, Base topic	*<prodinfo>*

C.21 X elements

Content models for elements beginning with "x".

Content models for *<xref>* (220)

In this document type	`<xref>` contains
Base map, Base topic	(text data | *<boolean>* | *<keyword>* | *<ph>* | *<b>* | *<i>* | *<line-through>* | *<overline>* | *<sup>* | *<sub>* | *<tt>* | *<u>* | *<q>* | *<term>* | *<text>* | *<tm>* | *<state>* | *<cite>* | *<data>* | *<sort-as>* | *<data-about>* | *<draft-comment>* | *<foreign>* | *<unknown>* | *<required-cleanup>* | *<image>* | *<desc>*)*

In this document type	`<xref>` is contained by
Base map	*<area>*, *<b>*, *<data>*, *<dd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<entry>*, *<fig>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<lq>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<shortdesc>*, *<sli>*, *<stentry>*, *<sub>*, *<sup>*, *<tt>*, *<u>*
Base topic	**, *<area>*, *<b>*, *<bodydiv>*, *<data>*, *<dd>*, *<desc>*, *<div>*, *<draft-comment>*, *<dt>*, *<entry>*, *<example>*, *<fig>*, *<figgroup>*, *<fn>*, *<howtoavoid>*, *<i>*, *<itemgroup>*, * *, *<line-through>*, *<lines>*, *<linkinfo>*, *<lq>*, *<note>*, *<overline>*, *<p>*, *<ph>*, *<pre>*, *<q>*, *<section>*, *<sectiondiv>*, *<shortdesc>*, *<sli>*, *<stentry>*, *<sub>*, *<sup>*, *<tt>*, *<u>*