



DITA Release 1.1 Architectural Specification

Committee Draft, 13 February 2007

- **Document identifier:**

DITA_v1.1ArchSpecificationIssue01

- **Locations:**

This Version:

<http://docs.oasis-open.org/dita/v1.1/CD01/archspec/archspec.html>

<http://docs.oasis-open.org/dita/v1.1/CD01/archspec/archspec.pdf>

<http://docs.oasis-open.org/dita/v1.1/CD01/archspec/archspec.zip> (DITA source; see archspec.ditamap)

Previous Version:

<http://docs.oasis-open.org/dita/v1.0/archspec/ditaspec.toc.html>

<http://docs.oasis-open.org/dita/v1.0/dita-v1.0-spec-os-ArchitecturalSpecification.pdf>

Latest Version:

<http://docs.oasis-open.org/dita/v1.1/CD01/archspec/archspec.html>

<http://docs.oasis-open.org/dita/v1.1/CD01/archspec/archspec.pdf>

<http://docs.oasis-open.org/dita/v1.1/CD01/archspec/archspec.zip> (DITA source; see archspec.ditamap)

- **Technical Committee:**

OASIS Darwin Information Typing Architecture (DITA) TC

Chair(s):

Don Day

Editor(s):

Michael Priestley

JoAnn Hackos

- **Related Work:**

This specification replaces or supercedes:

OASIS DITA Language Specification Version 1.0

This specification is related to:

OASIS DITA Language Specification Version 1.1 Committee Draft

(<http://docs.oasis-open.org/dita/v1.1/CD01/langspec/ditaref-type.html>)

OASIS DITA DTDs Committee Draft Version (1.1 <http://docs.oasis-open.org/dita/v1.1/CD01/dtd/ditadtd.zip>)

OASIS DITA Schemas Committee Draft Version 1.1

(<http://docs.oasis-open.org/dita/v1.1/CD01/schema/ditaschema.zip>)

Declared XML Namespace(s):

[none]

- **Abstract:**

The Darwin Information Typing Architecture (DITA) 1.1 specification defines both a) a set of document types for authoring and organizing topic-oriented information; and b) a set of mechanisms for combining and extending document types using a process called specialization.

- **Status:**

This document was last revised or approved by the Darwin Information Typing Architecture (DITA) TC on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at

http://www.oasis-open.org/committees/tc_home.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 Technical Committee web page (<http://www.oasis-open.org/committees/dita/ipr.php>).

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dita.

Notices

Copyright © OASIS® 1993-2007. All Rights Reserved. OASIS trademark, IPR and other policies apply. 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](http://www.oasis-open.org), 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 <http://www.oasis-open.org/who/trademark.php> for above guidance.

Contents

Chapter 1. About the DITA 1.1 Specification. 1

Chapter 2. An introduction to DITA 5

Definitions and background concepts	5
Terminology	6
Naming conventions and file extensions	8
DTD organization	8
Public identifiers for DTDs	10
XML Schema organization	11
XML Schema catalog identifiers	12

Chapter 3. DITA markup 15

DITA topics	15
What are topics?	15
Why topics?	16
Information typing	16
Transitional text	16
Generic topics	18
Concepts	20
Tasks	20
Reference	22
Glossary	23
Topic domains	24
DITA maps	24
What are maps?	24
Why DITA maps?	25
Common DITA map attributes and metadata	25
DITA map structure	28
DITA map modules.	29
Inheritance of attributes and metadata in maps	29
Bookmap	30
Map domains.	33
Metadata elements and common attributes	33
Common metadata elements.	34
Common attributes.	35
Topic properties in topics and maps	37

Chapter 4. DITA processing 41

IDs and references	41
Navigation behaviors	42
Content inclusion (conref)	42
Conditional processing (profiling)	43
Conditional processing attributes	44

Using metadata attributes	45
Processing metadata attributes	45
Filtering logic.	45
Flagging logic	45
Chunking	46
Usage of the chunk attribute.	46
Translation	49
The xml:lang attribute.	49
The dir attribute.	51
All elements with translation properties	53

Chapter 5. DITA specialization. 65

What is specialization?	65
Why specialization?	66
Structural versus domain specialization	66
Specializing foreign or unknown content	67
Data extensibility	69
Limits of specialization	70
Specialize from generic elements or attributes	70
Customized subset document types for authoring	71
Map from customized document type to DITA	72
during preprocessing	72
Specialization in content	72
Why specialization in content?	72
Specialization of attributes	72
The class attribute	73
Class attribute syntax	73
The domains attribute	74
Specialization validity	75
Generalization	75
Specialization in design	79
Why specialization in design?	79
Modularization and integration of design	79
Integration	79
Modularization in DTDs	80
Modularization in schemas	83
Specialization in processing	87
Using the class attribute	87
Processing specialized attributes	88
Processing foreign content	88
Modularization and integration of processing	88
Customization	88
Modularization in CSS.	89
Modularization in XSLT	90

Chapter 1. About the DITA 1.1 Specification

The Darwin Information Typing Architecture (DITA) 1.1 specification defines both a) a set of document types for authoring and organizing topic-oriented information; and b) a set of mechanisms for combining and extending document types using a process called specialization.

The specification consists of:

- The DTDs and schemas that define DITA markup for the base DITA document types, as well as catalog files. While the DTDs and schemas should define the same DITA elements, the DTDs are normative if there is ever any discrepancy.
- The language reference that provides explanations for each element in the base DITA document types
- This document, which has the following parts:
 - an introduction, which provides background concepts and an overview of the architecture
 - the DITA markup specification, which provides an overview of DITA's base document types
 - the DITA processing specification, which provides descriptions of common DITA processing expectations
 - the DITA specialization specification, which provides details of the mechanisms DITA provides for defining and extending DITA document types.

This document is part of the technical specification for the DITA architecture. While the specification does contain some introductory information, it is not intended as an introduction to DITA nor as a users guide. The intended audience of this specification consists of implementers of the DITA standard, including tool developers and specializers.

Changes since DITA 1.0

The DITA 1.1 specification is designed to be backwards-compatible with 1.0-conforming applications. The following major changes to the architecture provide added functionality for 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

Editors

Michael Priestley and JoAnn Hackos

Members of the OASIS DITA Technical Committee

Robert Anderson, IBM
Paul Antonov
France Baril, IXIASOFT
Paul Barley, IBM
Bob Bernard, IBM

Kylene Bruski, Comtech Services, Inc.
Sissi Closs, Comet Communication
Robin Cover, OASIS
Don Day, IBM
Greg DeVore, Blue Mango Learning Systems
Stanley Doherty, Sun Microsystems
Bruce Esrig, Lucent Technologies
Yas Etessam, XMetal
Kay Ethier
Alice Etim, IBM
Kevin Farwell, Lionbridge
Klaus Fenchel, Ovidius
Rob Frankland, Rascal Software
Michael Gannon, XMetal
Anne Gentle, BMC Software
Gerald Goetz, Noxum GmbH
Paul Grosso, PTC/Arbortext
JoAnn Hackos, Comtech Services, Inc.
Nancy Harrison, IBM
Erik Hennum, IBM
Eric Hixson, BMC Software
Alan Houser
Scott Hudson, Flatirons Solutions Corporation
John Hunt, IBM
Gershon Joseph, Tech-Tav Documentation Ltd.
Eliot Kimber, Innodata Isogen
Chris Kravogel
Seraphim Larsen, Intel Corporation
Indi Liepa, Nokia Corporation
Jennifer Linton, Gambro BCT
Steven Manning, Rockley Group
Glen Mules, IBM
Sukumar Munshi, Lionbridge
Ultan Obroin, Oracle Corporation
Frank Peters, Sun Microsystems
Deborah Pickett, Moldflow Corporation
Mark Poston, Mekon Limited
Alex Povzner, SiberLogic, Inc.
Paul Prescod, XMetaL
Michael Priestley, IBM
Rodolfo Raya, Heartsome Holdings Pte Ltd.
Matthias Rehsoeft, Gesellschaft fur technische Kommunikation-tekomp
Nick Rosenthal
David Schell, IBM
Roland Schmeling

Anthony Self, Hyperwrite Pty. Ltd.
Punyanjan Sen, Adobe Systems
Wendy Shepperd, BMC Software
Peter Shepton, Intel Corporation
Eric Sirois, IBM
Jerry Smith, US Department of Defense (DoD)
Dana Spradley, Oracle Corporation
Robert Stayton
Amber Swope, XMetaL
Mark Tiegs, Research In Motion
Armando Torres, Comtech Services, Inc.
Scott Tsao, The Boeing Company
Sharon Veach, Sun Microsystems
Kate Wilhelm, Research in Motion
Christopher Wong, Idiom Technologies, Inc
Andrzej Zydron, XML Intl.

Special thanks

Special thanks to Robert Anderson of IBM for his work on improving the language reference through document generation processes and through editing of topics to incorporate numerous review comments.

Special thanks to Jeff Ogden of PTC for his numerous detailed reviews of both the language specification and architectural specification, which has resulted in a large number of editorial improvements.

History

2004-04-12	OASIS DITA Technical Committee formed with initial submission of materials from IBM
2005-06-01	OASIS DITA 1.0 specification approved by OASIS

Chapter 2. An introduction to DITA

DITA is an architecture for creating topic-oriented, information-typed content that can be reused and single-sourced in a variety of ways. It is also an architecture for creating new topic types and describing new information domains based on existing types and domains.

The process for creating new topic types and domains is called specialization. Specialization allows the creation of very specific, targeted document type definitions while still sharing common output transforms and design rules developed for more general types and domains, in much the same way that classes in an object-oriented system can inherit methods of ancestor classes.

DITA topics are XML conforming. As such, they are readily viewed, edited, and validated with standard XML tools, although some features such as content referencing and specialization may benefit from customized support.

Definitions and background concepts

The following terms have specific meanings in DITA which should be understood before reading either the DITA markup specification or the DITA specialization specification.

Basic concepts

The following are basic concepts used in DITA.

“What are topics?” on page 15

A topic is a unit of information with a title and some form of content, short enough to be specific to a single subject or answer a single question, but long enough to make sense on its own and be authored as a unit.

“What are maps?” on page 24

DITA maps are documents that collect and organize references to DITA topics to indicate the relationships among the topics. They can also serve as outlines or tables of contents for DITA deliverables and as build manifests for DITA projects.

“What is specialization?” on page 65

Specialization allows you 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.

“Structural versus domain specialization” on page 66

Structural specialization defines new types of structured information, such as new topic types or new map types. Domain specialization creates new markup that can be useful in multiple structural types, such as new kinds of keywords, tables, or lists, or new attributes such as conditional processing attributes.

“Integration” on page 79

Each domain specialization or structural specialization has its own design module. These modules can be combined to create many different document types. The process of creating a new document type from a specific combination of modules is called integration.

“Customization” on page 88

When you just need a difference in output, you can use DITA customization to override the default output without affecting portability or interchange, and without involving specialization.

“Generalization” on page 75

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.

Terminology

DITA uses a number of terms in particular or unique ways. Within the scope of this specification, the following terms are used when talking about DITA models, DITA declarations, and DITA instances.

When particular attributes are listed together with elements, attribute names may be preceded by @ to distinguish them from elements. For example: @props, @class.

When particular elements are named outside of a list, they may be delimited with < and > to distinguish them from surrounding text. For example: <keyword>, <prolog>.

Model terminology

DITA can be understood at the level of an abstract model without reference to particular DTDs, schemas, or actual XML documents. When discussing DITA concepts at this level, the following terminology is used.

Element type

Defines the structure and semantics for a fragment of content.

Specialized element type

Defines an element type as a semantic refinement of another element type. The content allowed by the specialized element type must be a subset of or identical to the content allowed by the original element type.

Attribute type

Defines the structure and semantics for an attribute.

Specialized attribute type

Defines an attribute type as a semantic refinement of another attribute type. The attribute must specialize the base or props attribute, and its content must be a subset of or identical to the content allowed by the original attribute type.

Topic type

An element type that defines a complete unit of content. The topic type provides the root element for the topic and, through contained element types, substructure for the topic instances. The root element of the topic type is not necessarily the same as the root element of a document type: document types may nest multiple topic types, and may also declare non-DITA wrapper elements as the root element for compatibility with other processes.

Map type

An element type that defines a set of relationships for topic instances. The map type provides the root element and, through contained element types, substructure for the map instances. The map substructure provides hierarchy, group, and matrix organization of references to topic instances.

Structural type

A topic type or map type.

Domain

A set of elements or an attribute that supports a specific subject area. Elements or attributes in a domain can be integrated with topic or map types to enhance their semantic support for particular kinds of content. For example, the structural type <topic> declares the <keyword> element; when integrated with a domain for describing user interfaces, new keyword specializations (such as <wintitle>) become available wherever <keyword> was allowed in the original structural type.

Document type

The full set of element types and attribute types defined in the modules that are integrated by the document type shell. A DITA document type may support authoring multiple topic types or multiple map types, but not a mix of the two. The structural types can be augmented with elements from domains. The term "document type" is used for compatibility with existing

standards, since this is the point at which DITA's set of topic, domain, and map types are assembled into a document type that is functionally equivalent to a traditional non-modularized document type.

Declaration terminology

When the model is expressed in a DTD or schema, the various element types are declared. When referring to these declarations, the following terminology is used.

Element declaration

The representation within a schema technology (such as DTD, XML Schema, or Relax NG) for an element type.

Attribute declaration

The representation within a schema technology (such as DTD, XML Schema, or Relax NG) for an attribute type.

Type module

The representation within a schema technology for the element and attribute types uniquely defined by a topic type, map type, or domain.

Topic module

The representation within a schema technology for the element types uniquely defined by a topic type.

Map module

The representation within a schema technology for the element types uniquely defined by a map type.

Structural module

A topic or map module.

Domain module

The representation within a schema technology for the element types or attribute type uniquely defined by a domain.

Document type shell, head schema

The representation within a schema technology for a DTD shell or head schema that declares no element or attribute types itself (except potentially for a <dita> root element that allows multiple peer topic types in the same document type) but points to and assembles topic and domain modules or map and domain modules.

Document type declaration

The representation within a schema technology for a document type. The document type declaration includes the declaration modules assembled by the document declaration shell.

Instance terminology

When actual documents, topics, and elements are created based on a DITA document type, the following terminology is used.

Element instance

An occurrence of an element type in a document.

Attribute instance

An occurrence of an attribute type in a document.

Topic instance

An occurrence of a topic type in a document.

Map instance

An occurrence of a map type in a document.

Structural type instance

An occurrence of a topic type or a map type in a document.

Document instance

A document whose meaning and validity are determined by a document type declaration.

Naming conventions and file extensions

The following naming conventions and file extensions are in use by DITA.

DITA topic instance files

**.dita* (recommended), **.xml*

DITA map instance files

**.ditamap*

DTD structural module files

typename.mod

DTD domain module files

typenameDomain.mod

typenameDomain.ent

Schema structural module files

typenameMod.xsd

typenameGrp.xsd

Schema domain module files

typenameDomain.xsd

CSS override files (recommended convention for tool providers)

typename.css

customization-purpose.css

XSLT override files (recommended convention for tool providers)

typename.xsl

customization-purpose.xsl

Conditional processing profiles

profilename.ditaval

DTD organization

The OASIS DITA document types are implemented with a set of DTD modules. Some of these modules are used by every DITA document type; others are only used by topics or by maps, and some are only used in specific specializations.

DTDs versus Modules

A significant feature of the DITA implementation is that it places more importance on the modules than on the actual DTD. All element and attribute type declarations are made in modules, which are then integrated into a document type using a document type shell. Implementors are free to create new DTDs that reorganize the modules, introduce new modules, or remove modules as appropriate. For example, the standard topic DTD from OASIS includes all of the standard topic domains; in addition, while the default topic DTD allows topics to nest, it is not possible to include concepts. A new DTD can change one or both of these features and still be valid; the DTD may add or remove domains, and it may allow topic to nest concepts, or allow authoring of different types at the same level, as in the database document type. When creating a new or changing an existing DTD, users should remember to give the new or changed DTD a new public ID, so that various DITA tools will not confuse the new DTD with the default OASIS version.

Description of DITA Modules

Each specialization requires one or more modules to define elements for that specialization. Any DTD that makes use of a specialization must include the modules for that specialization.

In addition to the modules specific to a specialization, there are several files that are used by every DITA DTD. These are included in the base topic or map modules, so they do not need to be referenced in the DTD – they are already included simply by including the topic or map module.

The tables below describe each of the DTD modules that come with the OASIS DITA standard.

Table 1. Description of topic and its specialization modules

Common module file	Purpose	Used within DTD
topic.mod	Define all elements that may be used within the base topic type.	Must be included in any topic-based DITA DTD.
concept.mod	Define elements used within the concept specialization.	concept.dtd, ditabase.dtd
glossary.mod	Define elements used within the glossary specialization	glossary.dtd, ditabase.dtd
reference.mod	Define elements used within the reference specialization.	reference.dtd, ditabase.dtd
task.mod	Define elements used within the task specialization.	task.dtd, ditabase.dtd

Table 2. Description of map and its specialization modules

Common module file	Purpose	Used within DTD
map.mod	Define all elements that may be used within the base map type.	Must be included in any map-based DITA DTD.
bookmap.mod	Define elements used within the bookmap specialization.	bookmap.dtd

Table 3. Description of domain modules

Common module file	Purpose	Used within DTD
indexingDomain.ent, indexingDomain.mod	Define entities and elements used by the indexing domain.	bookmap.dtd, concept.dtd, ditabase.dtd, glossary.dtd, map.dtd reference.dtd, task.dtd, topic.dtd
highlightDomain.ent, highlightDomain.mod	Define entities and elements used by the highlighting domain.	concept.dtd, ditabase.dtd, glossary.dtd, reference.dtd, task.dtd, topic.dtd
programmingDomain.ent, programmingDomain.ent	Define entities and elements used by the programming domain.	concept.dtd, ditabase.dtd, glossary.dtd, reference.dtd, task.dtd, topic.dtd
softwareDomain.ent, softwareDomain.mod	Define entities and elements used by the software domain.	concept.dtd, ditabase.dtd, glossary.dtd, reference.dtd, task.dtd, topic.dtd
uiDomain.ent, uiDomain.mod	Define entities and elements used by the user interface domain.	concept.dtd, ditabase.dtd, glossary.dtd, reference.dtd, task.dtd, topic.dtd
utilitiesDomain.ent, utilitiesDomain.mod	Define entities and elements used by the utilities domain.	concept.dtd, ditabase.dtd, glossary.dtd, reference.dtd, task.dtd, topic.dtd

Table 3. Description of domain modules (continued)

Common module file	Purpose	Used within DTD
mapGroup.ent, mapGroup.mod	Define entities and elements used by the map group domain.	map.dtd, bookmap.dtd
xnalDomain.ent, xnalDomain.mod	Define entities and elements used by the XNAL domain.	bookmap.dtd

Table 4. Description of other common modules used by the files above

Common module file	Purpose	Used within module
commonElements.ent, commonElements.mod	Define all content elements that may appear in both maps and topics.	topic.mod, map.mod (this makes it a part of any DITA DTD)
metaDecl.mod	Define meta elements that may appear in both maps and topics	topic.mod, map.mod (this makes it a part of any DITA DTD)
tblDecl.mod	Defines the complex tables used within DITA, based on the OASIS Exchange Table model.	commonElements.mod (this makes it a part of any DITA DTD)
topicDefn.ent	Defines entities used within the topic module.	topic.mod (this makes it a part of any topic-based DITA DTD)

Public identifiers for DTDs

Each standard DTD has a public identifier that can be used to reference either the latest version or a specific version of the DTD.

PUBLIC "-//OASIS//DTD DITA Concept//EN" "concept.dtd"

PUBLIC "-//OASIS//DTD DITA 1.1 Concept//EN" "concept.dtd"

PUBLIC "-//OASIS//DTD DITA 1.0 Concept//EN" "concept.dtd"

PUBLIC "-//OASIS//DTD DITA Composite//EN" "database.dtd"

PUBLIC "-//OASIS//DTD DITA 1.1 Composite//EN" "database.dtd"

PUBLIC "-//OASIS//DTD DITA 1.0 Composite//EN" "database.dtd"

PUBLIC "-//OASIS//DTD DITA Glossary//EN" "glossary.dtd"

PUBLIC "-//OASIS//DTD DITA 1.1 Glossary//EN" "glossary.dtd"

PUBLIC "-//OASIS//DTD DITA Reference//EN" "reference.dtd"

PUBLIC "-//OASIS//DTD DITA 1.1 Reference//EN" "reference.dtd"

PUBLIC "-//OASIS//DTD DITA 1.0 Reference//EN" "reference.dtd"

PUBLIC "-//OASIS//DTD DITA Task//EN" "task.dtd"

PUBLIC "-//OASIS//DTD DITA 1.1 Task//EN" "task.dtd"

PUBLIC "-//OASIS//DTD DITA 1.0 Task//EN" "task.dtd"

PUBLIC "-//OASIS//DTD DITA Topic//EN" "topic.dtd"

PUBLIC "-//OASIS//DTD DITA 1.1 Topic//EN" "topic.dtd"

PUBLIC "-//OASIS//DTD DITA 1.0 Topic//EN" "topic.dtd"

PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd"

PUBLIC "-//OASIS//DTD DITA 1.1 Map//EN" "map.dtd"

PUBLIC "-//OASIS//DTD DITA 1.0 Map//EN" "map.dtd"

PUBLIC "-//OASIS//DTD DITA BookMap//EN" "bookmap.dtd"

PUBLIC "-//OASIS//DTD DITA 1.1 BookMap//EN" "bookmap.dtd"

XML Schema organization

The OASIS DITA document types are implemented with a set of schema modules. Some of these modules are used by every DITA schema document; others are only used by topics or by maps, and some are only used in specific specializations.

XML Schemas versus Modules

A significant feature of the DITA implementation is that it places more importance on the modules than on the actual head schema. All element and attribute type declarations are made in modules, which are then integrated into a document type using a head schema. Implementors are free to create new head schemas that reorganize the modules, introduce new modules, redefine modules, or remove modules as appropriate. For example, the standard topic XML Schema from OASIS includes all of the standard topic domains; in addition, while the default topic XML Schema allows topics to nest, it is not possible to include concepts. A new XML Schema can change one or both of these features and still be valid; the XML Schema may add or remove domains, and it may allow topic to nest concepts, or allow authoring of different types at the same level, as in the database document type.

Description of DITA Modules

Each specialization requires one or more modules to define elements for that specialization. Any XML Schema that makes use of a specialization must include the modules for that specialization.

In addition to the modules specific to a specialization, there are several files that are used by every DITA XML Schema. These are included in the base topic or map modules, so they do not need to be referenced in the XML Schema – they are already included simply by including the topic or map module.

The tables below describe each of the XML Schema modules that come with the OASIS DITA standard.

Table 5. Description of topic and its specialization module

Common module file	Purpose	Used within XML Schema
topicMod.xsd, topicGrp.xsd	Define all elements that may be used within the base topic type.	Must be included in any topic-based DITA XML Schema.
conceptMod.xsd, conceptGrp.xsd	Define elements used within the concept specialization.	concept.xsd, database.xsd
glossaryMod.xsd, glossaryGrp.xsd	Define elements used within the glossary specialization	glossary.xsd, database.xsd
referenceMod.xsd, referenceGrp.xsd	Define elements used within the reference specialization.	reference.xsd, database.xsd
taskMod.xsd, taskGrp.xsd	Define elements used within the task specialization.	task.xsd, database.xsd

Table 6. Description of map and its specialization modules

Common module file	Purpose	Used within DTD
mapMod.xsd	Define all elements that may be used within the base map type.	Must be included in any map-based DITA XML Schema.
bookmapMod.xsd	Define elements used within the bookmap specialization.	bookmap.xsd

Table 7. Description of domain modules

Common module file	Purpose	Used within DTD
indexingDomainMod.xsd	Define entities and elements used by the indexing domain.	bookmap.xsd, concept.xsd, ditabase.xsd, glossary.xsd, map.xsd, reference.xsd, task.xsd, topic.xsd
highlightDomainMod.xsd	Define entities and elements used by the highlighting domain.	concept.xsd, ditabase.xsd, glossary.xsd, reference.xsd, task.xsd, topic.xsd
programmingDomainMod.xsd	Define entities and elements used by the programming domain.	concept.xsd, ditabase.xsd, glossary.xsd, reference.xsd, task.xsd, topic.xsd
softwareDomainMod.xsd	Define entities and elements used by the software domain.	concept.xsd, ditabase.xsd, glossary.xsd, reference.xsd, task.xsd, topic.xsd
uiDomainMod.xsd	Define entities and elements used by the user interface domain.	concept.xsd, ditabase.xsd, glossary.xsd, reference.xsd, task.xsd, topic.xsd
utilitiesDomainMod.xsd	Define entities and elements used by the utilities domain.	concept.xsd, ditabase.xsd, glossary.xsd, reference.xsd, task.xsd, topic.xsd
mapGroupMod.xsd	Define entities and elements used by the map group domain.	map.xsd, bookmap.xsd
xnalDomainMod.xsd	Define entities and elements used by the XNAL domain.	bookmap.xsd

Table 8. Description of other common modules used by the files above

Common module file	Purpose	Used within module
commonElementsMod.xsd, commonElementsGrp.xsd	Define all content elements that may appear in both maps and topics.	topic.mod, map.mod (this makes it a part of any DITA XML Schema)
metaDeclMod.xsd, metaDeclGrp.xsd	Define meta elements that may appear in both maps and topics	topic.mod, map.mod (this makes it a part of any DITA XML Schema)
tblDeclMod.xsd, tblDeclGrp.xsd	Defines the complex tables used within DITA, based on the OASIS Exchange Table model.	commonElements.mod (this makes it a part of any DITA XML Schema)
ditaarch.xsd	Defines the attribute that defines DITA's architectural version	Must be imported in any topic-based DITA XML Schema.
xml.xsd	Defines the attributes with the XML namespace	Must be imported in any topic-based DITA XML Schema.

XML Schema catalog identifiers

An XML resolver needs to be able to uniquely identify an XML Schema in order to map to a local resource.

In a catalog, DITA XML Schemas must be uniquely identified using a URL location. The URLs can be used as the `xi:noNamespaceSchemaLocation` value to reference a schema.

<http://docs.oasis-open.org/dita/v1.1/schema/topic.xsd>
<http://docs.oasis-open.org/dita/v1.0/schema/topic.xsd>

<http://docs.oasis-open.org/dita/v1.1/schema/map.xsd>
<http://docs.oasis-open.org/dita/v1.0/schema/map.xsd>

<http://docs.oasis-open.org/dita/v1.1/schema/concept.xsd>
<http://docs.oasis-open.org/dita/v1.0/schema/concept.xsd>

<http://docs.oasis-open.org/dita/v1.1/schema/reference.xsd>
<http://docs.oasis-open.org/dita/v1.0/schema/reference.xsd>

<http://docs.oasis-open.org/dita/v1.1/schema/task.xsd>
<http://docs.oasis-open.org/dita/v1.0/schema/task.xsd>

<http://docs.oasis-open.org/dita/v1.1/schema/ditabase.xsd>
<http://docs.oasis-open.org/dita/v1.0/schema/ditabase.xsd>

<http://docs.oasis-open.org/dita/v1.1/schema/bookmap.xsd>

<http://docs.oasis-open.org/dita/v1.1/schema/glossary.xsd>

Chapter 3. DITA markup

The two main units of authoring in DITA are topics and maps. Each can be extended into new structural types and domains through specialization.

DITA topics

DITA topics are the basic units of DITA content. Each topic should be organized around a single subject. Topics may be of different types, such as task or reference, or may be generic, that is, without a specific type.

What are topics?

A topic is a unit of information with a title and some form of content, short enough to be specific to a single subject or answer a single question, but long enough to make sense on its own and be authored as a unit.

In DITA, a topic is the basic unit of authoring and of reuse. An XML document may contain one topic or multiple topics, and a document type may support authoring one or many kinds of topics. But regardless of where they occur, all DITA topics have the same basic structure and capabilities. Books, PDF files, Websites, and help sets, for example, can all be constructed from the same set of underlying topic content, although there may be some topics that are unique to a particular deliverable, and the organization of topics may differ to take advantage of the unique capabilities of each delivery mechanism.

DITA topics can be as small as a title that organizes other subtopics or links or as large as a few pages or screens of content. Larger units of content, such as complex reference documents or book chapters, can be created by nesting topics, either directly in a single document or indirectly through references in a DITA map. Nested DITA topics can be used to accommodate the migration of legacy non-topic-oriented content, as well as information with specific authoring requirements, such as marketing material or API reference documents.

Reference information is inherently topic-oriented, since it requires information to be modular and self-contained for the sake of retrievability.

Topic-oriented authoring for conceptual and task information has its roots in Minimalism, an instructional design technique first espoused by John Carroll. The minimalist approach to information design focusses on identifying the smallest amount of instruction that allows for the successful completion of a task, or that provides basic knowledge of a concept. Readers have goals, and they want to achieve those goals as quickly as possible. Generally, readers don't want to read information just for the pleasure of reading. They are reading to learn or to do something.

Some of the key principles of Minimalism are:

- Support actions. Let people act as they learn, and let them pursue the goals they want to accomplish.
- Document tasks, not tools or functions.
- Help readers anticipate and avoid errors.
- Let readers explore. They don't need explained what they can discover for themselves.

While DITA's topic-oriented approach has its roots in instructional design, the topic-based approach can be useful for any information that has human readers and a consistent structure.

Why topics?

Topics are the basis for high-quality information. They should be short enough to be easily readable, but long enough to make sense on their own.

By organizing content into topics, authors can achieve several goals simultaneously:

- Content is readable even when accessed from an index or search, not just when read in sequence as part of a chapter. Since most readers don't read information end-to-end, it's good information design to make sure each unit of information can be read on its own to give just-in-time help.
- Content can be organized differently for online and print purposes. Authors can create task flows and concept hierarchies for online orientation, and still have a print-friendly combined hierarchy that helps people who do want an organized reading flow.
- Content can be reused in different collections. Since the topic is written to make sense when accessed randomly (as by search), it should also make sense when included as part of different product deliverables, so authors can refactor information as needed, including just the topics that apply for each reuse scenario.

Topics are small enough to provide lots of opportunities for reuse, but large enough to be coherently authored and read. While DITA supports reuse below the topic level, this requires considerably more thought and review, since topics assembled out of smaller chunks often require editing to make them flow properly. By contrast, since topics are already organized around a single subject, authors can organize a set of topics logically and get an acceptable flow between them, since transitions from subject to subject don't need to be as seamless as the explanations within a single subject.

Information typing

Information typing is the practice of identifying types of topics that contain distinct kinds information, such as concepts, tasks, and reference information. Topics that answer different kinds of questions can be categorized as different information types. The base topic types provided by DITA provide a usable starter set that can be adopted for immediate authoring.

Classifying information by type helps authors:

- Design new information more easily and consistently.
- Ensure the right design gets used for the kind of information (retrieval-oriented structures like tables for reference information, simple sequences of steps for task information)
- Focus on tasks.
- Factor out supporting concepts and reference information into other topics, where they can be read if required and ignored if not.
- Eliminate unimportant or redundant information. Identify common or reusable subjects.

Information typing is part of the general authoring approach called structured writing, which is used across the technical authoring industry 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).

Information types in DITA are expressed as topic types. The base topic types provided by DITA can be used as a base for further specialization. New information types that require different structures and semantics are directly supported by topic type modules, each of which defines the specific markup and structural rules required to describe a particular type of topic. These modules can then be integrated into document types to support authoring information-typed topics.

Transitional text

Most writers are familiar with narrative writing, in which writers provide transitions that lead a reader from one section to the next (like "Next, we will consider..." or "Having completed the previous tasks,...").

The topic-oriented paradigm incorporated in DITA's design--and implicit in the bookmap model derived from DITA maps--favors a writing style that minimizes the use of such narrative filler between topics.

Why topic orientation in writing?

Topic-oriented writing is a disciplined approach to writing that emphasizes modularity and reuse of concise units of information--topics. A topic is enough information to fully convey an idea or task in one reading. Well-written topics can be reused in many contexts, as long as writers are careful about a few things.

Conciseness and appropriateness

Readers who are trying to quickly learn or do something will appreciate if that information is written in a clear pattern that is easy to follow and contains only the information they need to complete that task or grasp a fact. Recipes, encyclopedia entries, car repair procedures--all are informational units that serve up a uniquely focused unit of information that someone has consulted. Typically, anything before or after in the sequence of the document is not of interest--the topic contains everything pertinent to the reason the reader looked up the topic in the first place.

Locational independence

A well-written topic is reusable in other context if it is "context free," meaning that it can be inserted into a new document without revision of its content to refer to a changed context. This is where the issue of transitional text is perhaps most obvious to writers who are new to topic orientation. Phrases like "As we considered earlier..." or "Now that you have completed the initial step..." are likely to make little sense if a topic is reused in a new context in which those preliminary conditions are different or perhaps no longer existent. A well-written topic will read appropriately in any new context because the writing style within it does not lead the reader outside of the topic.

Navigational independence

Most HTML web pages are a mixture of both content and navigation, with the contained links representing the web of reading sequences and choices that a reader can make as he or she navigates through a web site. A standalone topic obviously should not have any navigation within it that would need to be changed if the topic were used in a different context. The DITA design supports this separation of navigation from topics through the use of ditamaps, which represent how topics should be organized for particular information deliverables. However, it is still tempting for writers to want to provide links within the content of a topic that allow a reader to consult external resources. DITA does not prohibit such linking, and in fact the markup allows writers to indicate that such links are external--choosing them opens a new browser window rather than taking the reader on a navigational tangent. A well-written DITA topic generally eschews internal linking to other topics within the document set--these are best supported through the map, which allows the topics to be used in any context without internal revision.

Transitional text workarounds

Topic orientation does not mean that transitions can't be authored in DITA. The key to providing both locational independence and intentional sequences is the adroit use of DITA markup features.

For DITA 1.1, the print attribute on the topicref element of a ditamap now allows a new value of "printonly," which directs processing to skip such a topic if the map is instead being output to an online format, for which transitions are not necessary. A topic designated as printonly can be written in whatever style an author feels necessary to let the topic serve as a transitional topic in the flow of printed information. A DITA topic can be as minimal as just a title, so you can craft your transitional topic to have just as much information as necessary for this output-specific nonce.

Another strategy that you can use to insert transitional sequences into a map of topics is the use of conditionality to include or exclude the content of shortdescs or of paragraphs at the end of a topic. With

rigor among your team, this can be made to work, but it is not as general as the printonly approach--if you share your conditionally marked topics with other business partners or teams, you will have to instruct them on the proper runtime settings to enable those conditions to be honored the way you intended. The printonly method works generally for everyone who inherits maps that reference transitional topics.

Generic topics

The generic or unspecialized topic type provides the base for other specialized topic types, and also provides a place to author content that does not belong in existing specialized types.

Why generic topics?

For specializers, generic topics provide an appropriate base for new specializations that do not fit the concept/task/reference mold. For authors, generic topics provide a way to author untyped content in DITA.

Although typed content is always preferable for consistency and processing concerns, the generic type can be useful when authors are not trained for information typing or when the currently available specialized types are inappropriate.

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 ID, a title, and a body. Topic structures can consist of the following parts:

Topic element

Required *id* attribute, contains all other elements

Title 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 is 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 Container for various kinds of topic metadata, such as change history, audience, product, and so on.

Body The actual topic content: paragraphs, lists, sections - whatever the information type allows.

Related links

Links 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. Nesting can result in complex documents that are less usable and less reusable, and should be used carefully. It is more often appropriate for reference information, which can support longer documents organized into multiple topics for scanning and retrieval. The rules for nesting topics can vary from document type to document type: for

example, the concept document type allows nesting of other concepts only, while the database document type allows nesting of any of the standard DITA topic type.

Topic content

All topics, regardless of topic type, build on the same common structures.

Topic bodies

While all topic types have the same elements for title, short description or abstract, and prolog, they each allow different content in their body.

Sections and examples

Sections and examples can be contained only by the body of a topic. They cannot nest. They can contain block-level elements like paragraphs, phrase-level elements like API names, or text.

Block-level elements

Paragraphs, lists, and tables are kinds 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

Authors can intermix markup with text when they need to identify part of a paragraph or even part of a sentence as having special significance. Phrases can usually contain other phrases and keywords as well as text. Keywords can only contain text.

Images

Authors can insert images using the image element. Images can be used at the block level, for example to show screen captures or diagrams, or at the phrase level, for example to show what icons or toolbar buttons look like.

Multimedia

Authors can create multimedia for online information using the object element, for example to display SVG diagrams that can be rotated and explored.

Topic modules

There are several modules that provide the basic topic elements and attributes. Some are specific to topic and its specializations, others are shared with DITA maps.

tblDecl.mod (DTD)

tblDeclMod.xsd, tblDeclGrp.xsd (Schema)

Defines the elements for authoring tables, based on the CALS table model but with some DITA-specific extensions.

metaDecl.mod (DTD)

metaDeclMod.xsd, metaDeclGrp.xsd (Schema)

Defines metadata elements, which can appear in both topics and in maps, where metadata can be defined for multiple topics at once.

commonElements.mod, commonElements.ent (DTD)

commonElementsGrp.xsd (Schema)

Defines common elements such as ph or keyword that appear in both topics and maps.

topicAttr.mod, topicDefn.ent (DTD only - folded into other schema files))

Common DITA attributes and entities.

xml.xsd, ditaarch.xsd (Schema only - folded into other DTD files)

Common XML attributes and the DITA architecture version attribute

topic.mod (DTD)

topicMod.xsd, topicGrp.xsd (Schema)

Defines the rest of the elements in a topic.

Concepts

DITA concept topics answer "What is..." questions. They include a body-level element with a basic topic structure, including sections and examples.

Why concepts?

Concepts provide background that helps readers understand essential information about a product, interface, or task. Often, a concept is an extended definition of a major abstraction such as a process or function. Conceptual information may explain a product and how it fits into its category of products. Conceptual information helps users to map their existing knowledge to tasks and other essential information about a product or system.

Concept structure

The `<concept>` element is the top-level element for a DITA concept topic. Every concept contains a `<title>` and a `<conbody>` and optional `<titlealts>`, `<shortdesc>` or `<abstract>`, `<prolog>`, `<related-links>`, and nested topics.

The `<conbody>` element is the main body-level element for a concept. Like the body element of a general topic, `<conbody>` allows paragraphs, lists, and other elements as well as sections and examples. But `<conbody>` has a constraint that a section or an example can be followed only by other sections or examples.

Here is an example of a simple concept topic.

```
<concept id="concept">
  <title>Bird Calling</title>
  <conbody>
    <p>Bird calling attracts birds.</p>
    <example>
      <p>Bird calling requires learning:</p>
      <ul>
        <li>Popular and classical bird songs</li>
        <li>How to whistle like a bird</li>
      </ul>
    </example>
  </conbody>
</concept>
```

Modules

concept.mod (DTD)

conceptMod.xsd, conceptGrp.xsd (Schema)

Tasks

Task topics answer "How do I?" questions, and have a well-defined structure that describes how to complete a procedure to accomplish a specific goal.

Why tasks?

Tasks are the essential building blocks for providing procedure information. A task topic answers the "How do I?" question by providing precise step-by-step instructions detailing what to do and the order in which to do it. The task topic includes sections for describing the context, prerequisites, expected results, and other aspects of a task.

Task structure

The `<task>` element is the top-level element for a task topic. Every task topic contains a `<title>` and a `<taskbody>` and optional `<titlealts>`, `<shortdesc>` or `<abstract>`, `<prolog>`, `<related-links>`, and nested topics.

The `<taskbody>` element is the main body-level element inside a task topic. A task body has a very specific structure, with the following elements in this order: `<prereq>`, `<context>`, `<steps>`, `<result>`, `<example>` and `<postreq>`. Each of the body sections is optional.

`<prereq>`

Describes information needed before starting the current task.

`<context>`

Provides background information for the task. This information helps the user understand what the purpose of the task is and what they will gain by completing the task. This section should be brief and does not replace or recreate a concept topic on the same subject, although the context section may include some conceptual information.

`<steps>`

Provides the main content of the task topic. A task consists of a series of steps that accomplish the task. The `<steps>` section must have one or more `<step>` elements, which provide the specifics about each step in the task.

The `<step>` element represents an action that a user must follow to accomplish a task. Each step in a task must contain a command `<cmd>` element which describes the particular action the user must do to accomplish the overall task. The step element can also contain information `<info>`, substeps `<substeps>`, tutorial information `<tutorialinfo>`, a step example `<stepxmp>`, choices `<choices>` or a stepresult `<stepresult>`, although these are optional.

`<result>`

Describes the expected outcome for the task as a whole.

`<example>`

Provides an example that illustrates or supports the task.

`<postreq>`

Describes steps or tasks that the user should do after the successful completion of the current task. It is often supported by links to the next task or tasks in the `<related-links>` section.

Here's an example of a task topic.

```
<task id="ertx">
  <title>Creating an ERTX file</title>
  <taskbody>
    <context>Each morning before breakfast you need to
create a fresh ERTX file.</context>
    <steps>
      <step><cmd>Start ERTX.</cmd></step>
      <step><cmd>Click New ERTX File.</cmd></step>
    </steps>
    <result>You now have your ERTX file for today!</result>
  </taskbody>
</task>
```

Modules

task.mod (DTD)

taskMod.xsd, taskGrp.xsd (Schema)

Reference

Reference topics describe regular features of a subject or product, such as commands in a programming language.

Why reference?

In technical information, reference topics are often used to cover subjects such as the commands in a programming language. Reference topics can hold anything that has regular content, such as ingredients for food recipes, bibliographic lists, catalogues, and the like. Reference topics provide quick access to facts. Information needed for deeper understanding of a reference topic or to perform related procedures should be provided in a concept or task topic.

Reference structure

The <reference> element is the top-level element for a reference topic. Every reference topic contains a <title> and a <refbody> and optional <titlealts>, <shortdesc> or <abstract>, <prolog>, <related-links>, and nested topics.

The <refbody> element holds the main content of the reference topic. Reference topics limit the body structure to tables (both simple and standard), property lists, syntax sections, and generic sections and examples.

All of the elements of <refbody> are optional and may appear in any sequence and number.

<section>

Represents an organizational division in a reference topic. Sections organize subsets of information within a larger topic. You can only include a simple list of peer sections in a topic; sections cannot be nested. A section may have an optional title.

<refsyn>

Contains syntax or signature content (for example, a command-line utility's calling syntax, or an API's signature). The <refsyn> contains a brief, possibly diagrammatic description of the subject's interface or high-level structure.

<example>

Provides containing examples that illustrate or support the current topic. The <example> element has the same content model as <section>.

<table>

Organizes information according into a tabular rows and columns structure. Table markup also allows for more complex structures, including spanning rows and columns, as well as table captions.

<simpletable>

Holds information in regular rows and columns and does not allow for a caption.

<properties>

Lists properties and their types, values, and descriptions.

Here's an example of a reference topic.

```
<reference id="boldproperty">
<title>Bold property</title>
<shortdesc>(Read-write) Whether to use a bold font for the specified
text string.</shortdesc>
<refbody>
  <refsyn>
    <synph>
      <var>object</var><delim>.</delim><kwd>Font</kwd><delim>.</delim>
      <kwd>Bold</kwd><delim> = </delim><var>trueorfalse</var>
    </synph>
```

```

</refsyn>
<properties>
  <property>
    <proptype>Data type</proptype>
    <propvalue>Boolean</propvalue>
  </property>
  <property>
    <proptype>Legal values</proptype>
    <propvalue>True (1) or False (0)</propvalue>
  </property>
</properties>
</refbody>
</reference>

```

Modules

reference.mod (DTD)
referenceMod.xsd, referenceGrp.xsd (Schema)

Glossary

Each DITA glossary topic defines a single sense of one term. Besides identifying the term and providing a definition, the topic can identify related terms.

Why glossary?

Defining terminology is helpful to the writer so a team of writers can use the same term for the same thing. The reader can also get an explanation of unfamiliar terminology. More generally, identification of the things described by the content encourages more precise treatment of those subjects. Glossary topics can be assembled by authors or processes to create glossaries for various purposes, including books, Web sites, or development projects.

Glossary structure

The <glossentry> element is the top-level element for a DITA glossary topic. Every glossary topic contains a <glossterm> and a <glossdef> element and optional <related-links>.

Where a term has multiple senses, the writer should create multiple topics with the same term in the <glossterm> element but different definitions in the <glossdef> element. A process can collate and group glossary entries by term when generating formatted output. Note that definitions with the same term in one language can have different terms in other languages, so translation can result in different collation and grouping of the same set of glossary entries.

Here is an example of a simple glossary entry:

```

<glossentry id="ddl">
  <glossterm>Data Definition Language</glossterm>
  <glossdef>A language used for defining database schemas.</glossdef>
</glossentry>

```

To create a glossary, authors can group multiple entries together either by authoring in a single document under a container topic using the database document type (database.dtd/database.xsd), or by referencing the glossary topics in a map, or by using an automated process, for example electing glossary topics from a repository based on the <term> markup in a particular collection of topics.

Modules

glossentry.mod (DTD)
glossentryMod.xsd, glossentryGrp.xsd (Schema)

Topic domains

A DITA domain defines a set of elements associated with a particular subject area or authoring requirement regardless of topic type.

The elements in a domain are defined in a domain module which can be integrated with a topic type to make the domain elements available within the topic type structure. Currently the following domains are provided:

Table 9. DITA topic domains

Domain	Description	Short name	Module name
Typographic	For highlighting when the appropriate semantic element doesn't exist yet	hi-d	highlightDomain.mod (DTD) highlightDomain.xsd (Schema)
Programming	For describing programming and programming languages	pr-d	programmingDomain.mod (DTD) programmingDomain.xsd (Schema)
Software	For describing software	sw-d	softwareDomain.mod (DTD) softwareDomain.xsd (Schema)
User interfaces	For describing user interfaces	ui-d	uiDomain.mod (DTD) uiDomain.xsd (Schema)
Utilities	For providing imagemaps and other useful structures	ut-d	utilitiesDomain.mod (DTD) utilitiesDomain.xsd (Schema)
Indexing	For extended indexing functions such as see and see-also	indexing-d	indexingDomain.mod (DTD) indexingDomain.xsd (Schema)

DITA maps

Maps organize topics for output to a specific deliverable, including generating navigation files and links to related topics.

What are maps?

DITA maps are documents that collect and organize references to DITA topics to indicate the relationships among the topics. They can also serve as outlines or tables of contents for DITA deliverables and as build manifests for DITA projects.

DITA maps represent the architecture of an information set – what topics are needed, in what order or relationships, to support a particular set of user goals or other requirements.

Maps describe the context in which the topics will be read – the audience, platform, relationships, requirements of the information set. In this way, the topics themselves become relatively context-free, and can be more easily used and reused in many different contexts, as defined by maps.

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 some similarities to RDF (Resource Description Framework) and ISO (International Standards Organization) topic maps. See <http://www.w3.org/RDF/> and <http://www.topicmaps.org/> for more information on those standards.

A DITA map file references one or more DITA topic files using <topicref> elements. The <topicref> elements can be nested or otherwise organized to reflect the desired relationships between the referenced topics. Map files need to have a file extension of .ditamap to be processed properly.

Why DITA maps?

Maps allow scalable reuse of content across multiple contexts. They can be used by information architects, writers, and publishers to plan, develop, and deliver content.

Among the specific uses that maps support:

Defining an information architecture

The map can be used to define what topics are required for a particular audience and user goals, even before the topics themselves exist.

Providing an authoring interface

The map can be used as a starting point for authoring new topics and integrating existing ones.

Defining what topics to build for a particular output

Maps point to topics that are included in output processing. Authors or publishers can use maps to specify a set of topics to transform at the same time, instead of transforming each topic individually.

Defining online navigation

Maps can define the online navigation or table of contents for the topics it points to.

Defining what topics to print

Maps can define a hierarchy that will determine how topics will be combined and nested for printing.

Defining related links

Maps define relationships among the topics they reference; on output, these relationships can be expressed as related links among the topics in each relationship.

Common DITA map attributes and metadata

DITA maps have many of the same common attributes as DITA content, but also have some additional ones for controlling the way relationships are interpreted for different output purposes.

Because DITA maps may encode structures that are wholly or partially specific to a particular medium or kind of output (for example, hyperlinked web pages or printed books), DITA maps contain attributes to help processors interpret the map for each kind of output. These attributes (such as print and toc) are not available in DITA content: individual topics, once separated from the high-level structures and dependencies associated with a particular kind of output, should be entirely reusable across multiple media.

Linking attributes

The collection-type and linking attributes affect how related links are generated for topics described in the map.

collection-type

The collection-type attribute indicates how a particular set of sibling topicrefs relate to each other. The collection-type attribute is set on the container element for the sibling topicrefs. The collection-type value can indicate whether to generate links among the siblings, and what kind of links to generate (for example, next and previous links for a sequence, or sibling links for a family). The collection-type attribute can also indicate how the parent topic should link to its children (for example, showing the child links as a numbered list when the collection-type is sequence).

linking

By default, relationships between topics in a map are reciprocal: children link to parents and vice versa; next and previous topics in a sequence link to each other; topics in a family link to their siblings; topics

in table cells of the same row in a relationship table link to each other. This default behavior can be modified using the linking attribute, which lets a topic modify how it participates in a relationship:

- A topic reference with `linking="none"` does not exist in the map for the purposes of calculating links
- `linking="sourceonly"` means that the topic will link to its related topics but not vice versa
- `linking="targetonly"` means that the related topics will link to it, but not vice versa
- `linking="normal"` is the default, and means that linking will be reciprocal (the topic will link to related topics, and they will link back to it)

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

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

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

Figure 1. Simple linking example

```

<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>A.dita</relcell>
    <relcell linking="sourceonly">B.dita</relcell>
  </relrow>
</reltable>

```

- A** links to A1, A2 as children
(no links to B as a child, no links to B as related)
- A1** links to A as a parent
 links to A2 as next in the sequence
(no links to B as previous)
- A2** links to A as a parent
 links to A1 as previous in the sequence
- B** links to A as related

Figure 2. Linking example with the linking attribute

Navigation, media, and chunking attributes

There are standard attributes available for identifying output-media-specific content and for rechunking topics.

toc, navtitle, locktitle

Authors can exclude entries from navigation output (such as an online table of contents, or a Web site map) using the toc attribute. By default, hierarchies are included in navigation output, and tables are excluded.

Authors can provide a shorter version of the title for use in the navigation using the navtitle attribute. By default the navtitle attribute is ignored, and used only to help the author keep track of the target topic's title. The locktitle attribute can be set to ensure that the navtitle takes effect and overrides any title values in the target topic, or defined elsewhere in the topic reference metadata.

print, search

You can set attributes on a topic to indicate whether it should be included in printed output and search indexes.

chunk

When a set of topics is transformed using a map, multi-topic files can be broken into smaller files, and multiple individual topics can be combined into a single larger file, using the chunk attribute. There is no default value for the chunk attribute, but a default for an entire map may be established by setting the chunk attribute on the map element or a specialization. For a detailed description of the chunk attribute and its usage see "Chunking" on page 46.

copy-to

When a set of topics is transformed using a map, duplicate topic versions can be created using the copy-to attribute. The copied topic will have a new file name or location as provided in the copy-to attribute, and the map can override the default title and shortdesc for this particular copy by providing values for them in the map using the topicref's navtitle and shortdesc. For information on how the copyto attribute can be used with the chunk attribute see "Chunking" on page 46.

Shared attributes

DITA maps use the same metadata and reuse attributes that DITA topics use.

- product, platform, audience, otherprops, rev, status, importance, xml:lang, translate
- id, conref
- props, base

DITA maps also use many of the same attributes that are used with link or xref elements in DITA content:

- format, scope, href, keyref, type, query

When new attributes are specialized off of props or base as a domain, they may be incorporated into both map and topic structural types.

DITA map structure

Maps organize topics into hierarchies, tables, and groups, and have special elements for referencing other maps.

topicref elements are the basic elements of a map. A topicref can point to a DITA topic, map, or to any other resource that can be processed or linked to. They can also have a title, short description, and the same kinds of prolog-level metadata available in topics.

topicref elements can be nested to create a hierarchy, which can be used to define print output, online navigation, and parent/child links. The topichead element can be used for nodes in the hierarchy that provide containers without equivalent topics: they are equivalent to topicref elements with a navtitle but no href or equivalent referencing attribute.

Relationship tables are defined with the reltable element. Relationship tables can be used to define relationships among the topics in different cells of the same row. In a relationship table, the columns define common attributes or metadata for the topics in that column. The rows define relationships, with each cell representing a different role in the relationship. For example, a table with different columns for concepts, tasks, and reference topics could be used to define the relationship between a task and the topics that support it.

Both hierarchies and tables can be annotated using the collection-type attribute to define sets of siblings that are part of a particular kind of collection, for example a set of choices, a sequence, or a family. These collection-types can affect link generation, and may be interpreted differently for different outputs.

Groups or collections outside of a hierarchy or table can be defined with the topicgroup element, which is equivalent to a topicref with no referencing attributes or titles. Groups can be combined with hierarchies and tables, for example by including a group within a table cell or within a set of siblings in a hierarchy.

Most elements in the map, including the map itself, can contain metadata, which typically applies to the element and its descendants, as described in "Inheritance of attributes and metadata in maps" on page 29.

Example of a simple map with a 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>
```

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

A links to B, C1, C2

B links to A, C1, C2

C1, C2 link to A, B

DITA map modules

Maps have the same module structure as topics, and share some of the same modules for defining metadata.

map.mod (DTD)

mapMod.xsd, mapGrp.xsd (Schema)

Defines the base map structures.

mapGroup.mod (DTD)

mapGroup.xsd (Schema)

Adds topicgroup and topichead as specialized variants of topicref.

Inheritance of attributes and metadata in maps

Some of the attributes and metadata in a map can be inherited based on the structures in the map.

Inheritance is additive except where this would cause a conflict. When there is a conflict, the value defined closest (most specifically) to the topicref 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)
 - topichead/topicgroup/topicref container (more specific)
 - topicref (most specific)
 - reltable (more specific)
 - relcolspec (more specific)
 - relrow (more specific)
 - topichead/topicgroup/topicref container (more specific)

- topicref (most specific)

The following attributes and metadata elements are inheritable:

Attributes

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

Elements

author, source, publisher, copyright, critdates, permissions
audience, category, prodinfo, othermeta

Attributes and metadata can be defined at the root level (attributes on the map element itself, topicmeta as a direct child of the map element) to apply them to the entire map. They can also be applied at any point in a hierarchy, group, or table. Tables can be particularly useful for attribute and metadata management, since they can be applied to entire columns or rows as well as individual cells.

While the chunk attribute no longer inherits a value from containers (new with DITA 1.1), specifying a value for the chunk attribute on the map or map specialization element establishes a default value for chunking that applies to the entire map unless overridden by more specific chunk attribute settings elsewhere within the document.

Related reference

“Metadata inheritance between maps and topics” on page 38

The topicmeta element in maps contains numerous elements for the declaration of metadata. In general, specifying metadata in a topicmeta element is equivalent to specifying it in the topic, while allowing that topic to be reused in other maps where the metadata does not apply. Many items in the metadata also cascade to nested topic references within the map.

Bookmark

The bookmark specialization of DITA’s standard DITA map allows you to organize your DITA topics into a collection that can be printed as a book or other pagged layout.

Why use a bookmark?

The OASIS bookmark application of DITA allows you to produce your DITA topics and even whole DITA maps as the content of a formally defined book. This allows you to produce not only maps for online deliverables, but also PDFs with the same content, replete with covers, formal notices and frontmatter, and so forth.

What is a bookmark?

A bookmark is a special kind of DITA map that defines the major structures and setup information for producing maps of information as a book.

A typical DITA map might have a title and then a set of topicrefs, in a sequence, in a hierarchy, or both, that define the structure by which topics are to be viewed as a complete information deliverable. A DITA map does not have structures to specifically designate how topics are to be treated as chapters, preface content, or even for constructing a cover or special content (such as edition notice boilerplate). To enable DITA content to be viewed in a more booklike manner, you need a context that represents all the special processing that might be invested on producing a map or set of maps as a book. That is the role of DITA’s bookmark specialization.

A bookmark has the following specialized major structures:

- An optional title or booktitle (which itself can have substructure for complex titles)
- Optional bookmeta (all the information about the book--its owners, authors, publishing data, and so forth)
- Optional frontmatter (the usual preface and introductory topics in a book prior to the gist of the book, and also including booklists--containers for special collections of information in a book)
- Any number of chapters or parts (which can contain chapters)
- Any number of appendix topics
- Optional backmatter (often for special notices or supplementary information, and also for booklists)
- Any number of reltables, as in a typical map.

What is bookmeta?:

Bookmeta is a specialization of the topicmeta element in a DITA map. It has specialized content for holding information about a particular book represented by the bookmap.

What are booklists?:

A booklist is a specialized topicref that indicates a collection of information from the content of a book. Sets of <booklist> elements (or elements derived from booklist) are contained in the <booklists> element.

A common kind of booklist is the Table of Contents. If you wish, you can define a completely new collection such as a Table of Footnotes and provide either a pre-populated topic with that content, or provide processing that will collect and insert that topical content during the book processing.

Specific booklist elements are provided in the OASIS bookmap for:

- toc--the usual Table of Contents (commonly located in the frontmatter booklists)
- figurelist--a list of figures
- tablelist--a list of tables
- abbrevlist--a list of abbreviations
- trademarklist--a list of trademarks
- bibliolist--a list of bibliographic references
- glossarylist--a list of glossary terms and definitions
- indexlist--an index (commonly located in the backmatter booklists)
- booklist--reference any other topic that contains booklist-like material, or specialize the element to represent a collection that will be gathered by new override processing.

What is in frontmatter?:

Front matter of a book typically contains prefaces, instructions, or other introductory material prior to the actual content of a book.

Specific frontmatter elements include any number of:

- booklists--collections of book parts, like ToC
- notices--edition notices, safety notices, terms and conditions, and so forth
- dedication--to my forebearing significant other...
- colophon--how the book was produced
- bookabstract--handy for search tools and registry into tracking/workflow systems
- draftintro--special content for reviewers
- preface--an introduction or introductory statement
- topicref--extensible placeholder for any topic

- `topichead`--as in a DITA map, this element allows grouping with a navigation title
- `topicgroup`--as in DITA map, this element allows grouping without a navigation title.

What is in backmatter?:

Back matter of a book typically contains closing information that follows the main content of a book.

Specific backmatter elements include any number of:

- `booklists`--collections of book parts, like the index
- `notices`--edition notices, safety notices, terms and conditions, and so forth
- `dedication`--to my first grade teacher...
- `colophon`--how the book was produced
- `amendments`--points to a list of amendments or updates to the book
- `topicref`--extensible placeholder for any topic
- `topichead`--as in a DITA map, this element allows grouping with a title
- `topicgroup`--as in DITA map, this element allows grouping without a title.

How is a bookmap authored and produced?

As a specialization of DITA map, the DITA bookmap will be supported by default like a typical DITA map in DITA-aware editors and processing tools.

With appropriate style and functional overrides, XML editors can display a DITA bookmap just as they would the XML structure for other book-supporting DTDs, and specialized DITA processing or DITA bookmaps will exploit the book metadata and book features in high quality paged output.

For example, you might have a DITA map that represents the hierarchy you intend for a particular Web deliverable of your DITA topics. Now you wish to also produce that map as a formal book to be packaged along with your boxed product. You can open a new bookmap instance and define a title for the book, indicate any required publisher information in the bookmeta area, link in a preface, indicate whether you want a table of figures along with the table of contents, and finally create a chapter element that references your existing DITA map. You have created a complete book deliverable, at least in a simplistic manner. The design of bookmap allows you to refine your deliverable and even make the book version of the map have content that is specific to the book version, versus the Web version.

Example

```
<bookmap id="taskbook">
  <booktitle>
    <mainbooktitle>Product tasks</mainbooktitle>
    <booktitlealt>Tasks and what they do</booktitlealt>
  </booktitle>
  <bookmeta>
    <author>John Doe</author>
    <bookrights>
      <copyrfirst>
        <year>2006</year>
      </copyrfirst>
    <bookowner>
      <person href="janedoe.xml">Jane Doe</person>
    </bookowner>
    </bookrights>
  </bookmeta>
  <frontmatter>
    <preface/>
  </frontmatter>
  <chapter format="ditamap" href="installing.ditamap"/>
  <chapter href="configuring.xml"/>
  <chapter href="maintaining.xml">
```

```

    <topicref href="maintainstorage.xml"/>
    <topicref href="maintainserver.xml"/>
    <topicref href="maintaindatabase.xml"/>
  </chapter>
  <appendix href="task_appendix.xml"/>
</bookmap>

```

Modules

DTD:

```

bookmap.dtd
bookmap.mod

```

Schema:

```

bookmap.xsd
bookmapGrp.xsd
bookmapMod.xsd

```

Map domains

The following domains provide expanded functionality for map document types that incorporate them.

The xNAL domain

Book metadata has many commonalities with other metadata standards, in particular those for addressing persons and places. The OASIS xNAL Standard (extensible Name and Address Language) was selected to represent close mappings from the DITA bookmap metadata content model to an existing standard.

The OASIS CIQ standard for global customer information management contains the definition of the OASIS extensible Name and Address Language (xNAL) metadata elements. Version 2 of the standard states:

The objective of xNAL is to describe a common structure for Personal/Organization Names and Addresses that would enable any applications that want to represent customer names and addresses in a common standard format. The applications could be CRM/e-CRM, Customer Information Systems, Data Quality (Parsing, Matching, Validation, Verification, etc), Customer Data Warehouses, Postal services, etc.

However, any party for its own purposes and applications may use xNAL grammar or parts of it.

The metadata elements defined for bookmap naturally include structures that identify the authors and content owners. Although it was not possible to subset the xNAL standard definitions directly into DITA due to differences between the two processing architectures, OASIS DITA Technical Committee members determined that there was value in having at least a transformational equivalence between bookmap and xNAL definitions for names and addresses. This equivalence enables XML-aware tools in workflow systems to capture and manipulate DITA bookmap names and addresses in a standard way.

For DITA 1.1, the xNAL domain is part of the bookmap specialization. This means that the xNAL elements appear any place that bookmap allows the <author> element - within bookmeta and topicmeta.

Metadata elements and common attributes

The same metadata elements and many of the same attributes are available in both DITA topic types and DITA map types. The sharing of metadata elements allows the metadata assigned to a topic when it is created to be supplemented or overridden when the topic is included in a collection. The sharing of attributes allows processing behaviors, such as content reuse and conditional processing, to be implemented consistently across both maps and topics.

Common metadata elements

The following metadata elements occur in both maps and topic prologs.

Many of the DITA metadata elements map to Dublin Core metadata (<http://dublincore.org/>), with the exception of Dublin Core elements such as title, which maps to the DITA <title> element.

Publication metadata elements

These elements provide standard information about the topic as a publication.

Some content providers might choose to provide such information only in the map or the initial topic for a deliverable.

author The person or organization who created the content. This element is equivalent to the Dublin Core Creator.

publisher

The organization who provides and distributes the content. This element is equivalent to the Dublin Core Publisher.

copyright

The legal ownership for the content. This element is equivalent to the Dublin Core Rights.

Management metadata elements

These elements provide a basis for managing the publication process for topics.

The management elements might get updated by workflow processes or provide input for such processes:

source An identifier or name for the original form of the content. This element is equivalent to Dublin Core Source.

critdates

Milestones in the publishing cycle. This element is equivalent to Dublin Core Date.

permissions

Specification of the level of entitlement needed to access for content.

resourceid

The identifier associated with the topic when provided to the specified application.

Metadata qualification elements

These elements can be used to provide additional information about a topic, and may be used in conjunction with conditional processing attributes to provide more information about the attribute values.

The metadata elements apply to an entire topic, and can also be used in a map to apply metadata to multiple topics at a time. Metadata elements can expand on the values used in metadata attributes. (See metadata attributes.) For example, the audience element in a topic's prolog can define an audience in terms of type, job, and experience level, and give it a name; when there is content within the topic's body that applies only to that audience, that content can identify its audience by the same name used in the prolog.

When metadata is expressed in a map, it supplements any metadata expressed in the topics it references. When metadata in a map and a topic conflict (for example, both define a publisher), by default the value in the map takes precedence, on the assumption that the author of the map has more knowledge of the reusing context than the author of the topic.

audience

The type, job, experience level, and other characteristics of the reader for the topic. Many of these characteristics have enumerated values, but the enumeration can be extended through associated

attributes. For instance, the audience type enumeration can be extended through an `othertype` attribute. The audience element can provide more information about a value used by an audience attribute.

category

A classification of the topic content. Such classifications are likely to come from an enumerated or hierarchical set. This element is equivalent to both Dublin Core Coverage and Dublin Core Subject.

keywords

Terms from a controlled or uncontrolled subject vocabulary that apply to the topic.

prodinfo

The definition of the product or platform for the topic. The `prodinfo` element can provide more information about values used by the product and platform attributes.

othermeta

A name-value pair specifying other metadata about the topic.

data

Additional data or metadata about the topic or map, or elements in the topic or map; allows for more complex metadata using nested values. Primarily used as the base for specialized metadata structures.

Common attributes

The following attributes are common across most DITA elements.

Identifier and content referencing attributes

The `id` and `conref` attributes are available on almost all DITA elements, and allow sharing of content between DITA topics or maps.

Related concepts

“IDs and references” on page 41

The DITA identity attribute provides mechanisms for identifying content for retrieval or linking. The syntax for referencing IDs is consistent regardless of the referencing mechanism.

“Content inclusion (`conref`)” on page 42

The DITA `conref` attribute provides a mechanism for reuse of content fragments. The `conref` attribute stores a reference to another element and is processed to replace the referencing element with the referenced element.

Metadata attributes

The metadata attributes express qualifications on the content. These qualifications can be used to modify the processing of the content.

One typical use of the metadata attributes is to filter content based on their values. Another typical use is to flag content based on their values, for example by highlighting the affected text on output. Typically audience, platform, product, and otherprops are used for filtering, and the same attributes plus `rev` are used for flagging. Status and importance are used for tool-specific or transform-specific behavior, for example marking steps in a task as optional or required.

Conditional processing attributes:

In general, a conditional processing attribute provides a list of one or more values separated with whitespace. For instance, `audience="administrator programmer"` qualifies the content as applying to administrators and programmers.

There are several attributes intended for conditional processing, 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 text
- rev: the revision or draft number of the current document (typically used for flagging only, not for filtering)
- otherprops: other properties that do not require semantic identification
- props: a generic conditional processing attribute that can be specialized to create new semantic conditional processing attributes

Related concepts

“Conditional processing attributes” on page 44

Other metadata attributes:

Other attributes are still considered part of an element’s metadata but are not designed for conditional processing, such as 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.

Miscellaneous Attributes

DITA has several elements for supporting translation and for providing additional classification or typing of a specific element.

Miscellaneous attributes of DITA elements include the following:

xml:lang

The xml:lang attribute’s behavior is described in detail in the XML specification:

<http://www.w3.org/TR/REC-xml/#sec-lang-tag> The attribute identifies a language by means of the standard language and country codes (as described in RFC 4646). For instance, French Canadian would be identified by the value fr-ca. As is usual, the language applies to the contained content and attributes of the current element and contained elements, other than fragments that declare a different language.

translate

Determines whether the element requires translation. A default value may be inferred from the element type: for example, <apiname> may be untranslated by default, whereas <p> may be translated by default. A list of suggested defaults is provided in “All elements with translation properties” on page 53.

dir Determines the direction in which the content should be read.

outputclass

The outputclass attribute provides a label on one or more element instances, typically to specify a role or other semantic distinction. As the outputclass attribute doesn’t provide a formal type declaration or the structural consistency of specialization, it should be used sparingly, often 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: <uicontrol outputclass="button">Cancel</uicontrol>. The outputclass value could be used to trigger XSLT or CSS rules, as well as providing a mapping to be used for future migration to a more specialized set of UI elements.

base A generic attribute that has no specific purpose, but 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).

The attributes @xml:lang, @translate, and @dir are described in more detail in “Translation” on page 49.

Architectural attributes

DITA provides some attributes to provide type information to processors instead of qualifications or properties of content.

Ordinarily, architectural attributes don't appear in the source files for document instances. Instead, architectural attributes appear in document instances through defaults set in the DTD or Schema declaration. This practice ensures that the creation of document instances cannot produce invalid values for the architectural attributes. These attributes are as follows:

class This attribute identifies the specialization module for the element type as well as the ancestor element types and the specialization modules to which they belong. Every DITA element has a class attribute.

domains

This attribute identifies the domain specialization modules used in a topic and, for each domains module, its module dependencies. Every topic and map element has a domains attribute.

DITAArchVersion

This attribute identifies the version of the DITA architecture used by the DTD or schema. Every topic and map element has 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 without the DTD or Schema declaration, a normalization process can instill the architectural attributes in the document instance.

Topic properties in topics and maps

The properties of a topic (including metadata attributes and metadata elements) can be specified in the topic itself or in references to the topic within maps.

Within a topic, properties can be expressed using metadata attributes on the topic element or using publication, management, or metadata elements in the topic prolog.

DITA offers users three levels at which to apply topic properties. Topic attributes and metadata elements can be defined within the topic itself. Properties can also be set as attributes or subelements of a topicref that points to the topic. In addition, properties can be set on container or ancestor elements of the topicref within the map. Because the topics in a branch of the navigation hierarchy typically have some common subjects or properties, this is a convenient mechanism to set properties for a set of topics.

If a property is set in both the map and topic, the map properties are additive if the property (such as the audience type) takes a list of values. If, instead, the property (such as the importance) takes a single value, the map property overrides the topic property.

Example of audience metadata in prolog and body

Prolog metadata elements can provide more information about values used in metadata attributes on content. However, prolog metadata and attribute metadata can also be used and expressed independently. The coordination shown here is possible but is not required.

```
<prolog>
  <metadata>
    <audience name="AdminNovice"
               type="administrator"
               job="customizing"
               experiencelevel="novice">

  </metadata>
```

```

</prolog>
....
<p audience="AdminNovice ProgrammerExp">This paragraph applies to both
novice administrators and expert programmers</p>

```

In the preceding example, the attribute value "AdminNovice" is associated with the audience element with the same name, which gives authors and processes more information about the audience in question: in this case, that the "AdminNovice" audience is administrators who are customizing and are new at it.

Shared metadata elements, and the lockmeta attribute

You can associate topic metadata with a topic or branch of topics in a map. By default metadata in the map supplements or overrides metadata in the topic. If the lockmeta attribute is set to "no", then the metadata in the map will not take precedence over the metadata in the topic, and conflicts will be resolved in favor of the topic.

In a map, metadata elements are authored within a <topicmeta> element, which associates metadata with the <topicref> or other element that contains it, plus the other children of that element. For example, <topicmeta> in a <relcolspec> can associate metadata with all the topics referenced in a <reltable> column.

The metadata elements in a map are the same as those in a topic, although they may be in a different order. The map metadata also includes a short description and alternate titles, which can override their equivalents in the content. In sum, the map can override or supplement everything about a topic except its content (in the topic's body element) and primary title.

Metadata inheritance between maps and topics

The topicmeta element in maps contains numerous elements for the declaration of metadata. In general, specifying metadata in a topicmeta element is equivalent to specifying it in the topic, while allowing that topic to be reused in other maps where the metadata does not apply. Many items in the metadata also cascade to nested topic references within the map.

For each element in the topicmeta, the following table provides three different behaviors.

- **How does it apply to the topic?** This describes how the metadata in a topicref interacts with the specified topic. In most cases, the properties are additive, *within the current context*. For example, when an audience is set to "user" within a topicref, this means that the audience for the specified topic is "user" when viewed within this context. This is in addition to any audience values specified within the topic, and does not apply to the topic when it is viewed in other contexts.
- **Does it cascade to other topics in the map?** This indicates whether the specified meta value cascades to nested topicrefs. For example, setting an audience to "user" implicitly means that other topicrefs within that branch of the map also have an audience of "user". Elements like linktext, which can apply only to the specified target, do not cascade.
- **What is the purpose when specified at the map level?** The map element allows for metadata to be specified for the entire map. This column describes what effect, if any, each element has when specified at this level.

Table 10. Topicmeta elements and their properties

Element	How does it apply to the topic (in this context)?	Does it cascade to other topics in the map?	What is the purpose when specified at the map level?
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

Table 10. Topicmeta elements and their properties (continued)

Element	How does it apply to the topic (in this context)?	Does it cascade to other topics in the map?	What is the purpose when specified at the map level?
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 inherits	No stated purpose, until the element is specified
data-about	Add the property to the specified target	No, unless specialized for a purpose that inherits	No stated purpose, until the element is specified
foreign	Add to the topic	No, unless specialized for a purpose that inherits	No stated purpose, until the element is specified
keywords	Add to the topic	No	No stated purpose
linktext	Not added to the topic; applies to links created based on this occurrence in the map	No	No stated purpose
othermeta	Add to the topic	No	Define metadata for the entire map
permissions	Add to the topic	Yes	Specify permissions for the entire map
prodinfo	Add to the topic	Yes	Specify product info for the entire map
publisher	Add to the topic	Yes	Specify a publisher for the map
resourceid	Add to the topic	No	Specify a resource ID for the map
searchtitle	Replace the one in the topic. If multiple searchtitles are given for one target, processors may choose to issue a warning.	No	No stated purpose
shortdesc	Not added to the topic; applies to links created based on this occurrence in the map	No	Provide a description of the map
source	Add to the topic	No	Specify a source for the map
unknown	Add to the topic	No, unless specialized for a purpose that cascades	No stated purpose, until the element is specified

Related concepts

“Inheritance of attributes and metadata in maps” on page 29

Some of the attributes and metadata in a map can be inherited based on the structures in the map.

Chapter 4. DITA processing

Several common DITA processing behaviors are driven by attributes, including navigation, content reuse, and conditional processing.

IDs and references

The DITA identity attribute provides mechanisms for identifying content for retrieval or linking. The syntax for referencing IDs is consistent regardless of the referencing mechanism.

The id attribute assigns a unique identifier to an element so the element can be referenced. The scope of uniqueness for the id attribute depends on the role of the element within the DITA architecture:

Topic IDs

Because topics are the basic units of information within DITA, the id attribute for the topic must be unique within the document instance.

A topic architecture assembles topics into a deliverable by reference. To ensure that topics can be referenced, the id attribute is required on the topic element.

The complete identifier for a topic consists of the combination of the URI for the document instance, a separating hash character, and the topic id (as in `http://some.org/some/directory/topicfile.xml#topicid`). URIs are described in RFC 2396. As is typical with URIs, a relative URI can be used as the identifier for the document instance so long as it is resolvable in the referencing context. For instance, within a file system directory, the filename of the document instance suffices (as in `some/directory/topicfile.xml#topicid`). Within the same document, the topic id alone suffices (as in `#topicid`). Where the topic element is the root element of the document instance, contexts outside the document instance may omit the topic id when referring to the topic element (as in `topicfile.xml`). If the target contains multiple peer topics organized by a `<dita>` element (as in the database document type), then the reference is resolved to the first topic in the target, except in the case of `topicrefs` being resolved as inclusion directives (for example during PDF output), in which case the entire target file is included, regardless of whether a specific topic is targeted.

The topic id can be referenced by other elements, including `topicrefs`, `links`, `xrefs`, or `conrefs` to the topic, as well as being used as part of references to the topic content.

The id attribute for DITA topics is of type ID in XML, and so must be unique within the document instance.

Element IDs within a topic

Because topic content is always contained within a topic, the id attribute for a topic content element must be unique only within the one topic that immediately contains it. This approach ensures maintainable references to content because the identifier remains valid so long as the document instance, topic, and content exist. The position of the content within the topic and the position of the topic within the document instance can change without invalidating the content identifier. In addition, this approach avoids the need to rewrite topic content ids to avoid naming collisions when aggregating topics.

The id is optional and need be added only to make the content referenceable.

The complete identifier for topic content consists of the combination of the complete identifier for the topic, a separating solidus (/), and the topic content id (as in `http://some.org/some/directory/topicfile.xml#topicid/contentid`). As noted before, the topic identifier portion can use a relative URI for the document instance in contexts where the relative URI can be resolved (as in `some/directory/topicfile.xml#topicid/contentid`).

The containing topic id must always be included when referencing an element id. Otherwise, a reference to another topic couldn't be distinguished from a reference to an element within the same topic. For references within the same document instance, the identifier for the document instance can be omitted altogether (as in #topicid/contentid).

The id attribute for elements within DITA topics is not of type ID and is not required to be unique within a document, although it should be unique within the topic.

Map IDs and element IDs within a map

For a map, the id of an element should be unique within the document instance. This approach ensures that these elements can be referenced outside the map without qualification by the map id.

For the anchor element, which exists only to identify a position within a map as a target for references, the id attribute is required. For the other elements, the id attribute is optional.

The complete identifier for a map element consists of the combination of the absolute URI for the map document instance and the element id (as in `http://some.org/some/directory/mapfile.xml#topicrefid`).

The id attributes for maps and anchors are of type ID and are required to be unique within the document instance. The id attributes for other elements in map are not of type ID and are not required to be unique.

Navigation behaviors

The following behaviors support the creation of reader navigation to or across DITA topics.

Tables of contents (TOCs)

Generated from DITA maps; can be compiled from multiple DITA maps. Parts of maps can be omitted from the generated TOC using the toc attribute.

Related links

Authored in individual topics, or generated from maps. Map-generated linking can be controlled using the linking attribute.

Indexing

Generated from index entries occurring in topic bodies, topic prologs, or DITA maps.

Content inclusion (conref)

The DITA conref attribute provides a mechanism for reuse of content fragments. The conref attribute stores a reference to another element and is processed to replace the referencing element with the referenced element.

The element containing the content reference acts as a placeholder for the referenced element. The identifier for the referenced element must be either absolute or resolvable in the context of the referencing element. (See "IDs and references" on page 41 for the details on identifiers.)

More formally, the DITA conref attribute can be considered a transclusion mechanism. In that respect, conref is similar to XInclude as well as HyTime value references. DITA differs from these mechanisms, however, by comparing the constraints of each context to ensure the ongoing validity of the replacement content in its new context. In other words, conref validity does not apply simply to the current content at the time of replacement, but to the ranges of possible content given the constraints of the two document types. A valid conref processor does not allow the resolution of a reuse relationship that could be rendered invalid under the rules of either the reused or reusing content.

If the referenced element is the same type as the referencing element and the list of domains in the referenced topic instance (declared on the domains attribute) is the same as or a subset of the list of domains 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 placeholder element. In the preferred approach, a processor resolving a conref should tolerate specializations of valid elements and generalize elements in the content fragment as needed for the referencing context.

Replacement of the placeholder occurs after parsing of the document but prior to any styling or other transformational or presentational operations on the full topic.

The target of the conref may be substituted based on build-time or runtime conditions. For example, content such as product names or install paths can be separated out from topic content since they change when the topic is reused by other products; the reusing product can substitute their own targets for the conref to allow resolution to their own product name and install paths, and so on.

The target of a conref must be within a DITA topic or DITA map (or point to the entire topic or map). Fragments of DITA content (such as a document containing only a single paragraph) do not contain enough information on their own to allow the conref processor to determine the validity of a reference to them.

The resolved element's attribute specifications can be drawn from both source and target elements in the following priority:

1. All attributes as specified on the source element except for attributes which specify the value `"-dita-use-conref-target"`
2. All attributes as specified on the target element except:
 - a. The id attribute
 - b. Any attribute also specified on the source element such that its specified value (on the source element) is not `"-dita-use-conref-target"`
3. The `xml:lang` attribute has special treatment as described in "The `xml:lang` attribute" on page 49.

The only time the resolved element would include an attribute whose specified value is `"-dita-use-conref-target"` is when the target element had that attribute specified with the `"-dita-use-conref-target"` value and the source element either had no specification for that attribute or had it specified with the `"-dita-use-conref-target"` value. If the final resolved element (after the complete resolution of any conref chain) has an attribute with the `"-dita-use-conref-target"` value, that should be treated as equivalent to having that attribute unspecified.

A given attribute value on the resolved element comes in its entirety from either the source or target: the attribute values of the target and source for a given attribute are never additive, even if the property (such as the audience type) takes a list of values.

If the target element has a conref attribute specified, the above rules should be applied recursively with the resolved element from one source/target combination becoming one of the two elements participating in the next source/target 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 topicA and topicC allow highlighting, and topicB does not, then a content reference chain of topicA->topicB->topicC should preserve any highlighting elements in the referenced content. The result is the same as if the conref pairs are resolved recursively starting from the source element.

Conditional processing (profiling)

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

DITA tries to implement conditional processing in a semantically meaningful way: rather than allowing arbitrary values to accumulate in a document over time in a general-purpose processing attribute, with meaning only to the original author, we encourage the authoring of metadata using specific metadata

attributes on content. These metadata values can then be leveraged by any number of processes, including filtering, flagging, search, and indexing, rather than being suitable for filtering only.

Conditional processing attributes

For a topic or topicref, the audience, platform, and product metadata can be expressed with attributes on the topic or topicref element or with elements within the topic prolog or topicmeta element. While the metadata elements are more expressive, the meaning of the values is the same, and can be used in coordination: for example, the prolog elements can fully define the audiences for a topic, and then metadata attributes can be used within the content to identify parts that apply to only some of those audiences.

audience

The values from the enumerated attributes of the audience metadata element have the same meaning when used in the audience attribute of a content element. For instance, the "user" value has the same meaning whether appearing in the type attribute of the audience element for a topic or in the audience attribute of a content element. The principle applies to the type, job, and experience level attributes of the audience element.

The values in the audience attribute may also be used to reference a more complete description of an audience in an audience element. Use the name of the audience in the audience element when referring to the same audience in an audience attribute.

The audience attribute takes a blank-delimited list of values, which may or may not match the name value of any audience elements.

platform

The platform might be the operating system, hardware, or other environment. This attribute is equivalent to the platform element for the topic metadata.

The platform attribute takes a blank-delimited list of values, which may or may not match the content of a platform element in the prolog.

product

The product or component name, version, brand, or internal code or number. This attribute is equivalent to the prodinfo element for the topic metadata.

The product attribute takes a blank-delimited list of values, which may or may not match the value of the prodname element in the prolog.

rev The identifier for the revision level. For example, if a paragraph was changed or added during revision 1.1, the rev attribute might contain the value "1.1".

otherprops

A catchall for metadata qualification values about the content. This attribute is equivalent to the othermeta element for the topic metadata.

The attribute takes a blank-delimited list of values, which may or may not match the values of othermeta elements in the prolog.

For example, a simple otherprops value list: `<codeblock otherprops="java cpp">`

The attribute can also take labelled groups of values, but this syntax is deprecated in DITA 1.1 in favor of attribute specialization. The labelled group syntax is similar to the generalized attribute syntax and may cause confusion for processors. A labelled group consists of a string value followed by an open parenthesis followed by one or more blank-delimited values followed by a close parenthesis. The simple format is sufficient when an information set requires only one additional metadata axis, in addition to the base metadata attributes of product, platform, and audience. The full format is similar to attribute specialization in that it allows two or more additional metadata axes. For example, a complex otherprops value list: `<codeblock otherprops="proglang(java cpp) commentformat(javadoc html)">`

props A generic attribute for conditional processing values. In DITA 1.1, the props attribute can be specialized to create new conditional processing attributes.

Using metadata attributes

Each attribute takes zero or more space-delimited string values. For example, you can use the product attribute to identify that an element applies to two particular products.

```
<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 platform="Linux">Do a special thing for Linux</li>
  </ul>
</p>
```

Figure 3. Example source

Processing metadata attributes

At processing time, you specify the values you want to exclude and the values you want to flag using a conditional processing profile (described in the DITA Language Specification). For example, a publisher producing information for a mixed audience using the basic product could choose to flag information that applies to administrators, and exclude information that applies to the extended product, and express those choices in a conditional processing profile like this:

```
<prop att="audience" val="administrator" action="flag" >
  <startflag>ADMIN</startflag>
</prop>
<prop att="product" val="extendedprod" action="exclude"/>
```

At output time, the paragraph is flagged, and the first list item is excluded (since it applies to extendedprod), but 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:

ADMIN Set the configuration options:

- Set your blink rate
- Do some other stuff
- Do a special thing for Linux

Filtering logic

When deciding whether to exclude a particular element, a process should evaluate each attribute, and then evaluate the set of attributes:

- If all the values in an attribute have been set to "exclude", the attribute evaluates to "exclude"
- If any of the attributes evaluate 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 process should exclude the paragraph; even if the paragraph applies to an audience or platform that you aren't excluding. 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.

Flagging logic

When deciding whether to flag a particular element, a process should evaluate each value. Wherever a value that has been set as flagged appears in its attribute (for example, audience="ADMIN") the process should add the flag. When multiple flags apply to a single element, multiple flags should be output, typically in the order they are encountered.

Flagging could be done using text (for example, bold text against a colored background) or using images. When the same element evaluates as both flagged and filtered (for example, flagged because of an audience attribute value and filtered because of its product attribute values), the element should be filtered.

Chunking

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

Examples of use

Here are some examples of potential uses of the chunk attribute:

Reuse of a nested topic

A content provider creates a set of topics as a single document. A reuser wants to incorporate only one of the nested topics from the document. The reuse 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.

Usage of the chunk attribute

When a set of topics is transformed for output using a map, the map author may use the chunk attribute to override whatever default chunking behavior applies. The chunk attribute allows the map author to request that multi-topic documents be broken into multiple documents, and that multiple individual topics be combined into a single document.

Chunking is necessarily output transformation specific with chunked output required for some and not supported for other types of output. Chunking is also implementation specific with some implementations supporting some, but not all, chunking methods, or adding new implementation specific chunking methods to the standard methods described in this specification.

The value of the chunk attribute consists of one or more space delimited tokens:

by-topic

When the chunk attribute value includes the "by-topic" token, a chunking policy is established for the current topicref element where a separate output chunk is produced for the target topic and each of its descendants. The policy only applies for a chunk action of the current element (for example, to-content), except when it is set on the map element, when the "by-topic" policy is established for the entire map.

by-document

When the chunk attribute value includes the "by-document" token, a chunking policy is established for the current topicref element where a single output chunk is produced for the referenced document. The policy only applies for a chunk action of the current element (for example, to-content), except when it is set on the map element, when the "by-document" policy is established for the entire map.

select–topic

When the chunk attribute value includes the “select–topic” token, an individual topic, without any other topics (ancestors, descendents, or peers) from within the same document, is selected.

select–document

When the chunk attribute value includes the “select–document” token, the content for the referenced topic, as well as any other topics (ancestors, descendents, or peers) contained within the same document are selected.

select–branch

When the chunk attribute value includes the “select–branch” token, an individual topic as well as any nested topics it contains are selected.

to–content

When the chunk attribute value includes the “to–content” token, processing generates a new chunk of content.

to–navigation

When the chunk attribute value includes the “to–navigation” token, processing generates new chunk of navigation (toc, related-links).

Note that the set of select–xxxxx token values are only useful when addressing a topic in a document that contains multiple topics.

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

There is no default value for the chunk attribute and the chunk attribute does not inherit values from container elements. A default for an entire map may be established by setting the chunk attribute on the map element.

When no chunk attribute values are given, chunking behavior is implementation dependent and may vary for different implementations. When variations of this sort are not desired, a default for the entire map may be established by including a chunk attribute value on the map element.

When creating new documents via chunk processing, the storage object name or identifier (if relevant) is taken from the copyto attribute if set, otherwise the root name is taken from the id attribute if the by-topic policy is in effect and from the name of the referenced document if the by-document policy is in effect.

Examples

Given several single topic documents, parent1.dita, parent2.dita, ..., child1.dita, child2.dita, ..., grandchild1.dita, grandchild2.dita containing topics with ids P1, P2, ..., C1, C2, ..., GC1, GC2, ..., several nested topic documents, nested1.dita, nested2.dita, ..., each containing two topics, parent topics with ids N1, N2, ... and child topics with ids N1a, N2a, ... nested within the parent, and database.dita with the following contents:

```
<dita>
  <topic id="X"/>
  <topic id="Y">
    <topic id="Y1">
      <topic id="Y1a"/>
    </topic>
    <topic id="Y2"/>
  </topic>
  <topic id="Z">
    <topic id="Z1"/>
  </topic>
</dita>
```

map1.ditamap:

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

Produces a single output document, parent1.xxxx containing topic P1 with topic Y1, but not topic Y1a nested in it.

map2.ditamap:

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

Produces a single output document, parent1.xxxx containing topic P1, topic Y1 nested within topic P1, and topic Y1a nested within Y1.

map3.ditamap:

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

Produces a single output document, P1.xxxx, containing topic P1 and topics X, Y, and Z together with their children nested within topic P1.

map4.ditamap:

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

Produces a single output document named parentchunk.xxxx containing topic P1 with topic N1 nested within P1 and topic N1a nested within N1.

map5.ditamap:

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

Produces two output documents: the P1, C1, C3, and GC3 topics in parentchunk.xxxx, and the C2 and GC2 topics in child2chunk.xxx.

map6.ditamap:

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

Produces a single output document, nestedchunk.xxxx, which contains topic N1 with no topics nested within.

map7.ditamap:

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

Produces two navigation chunks, one for P1, C1, ... and a second for P2, C2,

The above example identifies a “how to” for setting up a product as a single unit. The “how to” might be provided both as navigable HTML pages and as a printable PDF attached to the root HTML page.

Implementation specific tokens and future considerations

Additional chunk tokens may be added to the DITA Standard in the future. Additional implementation specific tokens may be defined as well. 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 chunking method name. For example: “acme:level2” could be a token for the Acme DITA Toolkit that requests the “level2” chunking method.

Translation

DITA has specific capabilities for preparing content for translation and working with multilingual content, including the xml:lang attribute, the dir attribute, and the translate attribute.

The xml:lang attribute

Specifies the language (and optionally the locale) of the element content. The intent declared with xml:lang is considered to apply to all attributes and content of the element where it is specified, unless overridden with an instance of xml:lang on another element within that content. When no xml:lang value is supplied, the processor should assume a default value.

This attribute must be set to a language identifier, as defined by IETF RFC 4646 (<http://www.ietf.org/rfc/rfc4646.txt>) or successor.

Recommended usage

For a DITA document that contains a single language, the highest level element containing content should always set the `xml:lang` attribute to the language (and optionally the locale) that applies to the document. Since the `dita` element does not support the `xml:lang` element, the highest level element that should set the `xml:lang` attribute is the `topic` element (or derivatives at the same level).

For a DITA document that contains more than one language, the highest level element should always set the `xml:lang` attribute to the primary language (and optionally the locale) that applies to the document. Wherever an alternate language occurs in the document, the element containing the text or structure in the alternate language should set the `xml:lang` attribute appropriately. The above way of overriding the default document language applies to both block and inline elements that use the alternate language.

Using markup to identify language is strongly recommended to make the document as portable as possible. In addition, the marked-up document can be read and understood by humans. Finally, when updating the document, the boundaries of each language are clear, which makes it much easier for the author to update the document.

Applications should ensure every highest level `topic` element and the root `map` element explicitly assign the `xml:lang` attribute.

Usage in maps

The `xml:lang` attribute can be specified on the `map` element. The expected language inheritance behavior on the `map` is similar to that on the `topic`. That is, the primary language for the `map` should be set on the `map` element (or assumed by the application if not explicitly set), and should remain in effect for all children unless a child specifies a different value for `xml:lang`.

If the `xml:lang` value on a `topicref` does not match the `xml:lang` value on a `topic`, the value on the `topic` takes precedence.

Usage with the `conref` attribute

When `conref` is used to include content from one `topic` in another `topic`, the `xml:lang` value must be obtained from the `topic` being included. If the included content does not explicitly set `xml:lang`, the processor must obtain the `xml:lang` value from the nearest parent of the included content. If the included content does not have a parent element that sets `xml:lang`, the application should default to the same value used for `topics` that do not set the `xml:lang` attribute.

This behavior is shown in the following example, where the `xml:lang` value of the included note is obtained from its ancestor `section` element (`id="qqwwee"`) that sets `xml:lang`. In this example, the `xml:lang` value applied to the note whose `id` is `"mynote"` is `"fr"`.

```
*****installingAcme.xml*****
<?xml version="1.0"?>
<!DOCTYPE dita PUBLIC "-//OASIS//DTD DITA Composite//EN" "dita.dtd">
<dita>
  <topic id="topic_3FD87D" xml:lang="en">
    <title>Installing Acme</title>
    <shortdesc>Step by step details on how to install Acme</shortdesc>
    <body>
      <p id="p_60A72">Welcome message goes here</p>
      <section id="section_C25">
        <title>Before you begin</title>
        <p id="p_E57324D">Special notes when installing Acme in
          France:</p>
```

```

        <note id="mynote" conref="warningsAcme.xml#topic_warnings/frenchwarnings"></note>
    </section>
</body>
</topic>
</dita>
*****

*****warningsAcme.dita*****
<?xml version="1.0"?>
<!DOCTYPE dita PUBLIC "-//OASIS//DTD DITA Composite//EN" "database.dtd">
<dita>
    <topic id="topic_warnings">
        <title>Warnings</title>
        <shortdesc>warnings in all languages</shortdesc>
        <body>
            <section id="qqwwee" xml:lang="fr">
                <title>French warnings</title>
                <p id="p_F2A">These are our French warnings.</p>
                <note id="frenchwarnings">Note in French!</note>
            </section>
            <section xml:lang="en" id="aassdd">
                <title>English warnings</title>
                <p id="p_5F961">These are our English warnings.</p>
                <note id="englishwarnings">Note in English!</note>
            </section>
        </body>
    </topic>
</dita>
*****

```

The dir attribute

While most languages are written in text where characters flow from left to right, Hebrew and many Arabic languages are written from right to left. In some languages, including Hebrew and Arabic, numbers and other content is written left to right. Also, a multilingual document containing, for example, English and Hebrew, contains some text that flows left to right and other text that flows right to left.

Text directionality is controlled by the following:

1. Directionality explicitly set on the root element (via the `dir` attribute) or, when not set, assumed by the processing application.
2. `dir="ltr|rtl"` attribute on an element that overrides the inherited direction. The specified direction overrides the Unicode bidirectional algorithm only on neutral Unicode characters (e.g. spaces and punctuation) in the element's content. The `"ltr"` and `"rtl"` values do not override the strongly bidirectional characters.
3. `dir="lro|rlo"` attribute on an element. The specified direction overrides the Unicode bidirectional algorithm on all Unicode characters in the element's content.

In most cases, authors need to use `dir="rtl|ltr"` to ensure punctuation surrounding a RTL phrase inside a LTR element is rendered correctly. In order to override the direction of strongly typed Unicode characters (most characters that apply to a language except for punctuation, spaces and digits), the author would need to use `dir="lro|rlo"`. The use of the `dir` attribute and the Unicode algorithm is clearly explained in the article [Specifying the direction of text and tables: the dir attribute](http://www.w3.org/TR/html4/struct/dirlang.html#ade-dir) (<http://www.w3.org/TR/html4/struct/dirlang.html#ade-dir>). The referenced article has several examples on the use of `dir="rtl|ltr"`. There is no example on the use of `dir="lro|rlo"`, though it can be inferred from the example using the `bdo` element (the old W3C way of overriding the entire Unicode bidirectional algorithm; the now favored method uses the override values on the `dir` attribute).

From the HTML 4.0 spec:

The `dir` attribute specifies the directionality of text: left-to-right (`dir="ltr"`, the default) or right-to-left (`dir="rtl"`). Characters in Unicode are assigned a directionality, left-to-right or right-to-left, to allow the text to be rendered properly. For example, while English characters are presented left-to-right, Hebrew characters are presented right-to-left. Unicode defines a bidirectional algorithm that must be applied whenever a document contains right-to-left characters. While this algorithm usually gives the proper presentation, some situations leave directionally neutral text and require the `dir` attribute to specify the base directionality. Text is often directionally neutral when there are multiple embeddings of content with a different directionality. For example, an English sentence that contains a Hebrew phrase that contains an English quotation would require the `dir` attribute to define the directionality of the Hebrew phrase. The Hebrew phrase, including the English quotation, would be contained within a `ph` element with `dir="rtl"`.

Recommended usage

The Unicode Bidirectional algorithm provides for various levels of bidirectionality, as follows:

1. Directionality is either explicitly specified via the `dir` attribute on the highest level element (topic or derived peer for topics, map for ditamaps) or assumed by the processing application. It is recommended to specify the `dir` attribute on the highest level element in the topic or document element of the map.
2. When embedding a RTL text run inside a LTR text run (or vice-verse), the default direction often provides incorrect results, 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 may appear at the beginning of the question instead of at the end). 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|rtl"` values override only the neutral characters, not all Unicode characters.
3. Sometimes you may want to override the default directionality for strongly bidirectional characters. This is done using the `"lro"` and `"rlo"` values, which overrides the Unicode directionality algorithm. This essentially forces a direction on the contents of the element. These override attributes give the author a brute force way of setting the directionality independently of the Unicode BIDI algorithm. The gentler `"ltr|rtl"` values have a less radical effect, only effecting punctuation and other so-called neutral characters.

For most authoring needs, the `"ltr"` and `"rtl"` values are sufficient. Only when the desired effect cannot be achieved using these values, should the override values be used.

While the Unicode standard includes hidden markers for directionality without the need for markup, these markers should not be used. It is strongly recommended to mark up the document using the `dir` attribute to set directionality. Using markup instead of the Unicode markers has the following advantages:

- The document will be as portable as possible.
- The document can be processed by applications that do not fully implement the Unicode BIDI algorithm.
- The marked-up document can be read and understood by humans.
- When updating the document, the boundaries of each text flow are clear, which makes it much easier for the author to update the document.

Implementation precautions





Users should be aware that descriptive markup isn't necessarily the end of their work. Each possible output rendition or display tool may have different requirements for managing bidirectional text. Just as different HTML browsers offer different levels of support for CSS, different output tools implement the bidirectional algorithm, and its accompanying directional controls, differently. For example, HTML displayed in Internet Explorer may have different requirements than HTML displayed in Firefox. Similarly, a control that works in one part of an HTML file, such as the body of the page, might not work in another, such as the title or the index in compiled HTML Help. The same uncertainty can be found in almost any output. PostScript or PDF rendering tools treat bidirectional text differently. Microsoft Word and OpenOffice Writer don't handle bidirectional RTF in the same way. Flash has little concern for directional markup of any kind, but does format strings according to the Unicode algorithm.

Because input is unpredictably dependent on eventual output, it is not sufficient to apply the "dir" attribute in such a way as to make the XML appear as it should in an editor. Additional care must be taken to make sure that markup is correctly transformed (or added to the source XML, if needed), with respect both to the target output format and the target output tool. To use the case of HTML, this could mean creating output tailored to the capabilities of the most common likely browser or creating output tailored to the least capable browser and ensuring the markup functions for the most likely and capable one. For example, bidirectional HTML that displays perfectly in Internet Explorer might not display correctly in Safari. However, if the HTML displays perfectly in Safari, chances are very good it will display correctly in Internet Explorer as well. This isn't a certainty, however. Each case should be tested and confirmed by qualified individuals.

Applications that process DITA documents, whether at the authoring, translation, publishing, or any other stage, should fully support the Unicode algorithm to correctly implement the script and directionality for each language used in the document. The recommended practice is to write all directionality markers via XML markup and not to use the Unicode Bidirectional markers. When reading XML markup that embeds the Unicode Bidirectional markers, these markers should be replaced with markup when the document is saved.

Applications should ensure every highest level topic element and the root map element explicitly assign the dir attribute.

Related information

-  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>)

All elements with translation properties

This contains a list of all current TC-approved elements, divided up by module. It describes which are usually block elements and which are usually inline, and lists which elements should typically be translated by default.

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 may be a block in one context and an inline element in another. Specializing document types may 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 default should be 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 may 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 derivations of keyword) is an inline, phrase-like element when it appears in the body of a document. It may also appear in the keywords element in topicmeta (for maps) or in the prolog (for topic). When it appears in the <keywords> element, each keyword represents an individual segment, and is not part of a larger segment; in that location, keyword can be considered a "subflow" element.

topic elements

Element name	Specialized from	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
abstract (<i>New in DITA 1.1</i>)	N/A	block	block	yes	
alt	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	
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	

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 may 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.
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.

Element name	Specialized from	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
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 (<i>New in DITA 1.1</i>)	N/A	N/A (metadata)	block	no (likely to change for some specializations)	
data-about (<i>New in DITA 1.1</i>)	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	
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 (<i>New in DITA 1.1</i>)	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 (<i>New in DITA 1.1</i>)	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
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	

Element name	Specialized from	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
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
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	
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	

Element name	Specialized from	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
ul	N/A	block	block	yes	@spectitle ²
unknown (<i>New in DITA 1.1</i>)	N/A	block	block	no	
vrml	N/A	block (metadata)	block	yes	
vrmlist	N/A	block (metadata)	block	yes	
xref	N/A	inline	inline	yes	

map elements

Element name	Specialized from	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
anchor	N/A	n/a	n/a	n/a	
linktext	N/A	block	block	yes	
map	N/A	block	block	yes	@title
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

bookmap elements (*new in DITA 1.1*)

The bookmap specialization contains many phrase-based elements inside the bookmeta. These are metadata, and should not be translated.

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
abbrevlist	topicref	yes	block	block	yes	@navtitle
amendments	topicref	yes	block	block	yes	@navtitle
appendix	topicref	yes	block	block	yes	@navtitle
approved	data	yes	block	block	no	
backmatter	topicref	yes	block	block	yes	
bibliolist	topicref	yes	block	block	yes	@navtitle
bookabstract	topicref	yes	block	block	yes	@navtitle
bookchangehistory	data	yes	block	block	no	
bookevent	data	yes	block	block	no	
bookeventtype	data	yes	block	block	no	
bookid	data	yes	block	block	no	
booklibrary	ph	yes	inline	inline	yes	
booklist	topicref	yes	block	block	yes	@navtitle

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
booklists	topicref	yes	block	block	yes	
bookmark	map	no	block	block	yes	<i>removed title attribute</i>
bookmeta	topicmeta	yes	block	block	yes	
booknumber	data	yes	block	block	no	
bookowner	data	yes	block	block	no	
bookpartno	data	yes	block	block	no	
bookrestriction	data	yes	block	block	no	
bookrights	data	yes	block	block	no	
booktitle	title	yes	block	block	yes	
booktitlealt	ph	yes	inline	inline	yes	
chapter	topicref	yes	block	block	yes	@navtitle
colophon	topicref	yes	block	block	yes	@navtitle
completed	ph	no	inline	inline	no	
copyrfirst	data	yes	block	block	no	
copyrlast	data	yes	block	block	no	
day	ph	no	inline	inline	no	
dedication	topicref	yes	block	block	yes	@navtitle
draftintro	topicref	yes	block	block	yes	@navtitle
edited	data	yes	block	block	no	
edition	data	yes	block	block	no	
figurelist	topicref	yes	block	block	yes	@navtitle
frontmatter	topicref	yes	block	block	yes	@navtitle
glossarylist	topicref	yes	block	block	yes	@navtitle
indexlist	topicref	yes	block	block	yes	@navtitle
isbn	data	yes	block	block	no	
mainbooktitle	ph	yes	inline	inline	yes	
maintainer	data	yes	block	block	no	
month	ph	no	inline	inline	no	
notices	topicref	yes	block	block	yes	@navtitle
organization	data	yes	block	block	no	
part	topicref	yes	block	block	yes	@navtitle
person	data	yes	block	block	no	
preface	topicref	yes	block	block	yes	@navtitle
printlocation	data	yes	block	block	no	
published	data	yes	block	block	no	
publisherinformation	publisher	yes	block	block	yes	
publishtype	data	yes	block	block	no	
reviewed	data	yes	block	block	no	
revisionid	ph	no	inline	inline	no	
started	ph	no	inline	inline	no	
summary	ph	yes	inline	inline	yes	
tablelist	topicref	yes	block	block	yes	@navtitle
tested	data	yes	block	block	no	
toc	topicref	yes	block	block	yes	@navtitle
trademarklist	topicref	yes	block	block	yes	@navtitle
volume	data	yes	block	block	no	
year	ph	no	inline	inline	no	

concept elements

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
conbody	body	yes	block	block	yes	
concept	topic	yes	block	block	yes	

glossary elements *(new in DITA 1.1)*

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
glossdef	abstract	yes	block	block	yes	
glossentry	topic	yes	block	block	yes	

reference elements

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
propdesc	stentry	yes	block	block	yes	@specentry ²
propdeschd	stentry	yes	block	block	yes	@specentry ²
properties	simpletable	yes	block	block	yes	@spectitle ²
property	strow	yes	block	block	yes	
prophead	sthead	yes	block	block	yes	
proptype	stentry	yes	block	block	yes	@specentry ²
proptypehd	stentry	yes	block	block	yes	@specentry ²
propvalue	stentry	yes	block	block	yes	@specentry ²
propvaluehd	stentry	yes	block	block	yes	@specentry ²
refbody	body	yes	block	block	yes	
reference	topic	yes	block	block	yes	
refsyn	section	yes	block	block	yes	@spectitle ²

task elements

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
chdesc	stentry	yes	block	block	yes	@specentry ²
chdeschd	stentry	yes	block	block	yes	@specentry ²
chhead	sthead	yes	block	block	yes	
choice	li	yes	block	block	yes	
choices	ul	yes	block	block	yes	<i>removes @spectitle</i>
choicetable	simpletable	yes	block	block	yes	@spectitle ²
choption	stentry	yes	block	block	yes	@specentry ²
choptionhd	stentry	yes	block	block	yes	@specentry ²
chrow	strow	yes	block	block	yes	
cmd	ph	NO	inline	block	yes	
context	section	yes	block	block	yes	<i>removes @spectitle</i>
info	itemgroup	NO	inline	block	yes	

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
postreq	section	yes	block	block	yes	<i>removes @spectitle</i>
prereq	section	yes	block	block	yes	<i>removes @spectitle</i>
result	section	yes	block	block	yes	<i>removes @spectitle</i>
step	li	yes	block	block	yes	
stepresult	itemgroup	NO	inline	block	yes	
steps	ol	yes	block	block	yes	<i>removes @spectitle</i>
steps-unordered	ul	yes	block	block	yes	<i>removes @spectitle</i>
stepxmp	itemgroup	NO	inline	block	yes	
substep	li	yes	block	block	yes	
substeps	ol	yes	block	block	yes	<i>removes @spectitle</i>
task	topic	yes	block	block	yes	
taskbody	body	yes	block	block	yes	
tutorialinfo	itemgroup	NO	inline	block	yes	

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	
i	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	

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	

pr-d elements (programming domain)

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
apiname	keyword	yes	inline	inline	yes	
codeblock	pre	yes	block	block	yes	@spectitle ²
codeph	ph	yes	inline	inline	yes	
delim	ph	yes	inline	inline	yes	
fragment	figgroup	yes	block	block	yes	
fragref	xref	yes	inline	inline	yes	

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
groupchoice	figgroup	yes	block	block	yes	
groupcomp	figgroup	yes	block	block	yes	
groupseq	figgroup	yes	block	block	yes	
kwd	keyword	yes	inline	inline	yes	
oper	ph	yes	inline	inline	yes	
option	keyword	yes	inline	inline	yes	
parml	dl	yes	block	block	yes	@spectitle ²
parmname	keyword	yes	inline	inline	yes	
pd	dd	yes	block	block	yes	
plentry	dlentry	yes	block	block	yes	
pt	dt	yes	block	block	yes	
repsep	ph	yes	inline	inline	yes	
sep	ph	yes	inline	inline	yes	
synblk	figgroup	yes	block	block	yes	
synnote	fn	yes	block	block	yes	
synnoteref	xref	yes	inline	inline	yes	
synph	ph	yes	inline	inline	yes	
syntaxdiagram	fig	yes	block	block	yes	removes @spectitle
var	ph	yes	inline	inline	yes	

sw-d elements (software domain)

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
cmdname	keyword	yes	inline	inline	yes	
filepath	ph	yes	inline	inline	yes	
msgblock	pre	yes	block	block	yes	@spectitle ²
msgnum	keyword	yes	inline	inline	yes	
msgph	ph	yes	inline	inline	yes	
systemoutput	ph	yes	inline	inline	yes	
userinput	ph	yes	inline	inline	yes	
varname	keyword	yes	inline	inline	yes	

ui-d elements (UI domain)

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
menucascade	ph	yes	inline	inline	yes	
screen	pre	yes	block	block	yes	@spectitle ²
shortcut	keyword	yes	inline	inline	yes	
uicontrol	ph	yes	inline	inline	yes	
wintitle	keyword	yes	inline	inline	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	

mapgroup-d elements (mapgroup domain)

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
topicgroup	topicref	yes	block	block	yes	@navtitle
topichead	topicref	yes	block	block	yes	@navtitle

xnal-d elements (XNAL domain) *(new in DITA 1.1)*

The XNAL information is all metadata, so it generally does not need to be translated. Exceptions may be needed when selections from this metadata are used for display purposes. The standard behavior may need to change based on local business rules. For example, in some cases it may be appropriate to translate the honorific, country, or organizationname elements.

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
addressdetails	ph	no	<i>block</i>	<i>block</i>	no	
administrativearea	ph	no	<i>block</i>	<i>block</i>	no	
authorinformation	author	no	block	block	no	
contactnumber	data	no	block	block	no	
contactnumbers	data	no	block	block	no	
country	ph	no	<i>block</i>	<i>block</i>	no	
emailaddress	data	no	block	block	no	
emailaddresses	data	no	block	block	no	
firstname	data	no	block	block	no	
generationidentifier	data	no	block	block	no	
honorific	data	no	block	block	no	
lastname	data	no	block	block	no	
locality	ph	no	<i>block</i>	<i>block</i>	no	
localityname	ph	no	<i>block</i>	<i>block</i>	no	
middlename	data	no	block	block	no	
namedetails	data	no	block	block	no	
organizationinfo	data	no	block	block	no	
organizationname	ph	no	<i>block</i>	<i>block</i>	no	
organizationnamedetails	ph	no	<i>block</i>	<i>block</i>	no	
otherinfo	data	no	block	block	no	
personinfo	data	no	block	block	no	
personname	data	no	block	block	no	

Element name	Specialized from	Inherits everything from ancestor?	Block/Inline (presentation)	Block/Inline (translation)	Translatable content?	Translatable attributes?
postalcode	ph	no	<i>block</i>	<i>block</i>	no	
thoroughfare	ph	no	<i>block</i>	<i>block</i>	no	
url	data	no	block	block	no	
urls	data	no	block	block	no	

Chapter 5. DITA specialization

Specialization is the process by which new designs are created based off existing designs, allowing new kinds of content to be processed using existing processing rules.

Specialization provides a way to reconcile the needs for centralized management of major architecture and design with the needs for localized management of group-specific and content-specific guidelines and behaviors. Specialization allows multiple definitions of content and output to co-exist, related through a hierarchy of types and transforms. This hierarchy lets general transforms know how to deal with new, specific content, and it lets specialized transforms reuse logic from the general transforms. As a result, any content can be processed by any transform, as long as both content and transform are specialization-compliant, and part of the same hierarchy. Specializers get the benefit of specific solutions, but also get the benefit of common standards and shared resources.

Content	Processing	Result
Unspecialized	Unspecialized	Base processing, expected output
Unspecialized	Specialized	Base processing, specialized overrides are ignored, expected output
Specialized	Unspecialized	Base processing, specialized content treated as general, output may fall short of expectations
Specialized	Specialized	Specialized processing, expected output
Specialized	Differently specialized	Some specialized processing, specialized content treated as nearest common denominator, output may fall short of expectations

The following topics provide an overview of specialization, some recommendations for use, and detailed rules for its mechanisms.

What is specialization?

Specialization allows you 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 is used when new structural types or new domains are needed. DITA specialization can be used when you want to make changes to your design for the sake of increased consistency or descriptiveness or have extremely specific needs for output that cannot be addressed using the current data model. Specialization is not recommended for simply creating different output types as DITA documents may be transformed to different outputs without resorting to specialization (see “Customization” on page 88).

There are two kinds of specialization hierarchy: one for structural types (with topic or map at the root) and one for domains (with elements in topic or map at their root, or the attributes props or base). Structural types define topic or map structures, such as concept or task or reference, which often apply across subject areas (for example, a user interface task and a programming task may both consist of a series of steps). Domains define markup for a particular information domain or subject area, such as programming, or hardware. Each of them represent 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.

Use specialization when you are dealing with new semantics (new, meaningful categories of information, either in the form of new structural types or new domains). The new semantics can be encoded as part of a specialization hierarchy, that allows them to be transformed back to more general equivalents, and also ensures that the specialized content can be processed by existing transforms.

Why specialization?

Specialization can have dramatic benefits for the development of new document architectures.

Among the benefits:

- No need to reinvent the base vocabulary - Create a module in 1/2 day with 10 lines vs. 6 months with 100s of lines; allows faster design iterations and scalable prototyping; automatically pick up changes to the base
- No impact from other designs that customize for different purposes - Avoid enormous, kitchen-sink vocabularies; Plug in the modules for your requirements
- Interoperability at the base type - Guaranteed mapping from specialized type documents to base type documents
- Reusable type hierarchies - Share understanding of information across groups, saving time and presenting a consistent picture to customers
- Output tailored to customers and information - More specific search, filtering, and reuse that is designed for your customers and information not just the common denominator
- Consistency - Both with base standards and within your information set
- Learning support for new writers - Instead of learning standard markup plus specific ways to apply the markup, writers get specific markup with guidelines built in
- Explicit support of different product architectural requirements - Requirements of different products and architectures can be supported and enforced, rather than suggested and monitored by editorial staff

Structural versus domain specialization

Structural specialization defines new types of structured information, such as new topic types or new map types. Domain specialization creates new markup that can be useful in multiple structural types, such as new kinds of keywords, tables, or lists, or new attributes such as conditional processing attributes.

Structural types define structures for modules of information, such as concept or task or reference, which often apply across subject areas (for example, a user interface task and a programming task may both consist of a series of steps). When new elements are introduced through structural specialization, the elements that contain the new elements must be specialized as well; and the new container elements must have their containers specialized in turn, all the way to the root element for the module (for example, the <topic> element or <map> element).

Domains typically define markup for a particular domain or subject area, such as programming, or hardware. Domain elements become available wherever their ancestor elements are allowed once the domains are integrated with the structural specializations in a document type. Domain attributes (based off of props or base) become available wherever the props or base attributes are allowed.

Both structural specialization hierarchies and domain specialization hierarchies are “is a” hierarchies, 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 programming domain is still concerned with programming.

Structural and domain hierarchies must share a common base module in order to be integrated together. For example, domains for use across topic types must ultimately be specialized off of elements or the specializable attributes in <topic>; domains for use across both topic types and map types must be specialized off of the elements or specializable attributes that are common to both types.

With the exception of the common base modules (topic and map), a domain cannot be specialized from a structural type. For example, a domain cannot be specialized from elements in <task>, only from the root structural modules for <topic> or <map>. This rule ensures that domains can be integrated and document types can be generalized predictably.

Elements and attributes created by specialization are scoped by the name of the structural type or domain in which they were declared. For structural types, the name is the same as the root element: for example, task is the name of the structural type whose root element is <task>. For domains, the name is not shared with any element, but is assigned by the developer of the specialization. By convention, domain names end with "-d" and are kept short; for example, ui-d for the user interface domain and pr-d for the programming domain. Attribute domains are a special case, typically named after the attribute being defined, with the addition of "-a".

Specializing foreign or unknown content

Specializing <foreign> or <unknown> elements is an open extension to the DITA Architecture that allows DITA adopters to incorporate new or existing standard vocabularies for non-textual content, like MathML and SVG, as in-line objects.

Incorporating foreign or unknown content

There are three methods in which a DITA adopter can incorporate foreign content into DITA.

- Implement <foreign> or <unknown> as a domain specialization.
- Implement <foreign> or <unknown> as part of a structural specialization.
- Do nothing; simply embed the foreign content within <foreign> or <unknown>.

Specialization of <foreign> or <unknown> should usually be implemented as a domain, but for those looking for more control over the content can implement foreign vocabulary as part of a structural specialization.

Embedding the content within an unspecialized <foreign> or <unknown> element, because of the ANY content model, offers the least amount of control over the content and hinders interoperability.

Foreign or unknown content and the architectural class attribute

The class attribute is used for a specific purpose in DITA. It identifies the specialization module for the element type as well as the ancestor element types and the specialization modules to which they belong. Every DITA element has to have a class attribute. Since the elements defined within the foreign vocabulary are non-DITA elements, there is no requirement for those elements to have an architectural class attribute. Elements that specialize <foreign> or <unknown>, therefore a DITA element, are required to have a class attribute.

Specializing foreign or unknown content using DTDs.

For more specific information on specializing DTDs, see Modularization in DTDs

```
<!-- declaration for the specialized wrapper -->
<!ENTITY % svg "svg">

<!-- included SVG document type -->
<!ENTITY % SVG.prefix "svg" >
<!ENTITY % svg-qname.mod
    PUBLIC "-//W3C//ENTITIES SVG 1.1 Qualified Name//EN"
        "svg-qname.mod" >
```

```

<!-- definition for the specialized wrapper -->
<!ELEMENT svg ((%SVG.svg.qname;)>
<!-- ATTLIST svg %global-atts;
      class CDATA "+ topic/foreign svg-d/svg "
>

```

An SVG example within a paragraph element

```

<p>This is an ellipse.
  <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>

```

Specializing foreign content using XML Schemas

For more specific information on specializing XML Schemas, see Modularization in schemas

```

<!-- importing MathML document type -->
<xs:import namespace="http://www.w3.org/1998/Math/MathML" schemaLocation="mathml2.xsd">

<!-- definition for the specialized wrapper -->
<xs:element name="mathML" type="mathML.class" />
<xs:complexType name="mathML.class">
  <xs:choice>
    <xs:element ref="mml:math" />
  </xs:choice>
  <xs:attribute name="outputclass" type="xs:string"/>
  <xs:attributeGroup ref="univ-atts"/>
  <xs:attributeGroup ref="global-atts"/>
  <xs:attribute ref="class" default="+ topic/foreign mathML/mathML"/>
</xs:complexType>

<!-- definition for each element extended by the domain -->
<xs:group name="ma-d-foreign">
  <xs:choice>
    <xs:element ref="mathML" />
  </xs:choice>
</xs:group>

<!-- definition for the named model groups -->
<xs:group name="foreign">
  <xs:choice>
    <xs:group ref="foreign"/>
    <xs:group ref="ma-d-foreign"/>
  </xs:choice>
</xs:group>

```

An XML Schema MathML specialization example

Use of a specialized <foreign> element with a <object> element using MathML.

```

<p>... as in the formula
<object>
  <desc>4 + x</desc>
  <mathML>
    <mml:math display="block">
      <mml:mrow>
        <mml:mo>&sum;</mml:mo>
        <mml:mn>4</mml:mn>
        <mml:mo>+</mml:mo>
        <mml:mi>x</mml:mi>
      </mml:mrow>
    </mml:math>
  </mathML>
</object>

```



```
</mm1:math>
</mathML>
<object>.
</p>
```

Data extensibility

The `<data>` element represents properties ranging from simple values to complex structures. Processes can harvest the `<data>` element for automated manipulation or to format data associated with the body flow. The `<data>` element is primarily intended for use in creating specializations.

You can nest `<data>` elements for structures. You can use the `name` attribute to indicate the semantic of instances of the `<data>` element such as addresses, times, amounts, and so on. In many cases, however, you may prefer to specialize the `<data>` element for more precise semantics and for constraints on structures and values. For instance, a specialization can specify an enumeration for the `value` attribute.

In some cases, it isn't possible or convenient to maintain a property as part of the content of its subject. For instance, you might prefer to maintain extensive data in the `<prolog>` that applies to a note or example within the body. To handle such exceptions, you can use the `<data-about>` element to identify the subject of the property.

A process can harvest the data values for a machine-processable representation such as RDF. The default formatting ignores the `<data>` element within the `<body>` element. Understanding whether and how the properties should display, customized or specialized processing can extend formatting to include data values in some formatted outputs.

CAUTION:

It is an abuse of the DITA architecture to specialize `<data>` element for text that is part of the body flow, particularly to escape the constraints of the base content models. For example, a special kind of paragraph must specialize the base `<p>` element rather than the `<data>` element. When exchanging content with others or retiring a specialization, a paragraph specialized from the `<data>` element will be generalized and thus skipped by the base formatting, mangling the discourse flow and resulting in invalid content.

Applications

Uses of the `<data>` element include the following:

- Complex metadata properties such as bibliographic records corresponding to citations.
- Hybrid documents with data values as part of the content. (Word processor formats using form fields provide an example of such hybrid documents.)
- 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 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/>).

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>`. These properties wouldn't appear in the formatted output (except perhaps for debugging problems in the refresh):

```

<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>

```

The following example identifies a real estate property as part of a house description. The `<realEstateProperty>` element and everything it contains are specialized from `<data>`. The `<houseDescription>` element is specialized from `<section>`. A specialized process can format the values to identify the lot if appropriate for the brochure.

```

<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>

```

Limits of specialization

There are times when a new structural or domain type appears not to fit into the existing hierarchy, based on the semantics of the existing types and the restrictions of the specialization process. In these cases there are a variety of options to consider.

The basic specialization mechanism used by the DITA document types can also be used for non-DITA document types in order to provide the same re-use, specialization, and interoperation benefits that one can get from the DITA document types, but restricted to the specific domain within which the new document types apply. Note that even if one uses the DITA-defined types as a starting point, any change to those base types not accomplished through specialization defines a completely new document type that has no meaningful or normative relationship to the DITA document types and cannot be considered in any way to be a conforming DITA application. In other words, the use of DITA specialization from non-DITA base types does not produce DITA-compliant document types.

However, given the substantial benefits of building from the common DITA base classes (including the ability to generalize to a common format, use of standards-compliant tools and processes, and reuse of content across document types through DITA maps and conref) there are some techniques to consider before complete departure from the DITA content architecture.

Specialize from generic elements or attributes

The first option to consider is to choose more generic base elements or attributes from the available set. There are generic elements in DITA available at every level of detail, from whole generic topics down to individual generic keywords, and the generic attribute base 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 ``, ``, `<sl>`, or `<dl>`, you can create a new set of list elements by specializing nested `<ph>` elements. This new list structure will not be semantically tied to the other lists by ancestry, and so will require

specialized processing to receive appropriate output styling. However, it will remain a valid DITA specialization, with the standard support for generalization, content referencing, conditional processing, and so forth.

The following base elements in <topic> are generic enough to support almost any structurally valid 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-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

keyword

any non-nesting division of content below the paragraph level

data any content designed for internal processing rather than human-readable output

foreign

any content that already has a non-DITA markup standard but needs to be authored as part of the DITA document

unknown

any non-standard markup that does not fit the DITA model but needs to be managed as part of a DITA document

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 doesn't already have a semantic equivalent

You should specialize from the semantically closest match whenever possible. When some structural requirement forces you to pick a more general ancestor, please inform the technical committee: over time a richer set of generic elements should become available.

Customized subset document types for authoring

DITA markup is organized into domain and structural type modules so that authoring groups can easily select the markup subset they require by creating a new document type shell. However, when an authoring group requires a subset of markup rules that does not follow the boundaries of the type modules (for example, global removal of certain attributes or elements), you can if necessary create a customized document type for the sake of enforcing these rules at authoring time, as long as the document types are validated using a standards-compliant document type at processing time.

A customized subset document type should be created without editing of the type modules. The document type shell can override entities in the module files, including attributes and content models, by providing a new definition of the entity before importing the module files.

Customized subset document types are not conformant with the DITA standard, and may not be supported by standards-conformant tools. For example, if you customize document types to remove the <xref> element, you may not be able to use off-the-shelf DITA editors to create your content.

Map from customized document type to DITA during preprocessing

While specialization can be used to adapt document types for many different authoring purposes, there are some authoring requirements that cannot be met through specialization, in particular, splitting or renaming attributes, or renaming elements through specialization without also specializing their containers.

In cases where structural and domain specialization of elements or attributes is not sufficient, and where the new document type can be straightforwardly and reliably transformed to a standard document type, the authoring group may be best served by a customized document type that is transformed to a standard document type as part of the publishing pipeline. 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>` and then preprocessed to rename `<paragraph>` back to `<p>` before feeding the documents into a standard publishing process.

A customized document type should be created without editing of the type modules. The document type shell can override entities in the module files, including attributes and content models, by providing a new definition of the entity before importing the type module files.

Customized document types are not compliant with the DITA standard, and will not be supported by standards-compliant tools. Preprocessing can ensure compatibility with existing publishing processes, but does not ensure compatibility with DITA-supporting authoring tools or content management systems. However, when an implementation is being heavily customized in any case, a customized document types can help isolate and control the implications of non-standard design in a customized implementation.

Specialization in content

Specialization is expressed in content through the use of two attributes: the class attribute and the domain attribute. These are not typically present in the document instance, but are provided by default values expressed in a DTD or schema.

Why specialization in content?

Specialization attributes let processes and tools know what set of rules your markup conforms to. This allows reuse of tools and processes for unfamiliar markup.

Specialization of attributes

When creating a structural specialization, you can limit the contents of existing attributes on your new elements; you can also create new attributes through domain specialization based off of the props attribute (for conditional processing) or the base attribute (for other simple token attributes).

Domain attribute specialization allows DITA document type developers to incorporate new conditional processing attributes that can be used for filtering and flagging, or new attributes with no existing equivalent that can be managed and generalized in the same way as conditional processing attributes.

The new attributes need to be based off of either props or base:

- Attributes specialized from props are identified as conditional processing attributes
- Attributes specialized from base have no existing behavior associated with them
- Values in specialized attributes should be preserved during generalization and respecialization
- While generalized, the attribute values should still be understandable by both general and specialized behaviors, and be treated as equivalent to their specialized form. For example, conditional filtering should work the same way on specialized attributes and on generalized attributes.

Because there is a strong possibility that the existing conditional processing attributes (audience, platform, product, and otherprops) will be moved into a domain based off of props in the future, structural specializations are discouraged from limiting the values in those attributes.

The class attribute

Each element declared in the DITA architecture has a class attribute. This attribute provides a mapping between the element's current name and its more general equivalents. The more specialized the element type, the longer its class attribute value.

For example, the class attribute for the task topic type's step element is:

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

This tells us that the step element is equivalent to the li element in a generic topic. It also tells us that step is equivalent to a step in a task topic, which we already knew, but it's worth noting this in the attribute because it enables round-trip migration between upper level and lower level types without loss of information. For example, if a user runs a "generalize" transform that maps all elements to their first class value, but preserves their content and attribute values, then the user can follow it up with a "specialize" transform that maps all elements to their last class value (preserving content and attribute values), and provide a full round trip for all content between the two document types, using nothing but two generic transforms and the information in the class attribute.

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

The sequence of the values is important because it tells processors which value is the most general and which is most specific. This is especially important for "specializing" transforms, where you can apply a general rule that says: if the element doesn't have a mapping to the target topic type, simply use the last value of the class attribute (and assume that the specialized topic type is reusing some general element declarations, which only have mappings for the level at which they were declared).

The class attribute value is provided as a default value that can be overridden and is not fixed. The ability to change the class attribute becomes important for mapping persistence through migration: when you migrate content to a more abstract element, you can preserve its more specialized history in the class attribute, allowing round-tripping of content between more specialized and more general representations, so it can be processed by different applications at whichever level they understand or support.

Class attribute syntax

The class attribute has a particular syntax that must be followed for it to be processed correctly.

Every element must have a class attribute. The class attribute starts with a "-" if it is declared in a structural module, or a "+" if it is declared in a domain module. After the starting token are one or more blank-delimited values, ending with a blank. Each value has two parts: the first part identifies a specialization module, for example a topic type or domain name, and the second part (after a /) identifies an element type. Structural names are taken from the root element for the topic type or map type. Domain names are defined in the domain module.

Typically, the class attribute value should be declared as a default attribute value in the DTD or schema rather than directly in the document instance. The class attribute should not be modified by the author.

```
<appstep class="- topic/li task/step bctask/appstep ">A specialized step</appstep>
```

Figure 4. Example structural type element with class attribute

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

Figure 5. Example domain element with class attribute

When the class attribute is declared in the DTD or schema, 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 the generalization process to overwrite the default values in a general document type with specialized values taken from the document being generalized.

When a specialized type declares new elements, it must provide a class attribute for the new element. The class attribute must include a mapping for every structural type or domain in the specialized type's ancestry, even those in which no element renaming occurred. The mapping should start with the value for the base type (for example topic or map), and finish with the current element type.

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

Figure 6. Example attribute with intermediate value

Intermediate values are necessary so that generalizing and specializing transforms can map values simply and accurately. For example, if task/kwd was missing as a value, and a user decided to generalize this guitask up to a task topic, then the transform would have to guess whether to map to kwd (appropriate if task is more general than guitask, which it is) or leave as windowname (appropriate if task were more specialized, which it isn't). By always providing mappings for more general values, we 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 specializing 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.

While this example is trivial, more complicated hierarchies (say, five levels deep, with renaming occurring at two and four only) make explicit intermediate values essential.

A specialized type does not need to change the class attribute for elements that it does not specialize, but simply reuses by reference from more generic levels. For example, since task, bctask, and guitask use the p element without specializing it, they don't need to declare mappings for it.

A specialized type only declares class attributes for the elements that it uniquely declares. It does not need to declare class attributes for elements that it reuses or inherits.

The domains attribute

The domains attribute lists the names of the domains in use by the current document type, and the ancestry for each domain. The domains attribute is declared on the root element for each topic type.

Each domain in use contributes a string in parentheses that gives the names of each ancestor domain plus the name of the contributing domain. Within each set of parentheses, the domain and its ancestry should be listed starting with the most distant ancestor (the root type off of which the domain hierarchy is based) and finishing with the name of the domain in use.

Example: task with three domains

```
<task id="mytask" class="- topic/topic task/task "
  domains="(topic ui-d) (topic sw-d) (topic pr-d cpp-d)">
...
</task>
```


In this example, the task allows the use of tags for describing user interfaces (ui-d), software (sw-d), and also C++ programming (cpp-d).

Specialization validity

When you specialize one element from another, or a new attribute from props or base, the new element or attribute must obey certain rules in order to be a valid specialization.

- A new element must have a content model that is equivalent to or more restrictive than its parent.
- A new element must have attributes that are equivalent to or a subset of the attributes of its parent.
- A new element's attributes must have values or value ranges that are equivalent to or a subset of the parent's attributes' values or value ranges.
- A new element must have a properly formed class attribute.
- A new attribute must be specialized from props or base, following the rules for attribute domain specialization.
- A new attribute's values must conform to the rules for conditional processing values, that is, alphanumeric space-delimited values, except when in generalized form, when the values should conform to the rules for attribute generalization.

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.

The generalization can either be for the purpose of migration (for example, when retiring an unsuccessful specialization) or for temporary round-tripping (for example, when moving content through a process that is not specialization aware and has only been enabled for instances of the base structural type). 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 general DTD or schema will be used. When generalizing for round-tripping, the class attribute and domains attribute should retain the original specialized values in the generalized instance document.

Any DITA document can contain a mix of markup from at least one structural type and zero or more domains. The structural types and domains allowed in a particular document type are defined by the document type shell.

When generalizing the document, the generalizer may choose to leave a structural type or domain as-is, or may choose to generalize that type or domain to any of its ancestors.

The generalizer can supply the source and target for each generalization: for example, generalize from reference to topic. The generalizer can specify multiple targets in one pass: for example, generalize from reference to topic and from ui-d to topic. When the source and target are not supplied, generalization is assumed to be from all structural types to the base (topic or map), and no generalization for domains.

The generalizer can also supply the target document type. When the target document type is not supplied, the generalized document will not contain a DTD or schema reference. At some time in the future it may be possible to automatically generate a document type shell and target document type based on the class and domains attributes in the generalized document.

The generalization process should be able to handle cases where it is given just sources for generalization (in which case the designated source types are generalized to topic or map), just targets for generalization (in which case all descendants of the target are generalized to that target), or both (in which case only the specified descendants of the target are generalized to that target).

For each structural type instance, the generalization process 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 an APIReference topic 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:

Target and source	Source unspecified	Source specified
Target unspecified	Generalize this structural type to its base ancestor	Check whether the root element of the topic type matches a specified source; generalize to its base ancestor if it does, otherwise ignore the structural type instance unless it has domains to generalize.
Target specified	Check whether the class attribute contains the target; generalize to the target if it does, otherwise skip the structural type instance unless it has domains to generalize.	If the root element matches a specified source but its class attribute does not contain the target, emit an error message. If the root element matches a specified source and its class attribute does contain the target, generalize to the target. 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:

Target and source	Source unspecified	Source specified
Target 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 specified	Check whether the domains attribute contains the target; generalize to the target if it does, otherwise skip the structural type instance unless it is itself a candidate for generalization.	If the domains attribute matches a specified source but the domain value string does not contain the target, emit an error message. If the domains attribute matches a specified source and the domain value string does contain the target, generalize to the target. Otherwise ignore the structural type instance unless it is itself a candidate for generalization.

For each element in a candidate structural type instance:

Target and source	Source unspecified	Source specified
Target 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 and source	Source unspecified	Source specified
Target specified	Check whether the class attribute contains the target; rename the element to the value associated with the target if it does contain the target, otherwise ignore the element.	If the last value in the class attribute matches a specified source but the previous values do not include the target, emit an error message. If the last value in the class attribute matches a specified source and the previous values do include the target, rename the element to the value associated with the target. Otherwise ignore the element.

When renaming elements during round-trip generalization, the generalization process 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.

Attribute generalization

Attributes specialized from props or base can be generalized using a special attribute generalization syntax. Specialization-aware processes should be able to recognize and process both the specialized and generalized forms of an attribute as being equivalent in their values.

When generalized, specialized attribute values are added to the end of their target ancestor attribute, within parentheses preceded by the name of the specialized attribute. For each attribute that has been generalized, a separate label and parentheses-enclosed set of values is added to the ancestor attribute.

For example, given that "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"

Generalization and respecialization can use the domains attribute to determine the ancestry of role, and therefore the validity of person as an intermediate target for generalization.

Generalized attributes are typically not expected to be authored or edited directly, but to preserve the values of the specialized attributes while the document is temporarily in a generalized form.

If a single element contains values for an attribute in more than one syntax, that is an error. For example `<p person="role(programmer)" role="admin">...</p>` both provide values for the role attribute, but one in a generalized syntax and one in a specialized syntax. This is an error condition, since it means the document has been only partially generalized, or has been generalized and then edited using a specialized document type.

Foreign generalization

During generalization, DITA elements are modified based on the class attribute to use the name of an ancestor element. This cannot be done with the contents of a specialized `<foreign>` element, because the contents are not DITA elements and do not have class attributes.

Instead, only the specialized foreign element itself should be generalized with normal rules. The contents of the foreign element should be exported to a separate file, and replaced in-line with an `<object>` element. The object element should point to the generated file using its data attribute. The type attribute on the object element should be set to "DITA-foreign".

The object element is already allowed within the <foreign> element. If it is present during generalization, it should not be exported to the separate file. The original object element should not be modified unless required by the ordinary rules of generalization. That object element is used for alternate content in publishing systems that cannot display the foreign content.

A foreign content element may contain several main elements, apart from the alternate content. To accommodate this, the exported content should have a root <foreign> element.

The name of the exported file should start with "dita-generalized-" in order to help users recognize the origin of the generated files. It is recommended that the file name also contain the topic ID, specialization type, and element ID or generated identifier. For example, the first mathml object in topic "topicid" might be named "dita-generalized-topicid-mathml1.xml".

Example: Simple object generalization

For example, a DITA document may contain a specialization of <foreign> for MathML. It could look like this:

```
<mathml class="+ topic/foreign mathml/mathml ">
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    <mi>x</mi><mo>+</mo><mn>3</mn>
  </math>
  <object><desc>X plus three</desc></object>
</mathcontent>
```

The mathml container, which is a normal DITA element, should be generalized using normal rules. The <math> element, which is not a DITA element, will be exported to another file. The <object> element will remain:

```
<foreign class="+ topic/foreign mathml/mathml ">
  <object data="dita-generalized-topicid_mathml1.xml" type="DITA-foreign"/>
  <object><desc>X plus three</desc></object>
</foreign>
```

Contents of dita-generalized-topicid_mathml1.xml:

```
<foreign class="+ topic/foreign mathml/mathml ">
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    <mi>x</mi><mo>+</mo><mn>3</mn>
  </math>
</foreign>
```

Example: Multiple object generalization

An object might also contain multiple object elements:

```
<mathml class="+ topic/foreign mathml/mathml ">
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    <mi>x</mi><mo>+</mo><mn>3</mn>
  </math>
  <object><desc>X plus three</desc></object>
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    <mi>y</mi><mo>-</mo><mn>2</mn>
  </math>
</mathcontent>
```

The mathml container, which is a normal DITA element, should be generalized using normal rules. A file should be 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:

```
<foreign class="+ topic/foreign mathml/mathml ">
  <object data="dita-generalized-topicid_mathml1.xml" type="DITA-foreign"/>
  <object><desc>X plus three</desc></object>
  <object data="dita-generalized-topicid_mathml2.xml" type="DITA-foreign"/>
</foreign>
```

```
Contents of dita-generalized-topicid_mathml1.xml:
<foreign class="+ topic/foreign mathml/mathml ">
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    <mi>x</mi><mo>+</mo><mn>3</mn>
  </math>
</foreign>
```

```
Contents of dita-generalized-topicid_mathml2.xml:
<foreign class="+ topic/foreign mathml/mathml ">
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    <mi>y</mi><mo>-</mo><mn>2</mn>
  </math>
</foreign>
```

Specialization in design

Specialization in design enables reuse of design elements, just as specialization in content allows reuse of processing rules. These rules involve the creation and management of markup modules as separate reusable units.

Why specialization in design?

Following the rules for specialization design enables reuse of design elements, just as following the rules for specialized content enables reuse of content

By using standard schemes for developing design modules, a specializer enables:

- Reuse of their design modules by others, allowing shared development of specific parts of a document type
- Faster integration of their designs with other specializations, allowing quicker deployment of new design elements and quicker adoption of new markup standards
- Better management of differences between authoring groups in the same organization: each group can create specific document types that integrate just the modules they require.

Modularization and integration of design

Specialization hierarchies are implemented as sets of module files that declare the markup and entities that are unique to each specialization. The modules must be integrated into a document type before they can be used.

The separation of markup into modules, as with the XHTML modularization initiative, (<http://www.w3.org/TR/xhtml-modularization/>), allows easy reuse of specific parts of the specialization hierarchy, as well as allowing easy extension of the hierarchy (since new modules can be added without affecting existing document types). This makes it easy to assemble design elements from different sources into a single integrated document type.

Integration

Each domain specialization or structural specialization has its own design module. These modules can be combined to create many different document types. The process of creating a new document type from a specific combination of modules is called integration.

Integration is accomplished using a document type shell, which defines the modules to be integrated and how they will be integrated. Integration defines both what topic types and domains will be allowed in the document type, and how the topic types will be allowed to nest.

The module for a specific type should contain only the declarations for elements and attributes that are unique to that type, and should not embed any other modules. The shell should contain no markup declarations, and should directly reference all the modules it requires. Nesting shells or nesting modules

(having shells that embed other shells, or modules that embed other modules) is discouraged since it adds complexity and may break some tools. Sharing between document types should be accomplished through shared modules, not through direct reference to any other document type. Dependencies between modules should be satisfied by the integrating shell, not through the module itself.

Modularization in DTDs

To support extensibility and pluggability, DITA requires that a DTD implementation of structural and domain specialization modules conform to well-defined design patterns.

This section describes those design patterns. These design patterns realize the specialization architecture with the capabilities and within the limitations of the DTD grammar.

Structural specialization pattern

Each structural type must be defined in a separate DTD module with a name consisting of the topic element name and the mod extension. To see an example, look at the `concepts.mod` module for the concept topic type.

The structural type module must conform to the following design pattern.

Default element entities

Each element defined in the module must have a corresponding entity whose default value is the name of the element. The following example comes from the definition for the concept topic.

```
<!ENTITY % conbody "conbody">
```

The document type shell can predefine an element entity to add domain specialized elements into every context in which the base element occurs.

Default included domains entity

The module must define the `included-domains` entity with a default empty that is empty as in the following example:

```
<!ENTITY included-domains "">
```

The document type shell can predefine the `included-domains` entity to list domains added to the document type.

Default nested topics entity

Topic type modules must define an `info-types` entity that is named with a prefix of the topic element name and a suffix of `-info-types`. This entity can default to a list of element entities if the topic has default subordinate topics. If the topic doesn't have default subordinate topics, the entity can default to the value of the `info-types` entity as in the following example:

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

The document type shell can then control how topics are allowed to nest by redefining the *topic*`-info-types` entity for each topic type, or quickly create common nesting rules by redefining the main `info-types` entity.

Structural type's root element content model

As with all specializations, the root element of a structural specialization must have a content model that restricts or conserves the content model of the element it specializes. In addition, for topic types, the last position in the content model must be the nested topics entity as in the following example:

```
<!ELEMENT concept      ((%title;), (%titlealts;)?, (%shortdesc;)?,
                        (%prolog;)?, (%conbody;), (%related-links;)?,
                        (%concept-info-types;)* )>
```

Attributes

As with all specializations, the root element's attributes must restrict or conserve the attributes of

the element it specializes. In particular, the topic must set the DITAArchVersion attribute to the DITAArchVersion entity and the domains attribute to the included-domains entity.

```
<!ATTLIST concept          id ID #REQUIRED
                        ...
                        DITAArchVersion CDATA "&DITAArchVersion;"
                        domains CDATA "&included-domains;"
>
```

These attributes give processes a reliable way to check the architecture version and look up the list of domains available in the document type.

Element and attribute definitions

The module defines every specialized element used as substructure within the topic. The specialized elements must follow the rules of the architecture in defining content models and attributes. Content models must use element entities instead of literal element names.

In particular, the module defines a class attribute for every specialized element. The class attribute must include the value of the class attribute of the base element and append the element name qualified by the topic element name with at least one leading and trailing space. The class attribute for an element introduces by a structural specialization must start with a minus sign.

Domain specialization pattern

Each domain specialization must have two files:

- A DTD entity declaration file with a name consisting of the domain name and the ent extension.
- A DTD definition module with a name consisting of the domain name and the mod extension.

To see an example, look at the highlightDomain.ent and highlightDomain.mod files.

Domain entity declaration file

The domain entity declaration file must conform to the following design pattern:

Element extension entity

The declaration file must define an entity for each element extended by the domain. The contents of the entity must be the list of specialized elements for the extended element. The name of the entity has a prefix of the abbreviation for the domain and an extension of the name of the extended element. In the following example, the highlight domain (abbreviated as hi-d) extends the ph element.

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

Domain declaration entity

The declaration file must define one entity for the document type shell to register the domain. The name of the entity has a prefix of the abbreviation for the domain and an att extension. The value of the entity must list the dependencies of the domain module in order of dependency from left to right within enclosing parentheses, starting with the topic module and listing domain dependencies using their abbreviations (including the defining domain as the last item in the list). The following example declares the dependency of the highlight domain on the base topic module.

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

Domain definition module

The domain definition module conforms to the following design pattern:

Default element entities

As in a topic module, the domain definition module must declare a default entity for each element defined by the domain so that other domains can extend the elements.

```
<!ENTITY % b "b">
```

Element and attribute definitions

As in a topic module, the domain definition module must define each specialized element and its attributes. As with any specialization, the domain element must restrict the base element. The class attribute of the domain element must start with a plus sign but, otherwise, follows the same rules as the class attribute for an element introduced by a topic specialization.

Attribute domain specialization pattern

The attribute domain pattern is a special case of the domain specialization pattern, which allows the creation of new attributes specialized from the props or base attribute.

Create one module entity file per attribute, for example newAttDomain.ent. Each module should contain the following:

Attribute extension entity

The entity which holds the actual declaration of the attribute in entity form. This entity can then be used in document type shells to add the new attribute. For example:

```
<!ENTITY % newAtt-d-attribute "new CDATA #IMPLIED">
```

Domain declaration entity

The entity which holds the name of the attribute domain in entity form. This entity can then be used in document type shells to signal the availability of the attribute to DITA-aware processes. It uses the same syntax as a regular domain declaration entity but with the addition of a leading "a" to signal that it is in attribute domain. For example:

```
<!ENTITY newAtt-d-att "a(props new)" >
```

Document type shell pattern

The document type shell must conform to the following design pattern. To see an example, look at the concepts.dtd module for the concept document type.

Domain entity inclusions

The document type shell starts by including the domain entity declaration files. The entity for the domain declaration consists of the domain name prefix with the dec suffix, as in the following examples:

```
<!ENTITY % hi-d-dec PUBLIC
    "-//OASIS//ENTITIES DITA Highlight Domain//EN" "highlightDomain.ent">
%hi-d-dec;
<!ENTITY % newAtt-d-dec PUBLIC
    "-//My Company//ENTITIES new Attribute Domain//EN" "newAttDomain.ent">
%newAtt-d-dec;
```

Element extension redefinitions

For each element extended by one or more domains, the document type shell redefines the entity for the element to a list of alternatives including the literal name of the element and the element extension entity from each domain that is providing specializations.

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

For each attribute extended by one or more domains, the document type shell redefines the entity for the attribute to a list of alternatives including the literal name of the attribute and the attribute extension entity from each domain that is providing specializations. Attributes can only be specialized from props or base in DITA 1.1.

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

Topic nesting redefinitions

For each topic type, the document type shell can control nesting of subtopics by redefining the

nested topics entity to the literal element name for any of the topics included in the document type. The document type shell can also simply define the info-types entity to set the default for most topic types. Here is an example:

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

Domain declaration redefinition

The document type shell redefines the included-domains entity to list the domains included in the document type as in the following example:

```
<!ENTITY included-domains
    "&ui-d-att; &hi-d-att; &pr-d-att; &sw-d-att; &ut-d-att; &newAtt-d-att">
```

Structural definition inclusions

The document type shell includes the definitions for the structural type modules used in the document type. The entity for the structural definition consists of the structural type's name with the type suffix, as in the following example:

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

Domain definition inclusions

The document type shell includes the domain definitions for the domains used in the document type. The entity for the domain definition consists of the domain name prefix with the def suffix, as in the following example:

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

There are no domain definition files for attribute domains.

Modularization in schemas

To support extensibility and pluggability, DITA requires that an XML schema implementation of structural and domain specialization modules conform to well-defined design patterns.

This section describes those design patterns. These design patterns realize the specialization architecture with the capabilities and within the limitations of the XML schema grammar.

Structural specialization pattern

For each structural type, the document type shell document collects the schema documents, parent structural type modules, domain type modules, and content models needed to implement new topic type specializations. Each new structural type requires three files. To see an example, look at the concept.xsd document type shell document for the concept topic type.

1. Each structural type must define a separate module schema document with a name consisting of the root structural element name and Mod.xsd
2. Each structural type must define a separate model group definition schema document with a name consisting of the root structural element name and Grp.xsd

The module schema document must define an info-type model group that is named with a prefix of the topic element name and a suffix of -info-types. Here is an example of a info-types model group that is defined in conceptMod.xsd:

```
<xs:group name="concept-info-types">
  <xs:choice>
    <xs:group ref="concept" minOccurs="0"/>
    <xs:group ref="info-types" minOccurs="0"/>
  </xs:choice>
</xs:group>
```


The module schema document defines every specialized element used as substructure within the structural type. The specialized elements must follow the rules of the architecture in defining content models and attributes. The naming convention for content models must use the root structural element name and.class

As with all specializations, the root element of a structural specialization must have a content model that restricts or conserves the content model of the element it specializes. The root element must also reference the DITAArchVersion attribute and domains attribute. For the schemas, the domains attribute value is set in the document type shell. See Document type shell pattern for more information on how to set the values for the domains attribute. In addition the last position in the content model must be the nested topics entity as in the following example:

```
<xs:complexType name="concept.class">
  <xs:sequence>
    <xs:group ref="title"/>
    <xs:group ref="titlealts" minOccurs="0"/>
    <xs:choice minOccurs="0">
      <xs:group ref="shortdesc" />
      <xs:group ref="abstract" />
    </xs:choice>
    <xs:group ref="prolog" minOccurs="0"/>
    <xs:group ref="conbody" minOccurs="0"/>
    <xs:group ref="related-links" minOccurs="0"/>
    <xs:group ref="concept-info-types" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  . . .
</xs:complexType>
```

These attributes give processes a reliable way to check the architecture version and look up the list of domains available in the document type.

```
<xs:attribute name="id" type="xs:ID" use="required"/>
<xs:attribute ref="ditaarch:DITAArchVersion" />
```

The new file *topicTypeGrp.xsd* is needed to mimic substitutionGroups in XML Schema without using the inheritance model in W3C XML Schema 1.0 specification. The process is very similar to the DITA DTD design pattern. For a structural type the name of the schema document consists of the root structural element name and Grp.xsd extension. To see an example of a model group schema document, look at the file *conceptGrp.xsd* :

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

The model group schema document defines model groups for each new specialized element in a structural type. Each structural type and domain must have a model group schema document . The model group schema document is an essential part of the specialization.

Binding the class attribute to an element:

The class attribute must include the value of the class attribute of the base element and append the element name qualified by the root structural element name or domain name with at least one leading and trailing space. The class attribute starts with a ∇-∇ if it is declared in a structural module, or a ∇+∇ if it is declared in a domain module.

This attribute provides a mapping between the element's current name and its more general equivalents. It must be bound the elements declaration, and not in the complexType referenced by the element. To see an example, look at the reference.xsd schema document.


```

<xs:element name="reference">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="reference.class">
        <xs:attribute ref="class" default="- topic/topic reference/reference" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

```

Domain specialization pattern

A domain type schema document with a name consisting of the domain name and the Domain.xsd extension.

As in a structural module, the domain module must define each specialized element, its attributes and its model groups. As with any specialization, the domain element must restrict the base element. The class attribute of the domain element must start with a plus sign but, otherwise, follows the same rules as the class attribute for an element introduced by a topic specialization.

For each element extended by one or more domains, the domain type schema document defines a model group for the base element to a list of alternatives including the literal name of the element and the element extension entity from each domain that is providing specializations.

The schema document must define an a model group for each element extended by the domain. The contents of the model group must be the list of specialized elements for the extended element. The name of the model group has a prefix of the abbreviation for the domain and an extension of the name of the extended element. In the following example, the user interface domain (abbreviated as ui-d) extends the ph element.

```

<xs:group name="ui-d-ph">
  <xs:choice>
    <xs:element ref="uicontrol" />
    <xs:element ref="menucascade" />
  </xs:choice>
</xs:group>

```

Document type shell pattern

For each document shell type, the following named model group *info-types* must be defined. This model group can define a list of default subordinate topics. If the topic type does not have default subordinate topics, then the default value for the *info-types* model group must be defined as an empty group.

```

<xs:group name="info-types">
  <xs:sequence/>
</xs:group>

```

The default values for the domains attributes in the base root structural element and the specialized root structural elements must be defined using the schema <redefine> mechanism to populate the domains attribute. It identifies the domains used in the structural type. This attribute give processes a reliable way to look up the list of domains available in the document type. The list the domains is included in the document type as in the following example:

```

<xs:redefine schemaLocation="topicMod.xsd" >
  <xs:complexType name="topic.class">
    <xs:complexContent>
      <xs:extension base="topic.class">
        <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)"/>

```

```

        </xs:extension>
    </xs:complexContent>
</xs:complexType>
</xs:redefine>

```

For each element extended by one or more domains, the document type shell redefines 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 <redefine> mechanism to manage the number of domains used by the document type shell. The model group requires a reference to itself to extend the base model group. To see an example, look at the topic.xsd schema document.

```

<xs:group name="pre">
  <xs:choice>
    <xs:group ref="pre" />
    <xs:group ref="pr-d-pre" />
    <xs:group ref="ui-d-pre" />
    <xs:group ref="sw-d-pre" />
  </xs:choice>
</xs:group>

```

To add domains to a new structural type you can copy the contents of the parent structural type domains schema document into the document type shell. Add or remove the model group from the new domain to the appropriate named group.

```

<xs:group name="pre">
  <xs:choice>
    <xs:group ref="pre"/>
    <xs:group ref="pr-d-pre" />
    <xs:group ref="domainName-d-element"/>
  </xs:choice>
</xs:group>

```

For each attribute extended by one or more domains, the document type shell redefines 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. Attributes can only be specialized from props or base in DITA 1.1. To integrate a new attribute domain in the document type shell use the schema <redefine> mechanism to manage the number of attribute domains used by the document type shell.

```

<xs:attributeGroup name="props-attribute-extensions">
  <xs:attributeGroup ref="props-attribute-extensions"/>
  <xs:attributeGroup ref="newAtt-d-attribute"/>
  <xs:attributeGroup ref="othernewAtt-d-attribute"/>
</xs:attributeGroup>

<xs:attributeGroup name="base-attribute-extensions">
  <xs:attributeGroup ref="base-attribute-extensions"/>
  <xs:attributeGroup ref="newfrombaseAtt-d-attribute"/>
  <xs:attributeGroup ref="othernewfrombaseAtt-d-attribute"/>
</xs:attributeGroup>

```

Attribute domain specialization pattern

The attribute domain pattern is a special case of the domain specialization pattern, which allows the creation of new attributes specialized from the props or base attribute.

Create one module entity file per attribute, for example newAttDomain.xsd. Each module should contain the following:

The entity which holds the actual declaration of the attribute in entity form. This entity can then be used in document type shells to add the new attribute. For example:

```
<xs:attributeGroup name="newAtt-d-attribute">
  <xs:attribute name="new" type="xs:string"/>
</xs:attributeGroup>
```

This attribute domain declaration value can then be used in document type shells to signal the availability of the attribute to DITA-aware processes. It uses the same syntax as a regular domain declaration value but with the addition of a leading "a" to signal that it is in attribute domain. For example:

```
<xs:attribute name="domains" type="xs:string" default="... a(props new)"/>
```

Specialization in processing

Specialized processing is not necessary for every specialized element, only for those elements which do not have appropriate default behavior based on their ancestors.

Whether creating a new transform or extending an existing one, there are several rules that should be followed to ensure the effectiveness of the transform for other specialized types, and also the maintainability and extensibility of the transform to accommodate new requirements.

Using the class attribute

Applying an XSLT template based on class attribute values allows a transform to be applied to whole branches of element types, instead of just a single element type.

Wherever you would check for element name (any XPath statement that contains an element name value), you need to change this to instead check the contents of the element's class attribute. Even if the element is unknown to the processor, the class attribute can let the transform know that the element belongs to a class of known elements, and can be safely treated according to the rules for that class.

Be sure to include a leading and trailing blank in your class attribute string check. Otherwise you could get false matches (without the blanks, 'task/step' would match on both 'task/step' and on 'notatask/stepaway').

Make sure that when you create a transform that targets more than one type that you give the more specific rules a higher precedence to avoid conflicts. For example, when you combine the existing processing rules for topics with more specific processing rules for tasks, use a shell file to import both sets of rules and use import precedence to ensure task-specific rules will not conflict with generic rules for topics.

Example: match statement for list items

```
<xsl:template match="li">
```

becomes

```
<xsl:template match="*[contains(@class,' topic/li ')]">
```

This match statement will work on any li element it encounters. It will also work on step and appstep elements, even though it doesn't know what they are specifically, because the class attribute tells the template what they are generally.

Example: match statement for steps

```
<xsl:template match="*[contains(@class,' task/step ')]">
```

This match statement won't work on generic li elements, but it will work on both step elements and appstep elements; even though it doesn't know what an appstep is, it knows to treat it like a step.

Processing specialized attributes

There are two kinds of specialized attributes: those specialized from props for conditional processing, and those specialized from base for other purposes.

Attributes specialized from props should inherit conditional processing behavior by default, which should also work with their generalized forms. In other words, values in a new attribute specialized from props can be conditionally processed to filter or flag the element in exactly the same way as any other conditional processing attribute (such as platform or audience).

Attributes specialized from base will have no processing by default; any processing provided should be enabled to work with their generalized form as well.

See “Attribute generalization” on page 77 for information on the syntax of generalized attributes.

Processing foreign content

The default behavior for `<foreign>` is to try to display the content. If the processor cannot render the content, it may emit a warning. The default processing behavior for `<unknown>` is to suppress.

This section describes processing behaviour of specialized foreign content based on existing standard vocabularies for non-textual content.

Default processing for `<foreign>`

The enabler of the foreign vocabulary must provide the processing by overriding the base processing for `<foreign>`.

- If `<foreign>` contains more than one alternative content element, they will all be processed. In the case of `<desc>` they will be concatenated in a similar way to `<section>` but no title (similar to `<div>` in HTML).
- Where appropriate, the specializer may specialize a nested `<desc>` element to provide alternate content that is valid in the contexts for the `<foreign>` specialization. If an instance of the `<foreign>` element doesn't contain a `<desc>`, `<object>`, or `<image>` element, the base processing may emit a warning about the absence of processable content.
- If the instance of the `<foreign>` element doesn't contain a `<desc>`, `<object>`, or `<image>` element, the base processing may emit a warning about the absence of processable content.
- The base processing for `<object>` can 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 an `<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 will ignore one of them.

Default processing for `<unknown>`

The base processing for `<unknown>` is to suppress unless otherwise instructed.

Modularization and integration of processing

Processing should be divided into modules based on the structural types or domains they support, and can be integrated together into transforms or stylesheets in the same way that structural type and domain modules can be integrated into document types.

Customization

When you just need a difference in output, you can use DITA customization to override the default output without affecting portability or interchange, and without involving specialization.

For example, if your readers are mostly experienced users, you could concentrate on creating many summary tables, and maximizing retrievability; or if you needed to create a brand presence, you could customize the transforms to apply appropriate fonts and indent style, and include some standard graphics and copyright links.

Use customization when you need new output, with no change to the underlying semantics (you aren't saying anything new or meaningful about the content, only its display).

Modularization in CSS

Stylesheet support in CSS for DITA specializations can be applied using the same principles as for the DTDs or Schemas, resulting in stylesheets that are easy to maintain and that will support any subsequent specialization with a minimum of effort.

Specification of module definition

A specialization-aware property for CSS has this form of selector:

```
*[class~="topic\section"] {  
    margin-top: 12pt;  
    display: block;  
}
```

The CSS selector that associates the style to the element does not use a literal match to the element name. Instead, based on an element having the defaulted value `class="- topic/section reference/refsyn "` (for example) this rule will trigger on the "topic/section" value (or "word") and perform the associated styling or transform, regardless of what the actual element name is.

Note that the attribute string must contain an escape character for the "/" character which is otherwise not valid in a CSS selector.

The selector pattern in this example effectively reads, in CSS terminology, "Selects any element with a class attribute that contains the word topic\section."

Not all CSS systems can match based on values that are not physically present in the instance document. Since the class attribute values in DITA are typically provided by default declarations in the DTD or schema, not all CSS systems can match directly on DITA source.

When direct specialization-aware matches are not possible, alternatives include normalization (preprocessing the DITA source to push values from the DTD or schema directly into the instance) or the use of element-name-based rules.

Element-name-based rules will not be specialization-aware. Your calling-stylesheet will have to import each additional stylesheet required by the scope of specialized topics and vocabularies, each explicitly defined using element-name selectors. In this scheme, unsupported new elements will have no rendering properties associated, whereas in the specialization-aware systems such elements can fall back to a rule that triggers off a previously-supported value in the class attribute string.

Assembly rules for CSS

CSS supports specialization similarly to XSLT. This document describes a best practice for naming and populating CSS stylesheets that follow the specialization design pattern for DITA DTDs and Schemas. Although this practice is not required in order to implement CSS support for DITA, following the practice will make subsequent specializations off the pattern to be done with minimal work, and the files should be correspondingly easier to maintain.

To support a newly-specialized DITA DTD or Schema that has been specialization-enabled with unique class attribute values, create a module that will contain ONLY the rules required for the uniquely new

elements in the specialization. This is similar to the mod files that declare the unique elements in the specialization. The name of this module should be the same as the root name for the specialization module. In the case of DITA's reference DTD, the element declarations are in reference.mod and the corresponding delta rules for CSS are in reference.css.

Next, create an "override" CSS stylesheet that starts off with the @import instruction, naming the CSS file used by this specialization's parent DTD. This import picks up support for all elements that are common with the parent DTD. Then add another @import instruction in sequence, naming the CSS delta module that you created previously. Then copy in the CSS rules for any previously defined support that need to be associated to the new element names, and rename the selectors as needed to the new specialized values for each new element. These added CSS rules are deltas for the new stylesheet, much as specialized DTDs build on previous DTDs by adding delta element definitions. This technique approximates the "fall-through" support for what would normally happen if the class attribute actually could map to the root class.

Finally, if necessary, modify the behaviors of any of these new, delta CSS rules. Because this process reuses a great deal of previous behaviors, the time spent supporting the delta changes is minimal.

To use a specialization-enabled CSS stylesheet with a specialized DITA topic, simply associate it to the topic using either the W3C defined stylesheet link processing instruction or by following configuration rules for your editor or browser.

Modularization in XSLT

Stylesheet support in XSLT for DITA specializations can be applied using the same principles as for the DTDs or Schemas, resulting in stylesheets that are easy to maintain and that will support any subsequent specialization with a minimum of effort.

Specification of module definition

A specialization-aware template for XSLT has this form of match pattern:

```
<xsl:template match="*[contains(@class,' topic/section ')]">
  <div>
    <xsl:apply-templates/>
  </div>
</xsl:template>
```

The XSLT match statement that associates the style to the element does not use a literal match to the element name. Instead, based on an element having the defaulted value `class="- topic/section reference/refsyn "` (for example) this rule will trigger on the " topic/section " value (note the required space delimiters in the match string) and perform the associated template actions, regardless of what the actual element name is.

The XPath pattern in this example effectively reads, "Selects any element whose class attribute contains the space-delimited substring "topic/section"."

Assembly rules for XSLT

XSLT pattern matching is the basis for DITA's specialization-aware processing. As such, the base XSLT stylesheet for a DITA topic should minimally support any specialization, no matter how far removed in generations from the archetype topic.

To support a newly-specialized DITA DTD or Schema that has been specialization-enabled with unique class attribute values, create a module that will contain ONLY the templates required for the uniquely new elements in the specialization. This is similar to the mod files that declare the unique elements in the specialization. The name of this module should be the same as the root name for the specialization

module. In the case of DITA's reference DTD, the element declarations are in `reference.mod` and the corresponding delta rules for XSLT are in `reference.xsl`.

Next, create an "override" XSLT stylesheet that starts off with the `xsl:import` instruction, naming the XSLT file used by this specialization's parent DTD. This import picks up support for all elements that are common with the parent DTD. Then add another `xsl:import` instruction in sequence, naming the XSLT delta module that you created previously. Additionally you can add imports for any domain-specific templates that need to be applied with this shell. Then copy in the XSLT templates for any previously defined support that needs to be associated *uniquely* to the new element names, and rename the match pattern strings as needed to the new specialized values for each new element. These added XSLT templates are deltas for the new stylesheet, much as specialized DTDs build on previous DTDs by adding delta element definitions. For XSLT support, you only need to define templates if you need new behavior or if you need to modify the behavior of an ancestor element's processing.

Because this process reuses a great deal of previous behaviors, the time spent supporting the delta changes is minimal.

To use a specialization-enabled XSLT stylesheet with a specialized DITA topic, simply associate it to the topic using either the W3C defined stylesheet link processing instruction or by following configuration rules for your processing tools (usually an XSLT processing utility such as `saxon` or `xsltproc`).