Antenna House DITA Open Toolkit Testcase

DITA to XSL-FO Transformation

Unit test cases

Antenna House, Inc. Japan 12 11 2009
UD2009-001 1
2006 2009 Antenna House, Inc.

Test data title

The test data begins from here!

Preface

This test data contains the excerpts and the samples from DITA 1.1 specifications (http://docs.oasis-open.org/dita/v1.1/OS/langspec/ditaref-type.html). Also contains the excerpts from Wikipedia (http://en.wikipedia.org/) The Wikipedia text content is available under the Creative Commons Attribution-ShareAlike License.

- **Safety Precaution**
  - In order to protect the system controlled by the product and the product itself and ensure safe operation, observe the safety precautions described in this user's manual. We assume no liability for safety if users fail to observe these instructions when operating the product.
  - The following symbols are used in the product and user's manual to indicate that there are precautions for safety:

Symbol	Description
	Indicates that caution is required for operation. This symbol is placed on the product to refer the user to the user's manual in order to protect the operator and the equipment. In the user's manuals you will find precautions to avoid physical injury or death of the operator, including electrical shocks.
	Identifies a protective grounding terminal. Before using the product, ground the terminal.
	Identifies a functional grounding terminal. Before using the product, ground the terminal.*1
	Indicates an AC supply.*2
	Indicates a DC supply.
	Indicates that the main switch is ON.
	Indicates that the main switch is OFF.

*1 A terminal is a the point at which a conductor from an electrical component, device or network comes to an end and provides a point of connection to external circuits.

*2 Power is defined as the rate of flow of energy past a given point. In alternating current circuits, energy storage elements such as inductance and capacitance may result in periodic reversals of the direction of energy flow. The portion of power flow that, averaged over a complete cycle of the AC waveform, results in net transfer of energy in one direction is known as real power. The portion of power flow due to stored energy, which returns to the source in each cycle, is known as reactive power.

*3 Direct current (DC) is the unidirectional flow of electric charge. Direct current is produced by such sources as batteries, thermocouples, solar cells, and commutator-type electric machines of the dynamo type. Direct current may flow in a conductor such as a wire, but can also be through

semiconductors, insulators, or even through a vacuum as in electron or ion beams. The electric charge flows in a constant direction, distinguishing it from alternating current (AC).

■ Addendum

Warning

Indicates a potentially hazardous situation which, if instructions are not followed, could result in death or serious injury.*@

Caution

Indicates a potentially hazardous situation which, if instructions are not followed, may result in minor or moderate injury or damage to property. *1

Important

Indicates points to pay attention to when using the machine, and explanations of likely causes of paper misfeeds, damage to originals, or loss of data. Be sure to read these warnings.*2

Note

Indicates supplementary explanations of the machine's functions, and instructions on resolving user errors. *3

*@ In the People's Republic of China, warning signs appear with a black border and a yellow background. In Sweden, Serbia, Bosnia and Herzegovina, Croatia, Greece, Finland, Iceland, the Republic of Macedonia and Poland, they have a red border with an amber background. This is due to the weather, as it is easier to see a red/amber sign in the snowy weather than a red/white sign. The polar bear warning sign in Svalbard recently changed from displaying a black bear on white background to a white bear on black background (both signs are triangular with a red border). Some countries that normally use a white background have adopted an orange or amber background for road work or construction signs.

*1 The solid yellow flag, or caution flag, universally requires drivers to slow down due to a hazard on the track. However, the procedures for displaying the yellow flag vary for different racing styles and sanctioning bodies.

*2 The quality or condition of being important or worthy of note

*3 The term "note" can be used in both generic and specific senses: one might say either "the piece Happy Birthday to You begins with two notes having the same pitch," or "the piece begins with two repetitions of the same note." In the former case, one uses "note" to refer to a specific musical event; in the latter, one uses the term to refer to a class of events sharing the same pitch.

Contents

Chapter 1. Title testing **1**

1.1 *Weird* **title** testing ® (1) 1

1.2 Weird title testing ™ (2) 2

Chapter 2. Testing examples **3**

2.1 Abstract testing 3

2.2 Abstract testing 4

2.3 Xref testing 5

2.4 Note testing 6

2.5 Bodyelements testing 9

2.6 Miscellaneous elements testing 13

2.7 Specialization elements testing 15

2.8 Typographic elements testing 17

2.9 Programming elements testing 19

2.10 Software elements testing 25

2.11 Utility elements testing 27

2.12 Fig and table testing 28

2.12.1 Fig testing 28

2.12.2 Table testing 31

2.12.3 Properties 40

2.12.4 Foreign testing 41

Chapter 3. Logging in to DB3 Client **43**

Chapter 4. DITA 1.2 element test **47**

4.1 Longdescref testing 47

4.2 Longquoteref testing 49

4.3 Sectiondiv testing 51

Appendices **52**

Appendix 1. XSL Transformations 52

Backmatter testing **57**

Glossary **58**

Index **60**

Figures

Figure 2.9-1 CopyFile 21
Figure 2.9-2 SAA CPI Database Reference 21
Figure 2.9-3 CopyFile 22
Figure 2.9-4 URL syntax 22
Figure 2.9-5 `uname` can take the -a option 22
Figure 2.9-6 `uname` assumes the -s option by default 22
Figure 2.9-7 `uname` synopsis 22
Figure 2.9-8 Hexadecimal literal 22
Figure 2.9-9 Comma-separated list of numbers 22
Figure 2.9-10 Repeating a group *A* exactly 1 to 3 times 23
Figure 2.9-11 Positive decimal integer 23
Figure 2.9-12 Applying adjectives to a noun 23
Figure 2.9-13 How repsep in combination with importance is disambiguated . . . 23
Figure 2.9-14 Refactoring repsep 23
Figure 2.9-15 IPV4 address in dot form 23
Figure 2.9-16 Decimal integer 24
Figure 2.9-17 Decimal integer 24
Figure 2.12-1 Flower 28
Figure 2.12-2 Flower frame='all' 28
Figure 2.12-3 Flower frame='topbot' 28
Figure 2.12-4 Flower width="3cm" 28
Figure 2.12-5 Flower height="5cm" 29
Figure 2.12-6 Flower width="3cm", height="5cm" 29
Figure 2.12-7 Flower scale="200" 29
Figure 2.12-8 Flower placement="break"&align="center" 29
Figure 2.12-9 Flower placement="break"&align="rifht" 29
Figure 2.12-10 Flower placement="break"&align="left" 30
Figure 2.12-11 Flower placement="break"&align="current" 30
Figure 2.12-12 Dragon 30
Figure 2.12-13 Sample complex figure 30
Figure 2.12-14 Sample complex figure (2) 30
Figure 2.12-15 Sample complex figure (3): no frame 31
Figure 2.12-16 Flower 31
Figure 2.12-17 Link to Wikipedia 31
Figure 2.12-18 Book & bird 37
Figure 2.12-19 MathML example (1) 41
Figure 2.12-20 MathML example (2) 41
Figure 2.12-21 SVG example. 42

Tables

Table 2.9-1 parml in table sample 20
Table 2.12-1 Normal Table Sample 32
Table 2.12-2 Normal Table Sample with scale="140" frame="none" 32
Table 2.12-3 Normal Table Sample with rowheader='firstcol' 32
Table 2.12-4 Normal Table with rowsep="0" colsep="0". 33
Table 2.12-5 Normal Table with pgwide="1" 33
Table 2.12-6 dl in table sample 33
Table 2.12-7 sl in table sample 35
Table 2.12-8 ol in table sample 35
Table 2.12-9 ul in table sample 36
Table 2.12-10 fig in table sample 36
Table 2.12-11 Valign sample 37
Table 2.12-12 Cell span sample 37
Table 2.12-13 Table align sample (No cell rule) 37
Table 2.12-14 Table align sample 38
Table 2.12-15 Normal Table Sample (Again) 39

Chapter 1. Title testing

1.1 *Weird* title testing ® (1)

The <title> element contains a heading or label for the main parts of a topic, including the topic as a whole, its sections and examples, and its labelled content, such as figures and tables.

The title content model is as follows.

```
( text data or ph or codeph or synph or filepath or msgph or userin
put or systemoutput or b or u or
i or tt or sup or sub or uicontrol or menucascade or term or q or b
oolean or state or keyword or
option or parmname or apiname or cmdname or msgnum or varname or wi
ntitle or tm or image or
data or data-about or foreign or unknown) (any number)
```

This topic also contains related-link test.

RELATED LINKS

1.2 Weird title testing ™ (2) on page 2

Extensible Stylesheet Language (XSL) Version 1.1 (PDF)

Extensible Stylesheet Language (XSL) Version 1.1 (HTML)

Extensible Stylesheet Language (XSL) Version 1.1 (BOOK)

Sample PDF

Japanese Constitution Preamble

1.2 Weird title testing ™ (2)

The <title> element contains a heading or label for the main parts of a topic, including the topic as a whole, its sections and examples, and its labelled content, such as figures and tables.

The title content model is as follows.

```
( text data or ph or codeph or synph or filepath or msgph or userin
put or systemoutput or b or u or
        i or tt or sup or sub or uicontrol or menucascade or term o
r q or boolean or state or keyword or
        option or parmname or apiname or cmdname or msgnum or varna
me or wintitle or tm or image or
        data or data-about or foreign or unknown) (any number)
```

This topic also contains related-link test.

RELATED LINKS

1.1 *Weird* **title** testing ® (1) on page 1

Extensible Stylesheet Language (XSL) Version 1.1 (PDF)

Extensible Stylesheet Language (XSL) Version 1.1 (HTML)

Extensible Stylesheet Language (XSL) Version 1.1 (BOOK)

Sample PDF

Japanese Constitution Preamble

Chapter 2. Testing examples

[Short desc] This part contains DITA element testing examples.

First paragraph.

Second paragraph.

2.1 Abstract testing

The abstract is being used to provide more complex content. The shortdesc must be directly contained by the abstract.

The abstract can put text around the shortdesc.
There can be more than one shortdesc.

First paragraph of the introduction.

Second paragraph of the introduction.

2.2 Abstract testing

The abstract is being used to provide more complex content. The shortdesc must be directly contained by the abstract.

The abstract can put text around the shortdesc.
There can be more than one shortdesc.

First paragraph of the introduction.

Second paragraph of the introduction.

2.3 Xref testing

Xref is cross-reference element to link a diffrent location within the current topc, or a diffrent topic within the same help system, or to external source such as Web pages, or to a location in another topic.

Fn definition

This is fn test.

*1 This is <fn> body

- **Xref to topic**

  Abstract testing is found in the topic titled 2.1 Abstract testing on page 3.

- **Xref to section**

  Xref to section is found in the section titled ● Xref to section on page 5.

- **Xref to example**

  Xref to example is found in the example titled ● Xref to example on page 5.

- **Xref to refsyn**

  Second Property sample is found in the topic titled ● No @relcolwidth on page 40.

- **Xref to external PDF file**

  Refer to Sample.pdf for details.

- **Xref to table sample**

  Refer to Table 2.12-13 Table align sample (No cell rule) on page 37 for details.

- **Xref to li sample**

  Then you should back to step b. to complete logging into system.

- **Xref to fig sample**

  Then you should refer to Figure 2.12-1 Flower on page 28 for details.

- **Xref to fn sample**

  Xref to fn *3 .

  Xref to fn *1 .

- **Xref to sub sample**

  This sample causes DOTX032E error in topicpull because <b> has no <title>.

  Xref to sub 2 .

2.4 Note testing

A <note> element contains information, differentiated from the main text, which expands on or calls attention to a particular point.

- **Note**

  Note

  Defines the type of a note. For example, if the note is a tip, the word Tip is used to draw the reader's attention to it. Note that this differs from the type attribute on many other DITA elements. See "The type attribute" on page 480 for detailed information on supported values and processing implications.

- **Tip**

  Tip

  Defines the type of a note. For example, if the note is a tip, the word Tip is used to draw the reader's attention to it. Note that this differs from the type attribute on many other DITA elements. See "The type attribute" on page 480 for detailed information on supported values and processing implications.

- **fastpath**

  Fast Path

  Defines the type of a note. For example, if the note is a tip, the word Tip is used to draw the reader's attention to it. Note that this differs from the type attribute on many other DITA elements. See "The type attribute" on page 480 for detailed information on supported values and processing implications.

- **restriction**

  Restriction

  Defines the type of a note. For example, if the note is a tip, the word Tip is used to draw the reader's attention to it. Note that this differs from the type attribute on many other DITA elements. See "The type attribute" on page 480 for detailed information on supported values and processing implications.

- **important**

  Important

  Defines the type of a note. For example, if the note is a tip, the word Tip is used to draw the reader's attention to it. Note that this differs from the type attribute on many other DITA elements. See "The type attribute" on page 480 for detailed information on supported values and processing implications.

- **remember**

  Remember

  Defines the type of a note. For example, if the note is a tip, the word Tip is used to draw the reader's attention to it. Note that this differs from the type attribute on many other DITA elements. See "The type attribute" on page 480 for detailed information on supported values and processing implications.

- **attention**

  Attention

  Defines the type of a note. For example, if the note is a tip, the word Tip is used to draw the reader's attention to it. Note that this differs from the type attribute on many other DITA elements. See "The type attribute" on page 480 for detailed information on supported values and processing implications.

- **caution**

  Caution

  Defines the type of a note. For example, if the note is a tip, the word Tip is used to draw the reader's attention to it. Note that this differs from the type attribute on many other DITA elements. See "The type attribute" on page 480 for detailed information on supported values and processing implications.

- **danger**

  Danger

  Defines the type of a note. For example, if the note is a tip, the word Tip is used to draw the reader's attention to it. Note that this differs from the type attribute on many other DITA elements. See "The type attribute" on page 480 for detailed information on supported values and processing implications.

- **other**

  Other

  Defines the type of a note. For example, if the note is a tip, the word Tip is used to draw the reader's attention to it. Note that this differs from the type attribute on many other DITA elements. See "The type attribute" on page 480 for detailed information on supported values and processing implications.

- **other='Warning'**

  Warning

  Defines the type of a note. For example, if the note is a tip, the word Tip is used to draw the reader's attention to it. Note that this differs from the type attribute on many other DITA elements. See "The type attribute" on page 480 for detailed information on supported values and processing implications.

2.5 Bodyelements testing

The body elements support the most common types of content authoring for topics: paragraphs, lists, phrases, figures, and other common types of exhibits in a document.

- **ph element**

The phrase (<ph>) element is used to organize content for reuse or conditional processing (for example, when part of a paragraph applies to a particular audience). It can be used by specializations of DITA to create semantic markup for content at the phrase level, which then allows (but does not require) specific processing or formatting.

This was not changed. This was updated. This was not.

- **keyword element**

The <keyword> element identifies a keyword or token, such as a single value from an enumerated list, the name of a command or parameter, product name, or a lookup key for a message.

The assert pragma statement allows messages to be passed to the emulator, pre-compiler, etc..

- **sl element**

The simple list (<sl>) element contains a simple list of items of short, phrase-like content, such as in documenting the materials in a kit or package. On output, the list should have no bullets, on the assumption that each item is short enough to fit on one line, and needs no additional differentiation from its neighbors.

Messages from the ags_open module are identical with messages from:

ags_read

ags_write

ags_close

sl/@compact='yes'

Messages from the ags_open module are identical with messages from:

ags_read
ags_write
ags_close

- **dl element**

A definition list (<dl>) is a list of terms and corresponding definitions. The term (<dt>) is usually flush left. The description or definition (<dd>) is usually either indented and on the next line, or on the same line to the right of the term. You can also provide an optional heading for the terms and definitions, using the <dlhead> element, which contains header elements for those columns. The default formatting for the <dlhead> looks like a table with a heading row.

Simple:

Bytes returned

The number of bytes of data returned.

Bytes available

The number of bytes of data available to be returned.

Handle

The returned handle value

Simple & @compact="yes":

Bytes returned

The number of bytes of data returned.

Bytes available

The number of bytes of data available to be returned.

Handle

The returned handle value

- ## foreign element

The <foreign> element is an open extension that allows information architects to incorporate existing standard vocabularies for non-textual content. like MathML and SVG, as inline objects. If <foreign> contains more than one alternative content element, they will all be processed. Specialization of <foreign> should be implemented as a domain, but for those looking for more control over the content can implement foreign vocabulary as an element specialization.

What's EIM? .

- ## pre element

The preformatted element (<pre>) preserves line breaks and spaces entered manually by the author in the content of the element, and also presents the content in a monospaced type font (depending on your output formatting processor). Do not use <pre> when a more semantically specific element is appropriate, such as <codeblock>.

No parameter

```
MEMO: programming team fun day
Remember to bring a kite, softball glove, or other favorite
outdoor accessory to tomorrow's fun day outing at Zilker Park.
Volunteers needed for the dunking booth.
```

frame="topbot" scale="90"

```
MEMO: programming team fun day
Remember to bring a kite, softball glove, or other favorite
```

```
outdoor accessory to tomorrow’s fun day outing at Zilker Park.
Volunteers needed for the dunking booth.
```

frame="all" scale="90"

```
MEMO: programming team fun day
Remember to bring a kite, softball glove, or other favorite
outdoor accessory to tomorrow’s fun day outing at Zilker Park.
Volunteers needed for the dunking booth.
```

- ## lines element

The <lines> element may be used to represent dialogs, lists, text fragments, and so forth. The <lines> element is similar to <pre> in that hard line breaks are preserved, but the font style is not set to monospace, and extra spaces inside the lines are not preserved.

On a trip to the beach, don’t forget:

suntan lotion
sunglasses
a beach towel

frame="all"

On a trip to the beach, don’t forget:

suntan lotion
sunglasses
a beach towel

- ## cite element

The <cite> element is used when you need a bibliographic citation that refers to a book or article. It specifically identifies the title of the resource.

The online article *Specialization in the Darwin Information Typing Architecture* provides a detailed explanation of how to define new topic types.

- ## lq element

The long quote (<lq>) element indicates content quoted from another source. Use the quote element <q> for short, inline quotations, and long quote <lq> for quotations that are too long for inline use, following normal guidelines for quoting other sources. You can store a URL to the source of the quotation in the href attribute; the href value may point to a DITA topic.

This is the first line of the address that Abraham Lincoln delivered on November 19, 1863 for the dedication of the cemetery at Gettysburg, Pennsylvania.

> Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal. Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal.
>
> *Gettysburg, Pennsylvania November 19, 1863*

Another lq

> This is the test of properties table with no @relcolwidth.
>
> *no @relcolwidth*

- **q element**

  A quotation element (<q>) indicates content quoted from another source. This element is used for short quotes which are displayed inline. Use the long quote element <lq>) for quotations that should be set off from the surrounding text.

  George said, *"Disengage the power supply before servicing the unit."*

2.6 Miscellaneous elements testing

Most DITA elements represent discourse, or information that is placed exactly as entered. However, there are also types of information that are usually authored in context with a thought or issue, but upon output, the content might be relocated, suppressed, or used only for purposes such as inline annotations for drafts. These elements include footnotes, index entries, draft comments, and special cleanup containers that can hold migrated data that still needs a writer's intervention to get into the right place.

- **draft-comment**

The <draft-comment> element allows simple review and discussion of topic contents within the marked-up content. Use the <draft-comment> element to ask a question or make a comment that you would like others to review. To indicate the source of the draft comment or the status of the comment, use the author, time or disposition attributes.

Processing systems should provide a run-time flag or parameter

[DRAFT-COMMENT Author:toshi Time:Sep 8, 2009 Status:accepted]
This parameter is implemented as "PRM_OUT_DRAFT_COMMENT".
to cause the content of this element to be specially displayed for draft output only. By default, it is stripped out to prevent publishing internal comments by mistake.

[DRAFT-COMMENT Author:EBP]
Where's the usage information for this section?

- **index-base**

The <index-base> element allows indexing extensions to be added by specializing off this element. It does not in itself have any meaning and should be ignored in processing.

The <index-base> element can only exist as a child of an <indexterm> element. This characteristic makes it the appropriate element to specialize to add indexing extensions. Specifically, the index-see , index-see-also, and index-sort-as elements only make sense as children of <indexterm> and so are specializations of <index-base>. Those elements are all part of the indexing domain, which is new for DITA 1.1.

- **tm**

The trademark (<tm>) element in DITA is used to markup and identify a term or phrase that is trademarked. Trademarks include registered trademarks, service marks, slogans and logos.

The business rules for indicating and displaying trademarks may differ from company to company and may be enforced by authoring policy and by specific processing.

The advantages of using DB2® Universal Database™ are well known.

Hyper Gear Corporation offers Worldwide Techinical Assistant Service℠ for their products.

- **data-about & data**

The <data-about> element identifies the subject of a property when the subject isn't associated with the context in which the property is specified. The property itself is expressed

by the <data> element. The <data-about> element handles exception cases where a property must be expressed somewhere other than inside the actual subject of the property. The <data-about> element is particularly useful as a basis for specialization in combination with the <data> element.

The <data> element represents a property within a DITA topic or map. While the <data> element can be used directly to capture properties, it is particularly useful as a basis for specialization. Default processing treats the property values as an unknown kind of metadata, but custom processing can match the name attribute or specialized element to format properties as sidebars or other adornments or to harvest properties for automated processing.

2.7 Specialization elements testing

Several DITA elements exist either for architectural reasons or for support of specialized markup yet to be designed. Although there is little need to use these elements unless you are directed to, some of them, such as <state>, can be used if your content makes use of these semantic distinctions. A discussion of signals on a gate of an integrated logic circuit, for example, might use the state element to represent either on or off conditions of that gate.

- **itemgroup**

The <itemgroup> element is reserved for use in specializations of DITA. As a container element, it can be used to sub-divide or organize elements that occur inside a list item, definition, or parameter definition.

1. First point of a list.
2. Second point of a list.

   related discourse

- **required-cleanup**

A <reqired-cleanup> element is used as a placeholder for migrated elements that cannot be appropriately tagged without manual intervention. As the element name implies, the intent for authors is to clean up the contained material and eventually get rid of the <reqired-cleanup> element. Authors should not insert this element into documents.

Note

Because the content of <reqired-cleanup> is not considered to be verified data, **DITA processors are required to strip this element from output by default.** A runtime flag may be provided to allow a draft view of migrated content in context.

[REQUIRED-CLEANUP Remap:center]
Some original content migrated from a tag.

- **state**

The <state> element specifies a name/value pair whenever it is necessary to represent a named state that has a variable value. The element is primarily intended for use in specializations to represent specific states (like logic circuit states, chemical reaction states, airplane instrumentation states, and so forth).

1. Verify the presence of an "on" or high condition at the input gate (ie, inflag=high)

- **term**

The <term> element identifies words that may have or require extended definitions or explanations. In future development of DITA, for example, terms might provide associative linking to matching glossary entries.

The *reference implementation* of DITA represents the standard, *"fallback"* behaviors intended for DITA elements.

- **boolean**

  The <boolean> element is used to express one of two opposite values, such as yes or no, on or off, true or false, high or low, and so forth. The element itself is empty; the value of the element is stored in its state attribute, and the semantic associated with the value is typically in a specialized name derived from this element.

  Note

  This element is deprecated. It is functionally equivalent to <state value="yes|no"/>, which is recommended as its replacement in all cases.

  She said "yes" when I asked her to marry me!

- **glossentry & no-topic-nesting**

  Following is the glossentry & no-topic-nesting test data.

2.8 Typographic elements testing

The typographic elements are used to highlight text with styles (such as bold, italic, and monospace). Never use these elements when a semantically specific element is available. These elements are not intended for use by specializers, and are intended solely for use by authors when no semantically appropriate element is available and a formatting effect is required.

- **b element**

  The bold (<b>) element is used to apply bold highlighting to the content of the element. Use this element only when there is not some other more proper element. For example, for specific items such as GUI controls, use the <uicontrol> element. This element is part of the DITA highlighting domain.

  STOP! This is **very** important!

- **i element**

  The italic (<i>) element is used to apply italic highlighting to the content of the element. Use this element only when there is not some other more proper element. For example, for specific items such as GUI controls, use the <uicontrol> element. This element is part of the DITA highlighting domain.

  Unplug the unit *before* placing the metal screwdriver against the terminal screw.

- **u element**

  The underline (<u>) element is used to apply underline highlighting to the content of the element. Use this element only when there is not some other more proper element. For example, for specific items such as GUI controls, use the <uicontrol> element. This element is part of the DITA highlighting domain.

  Beware: overuse *of* **highlighting** is sometimes known as font-itis!

- **tt element**

  The teletype (<tt>) element is used to apply monospaced highlighting to the content of the element. Use this element only when there is not some other more proper element. For example, for specific items such as GUI controls, use the <uicontrol> element. This element is part of the DITA highlighting domain.

  Make sure that the screen displays `File successfully created` before proceeding to the next stage of the task.

- **sup/sub element**

  The superscript (<sup>) element indicates that text should be superscripted, or vertically raised in relationship to the surrounding text. Superscripts are usually a smaller font than the surrounding text. Use this element only when there is not some other more proper tag. This element is part of the DITA highlighting domain.

A subscript (<sub>) indicates that text should be subscripted, or placed lower in relationship to the surrounding text. Subscripted text is often a smaller font than the surrounding text. Formatting may vary depending on your output process. This element is part of the DITA highlighting domain.

The power produced by the electrohydraulic dam was 10^{10} more than the older electric plant. The difference was H_2O.

2.9 Programming elements testing

The programming domains elements are used to define the syntax and to give examples of programming languages.

- **apiname**

The <apiname> element provides the name of an application programming interface (API) such as a Java class name or method name. This element is part of the DITA programming domain, a special set of DITA elements designed to document programming tasks, concepts and reference information.

Use the `document.write` method to create text output in the dynamically constructed view.

- **codeblock**

The <codeblock> element represents lines of program code. Like the <pre> element, content of this element has preserved line endings and is output in a monospaced font. This element is part of the DITA programming domain, a special set of DITA elements designed to document programming tasks, concepts and reference information.

```
/* a long sample program */
Do forever
Say "Hello, World"
End
```

Following is the codeblock template. scale="90" frame="all"

```
<!--
 function:    codeblock template
 param:        prmTopicRef
 return:    fo:block
 note:
 -->
<xsl:template match="*[contains(@class, ' pr-d/codeblock ')]" priority="2"
>
    <xsl:param name="prmTopicRef" required="yes"  as="element()?"/>
    <xsl:param name="prmNeedId"   required="yes"  as="xs:boolean"/>

    <fo:block>
        <xsl:copy-of select="ahf:getAttributeSet('atsCodeBlock')"/>
        <xsl:copy-of select="ahf:getDisplayAtts(.,'atsCodeBlock')"/>
        <xsl:copy-of select="ahf:getUnivAtts(.,$prmTopicRef,$prmNeedId)"/>
        <xsl:apply-templates>
            <xsl:with-param name="prmTopicRef" select="$prmTopicRef"/>
            <xsl:with-param name="prmNeedId"   select="$prmNeedId"/>
        </xsl:apply-templates>
    </fo:block>
</xsl:template>
```

- **codeph**

  The code phrase (<codeph>) element represents a snippet of code within the main flow of text. The code phrase is displayed in a monospaced font for emphasis. This element is part of the DITA programming domain, a special set of DITA elements designed to document programming tasks, concepts and reference information.

  The second line of the sample program code, `Do forever`, represents the start of a loop construct.

- **option**

  The <option> element describes an option that can be used to modify a command (or something else, like a configuration). This element is part of the DITA programming domain, a special set of DITA elements designed to document programming tasks, concepts and reference information.

  The `--valid` option of `xmllint` command enables to validate the document in addition to standard well-formed check.

- **parmname**

  When referencing the name of an application programming interface parameter within the text flow of your topic, use the parameter name (<parmname>) element to markup the parameter. This element is part of the DITA programming domain, a special set of DITA elements designed to document programming tasks, concepts and reference information.

  Use the `/env` parameter of the `config` command to update the field value.

- **parml**

  The parameter list (<parml>) element contains a list of terms and definitions that describes the parameters in an application programming interface. This is a special kind of definition list that is designed for documenting programming parameters. This element is part of the DITA programming domain, a special set of DITA elements designed to document programming tasks, concepts and reference information.

  This code example is a basic method signature:

```
returnType methodName(pList1, pList2) {
```

  where

  pList1
  is the first variable declaration passed to methodName

  pList2
  is the second variable declaration passed to methodName

  Table 2.9-1 parml in table sample

parml	
This code example is a basic method signature:	This code example is a basic method signature:

parml	
`returnType methodName(pList1, pList2 ) {`	`returnType methodName(pList1, pList2 ) {`
where **pList1** is the first variable declaration passed to methodName **pList2** is the second variable declaration passed to methodName	where **pList1** is the first variable declaration passed to methodName **pList2** is the second variable declaration passed to methodName

- ## synph

The syntax phrase (<synph>) element is a container for syntax definition elements. It is used when a complete syntax diagram is not needed, but some of the syntax elements, such as kwd, oper, delim, are used within the text flow of the topic content. This element is part of the DITA programming domain, a special set of DITA elements designed to document programming tasks, concepts and reference information.

[DRAFT-COMMENT Author:toshi]
Stylesheet does not apply special formatting for synph.

Synph sample: `format` *volumename*

- ## syntaxdiagram

This section contains sample data from Yahoo! dita-users group file section.

The syntax diagram (<syntaxdiagram>) element is the main container for all the syntax elements that make up a syntax definition. The syntax diagram represents the syntax of a statement from a computer language, or a command, function call or programming language statement. Traditionally, the syntax diagram is formatted with “railroad tracks” that connect the units of the syntax together, but this presentation may differ depending on the output media. The syntax diagram element is part of the DITA programming domain, a special set of DITA elements designed to document programming tasks, concepts and reference information.

Example from syntaxdiagram

```
COPYF
input-filename*INFILE
output-filename*OUTFILE
{input-filename | *INFILE}
{output-filename | *OUTFILE}
```

Figure 2.9-1 CopyFile

Example from "Table-based rendering of DITA Syntax Diagram markup"

```
ERASE {FORM | PROC | QUERY | TABLE} name [(CONFIRM= {YES | NO} )]
```

Figure 2.9-2 SAA CPI Database Reference

Example from "fragref"

```
COPYF
input-filename*INFILE
output-filename*OUTFILE
<Overlay>
{input-filename | *INFILE}
{output-filename | *OUTFILE}
  Overlay
{*OVERLAP | *Prompt}
```

Figure 2.9-3 CopyFile

Example from "Combining groups"

```
{ {http | https | ftp | file} :// [user@] host [:port] / | [/] } path
```

Figure 2.9-4 URL syntax

Example from "Optional elements"

```
  Optional importance
uname [-a]
  With empty sequence in groupchoice
uname { | -a}
```

Figure 2.9-5 `uname` can take the -a option

Example from "Default elements"

```
uname-s
```

Figure 2.9-6 `uname` assumes the -s option by default

Example from "importance on groupchoice"

```
uname [ {-a | -m | -n | -p | -s | -r | -v} ]
```

Figure 2.9-7 `uname` synopsis

Example from "Repetition using empty repsep element"

```
0x {digit | letter-a-to-f | letter-A-to-F} ( {digit | letter-a-to-f | letter-A-to-F} ) *
```

Figure 2.9-8 Hexadecimal literal

Example from "Repetition using non-empty repsep element"

```
number (, number) *
```

Figure 2.9-9 Comma-separated list of numbers

Example from "repsep is stateless"

A [*A* [*A*]]

Figure 2.9-10 Repeating a group *A* exactly 1 to 3 times

```
{1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}
[ {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9} ( {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9} ) * ]
```

Figure 2.9-11 Positive decimal integer

Example from "Importance on repsep"

adjective (, *adjective*) * *noun*

Figure 2.9-12 Applying adjectives to a noun

Example from "Repetition of optional groups"

Optional group, itself containing repsep

[*A* (, *A*) *]

Each repeated *A* is optional; commas can be adjacent

[*A*] (, [*A*]) *

Commas cannot be adjacent; entire production may be empty

[*A* (, *A*) *]

Additional sample: repetation is required.

A (, *A*) +

Figure 2.9-13 How repsep in combination with importance is disambiguated

Example from "Complex repetition separators"

Repetition with simple separator B

A (B *A*) *

Repetition with complex separator B

*A*B (*A*B) * *A*

Figure 2.9-14 Refactoring repsep

Example from "Fragments"

<IPv4-range integer> . <IPv4-range integer> . <IPv4-range integer> . <IPv4-range integer>

IPv4-range integer

IPv4-range integer Group Choice

{ <decimal digit> | {1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9} <decimal digit> | 1 <decimal digit> <decimal digit> | 2 {0 | 1 | 2 | 3 | 4} <decimal digit> | 25 {0 | 1 | 2 | 3 | 4 | 5} }

decimal digit

{0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}

Figure 2.9-15 IPV4 address in dot form

Example from "Notes"

Pattern 1

[- *1] {0 *2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9} ({0 *2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}) * *3

Pattern 2

[{± *4 | - *4}] *digit* (*digit*) *

*1 Minus sign must not be followed by zero digit.

*2 Zero must not be chosen for the first digit, unless it is the only digit.

*3 Thousands separator of , or . may occur every three digits.

*4 Sign must not be followed by zero.

Figure 2.9-16 Decimal integer

Example from "Notes": using @callout

Pattern 1

[-#a] {0#b | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9} ({0#b | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}) * #c

Pattern 2

[{± *1 | - *1}] *digit* (*digit*) *

#a Minus sign must not be followed by zero digit.

#b Zero must not be chosen for the first digit, unless it is the only digit.

#c Thousands separator of , or . may occur every three digits.

*1 Sign must not be followed by zero.

Figure 2.9-17 Decimal integer

2.10 Software elements testing

The software domain elements are used to describe the operation of a software program.

- **msgph**

  The message phrase (<msgph>) element contains the text content of a message produced by an application or program. It can also contain the variable name (varname) element to illustrate where variable text content can occur in the message.

  A server log entry of `I:0` is equivalent to the text message, `informational: successful`.

- **msgblock**

  The message block (<msgblock>) element contains a multi-line message or set of messages. The message block can contain multiple message numbers and message descriptions, each enclosed in a <msgnum> and <msgph> element. It can also contain the message content directly.

  A sequence of failed password attempts generates the following characteristic message stream:

```
I:0
S:3
I:1
S:3
I:1
S:4
S:99 (lockup)
```

  msgblock frame="all" scale="90"

```
check-arg:
[mkdir] Created dir: D:\DITA-OT1.5-M19\temp\temp20090917161047500
[echo] ****************************************************************
[echo] * basedir = D:\DITA-OT1.5-M19
[echo] * dita.dir = D:\DITA-OT1.5-M19
[echo] * input = samples/ud/bookmap_ud.ditamap
[echo] * transtype = pdf
[echo] * tempdir = temp\temp20090917161047500
[echo] * outputdir = out
[echo] * extname = .xml
[echo] * clean.temp = no
[echo] * xslt.parser = SAXON
[echo] ****************************************************************
```

- **msgnum**

  The message number (<msgnum>) element contains the number of a message produced by an application or program.

  A server log entry of `I:0` is equivalent to the text message, `informational: successful`.

- **cmdname**

  The command name (<cmdname>) element specifies the name of a command when it is part of a software discussion.

  After the DOS command prompt (`C:\>`) has displayed, change directory by entering `CD` command. You should change current directory to the DITA-OT folder.

- **varname**

  The variable name (<varname>) element defines a variable that must be supplied to a software application. The variable name element is very similar to the variable (var) element, but variable name is used outside of syntax diagrams.

  varname sample

  install-dir\projects\working*project-dir*\source*filename*.java

- **filepath**

  The <filepath> element indicates the name and optionally the location of a referenced file by specifying the directory containing the file, and other directories that may precede it in the system hierarchy.

  Uncompress the `gbbrsh.gz` file to the `/usr` directory. Ensure that the `/usr/tools/data.cfg` path is listed in the execution path system variable.

- **userinput/systemoutput**

  The user input (<userinput>) element represens the text a user should input in response to a program or system prompt.

  The system output (<systemoutput>) element represents computer output or responses to a command or situation. A generalized element, it represents any kind of output from the computer, so the author may wish to choose more specific markup, such as msgph, for messages from the application.

  After you type `mealplan dinner`, the meal planning program will print a `For what day?` message. Reply by typing the day of the week for which you want a meal plan, for example, `Thursday`.

2.11 Utility elements testing

The utilities domain elements represent common features of a language that may not necessarily be semantic, such as image maps.

- **imgmap**

The imagemap element supports the basic functionality of the HTML “client-side” image map markup. Imagemap allows you to designate a linkable area or region over an image, allowing a link in that region to display another topic.

2.12 Fig and table testing

2.12.1 Fig testing

The figure (<fig>) element is a display context (sometimes called an “exhibit”) with an optional title for a wide variety of content. Most commonly, the figure element contains an image element (a graphic or artwork), but it can contain several kinds of text objects as well. A title is placed inside the figure element to provide a caption to describe the content.

- **GIF sample**

  This is simple GIF sample

  The <desc> element contains the description of the current element. A description should provide more information than the title. This is its behavior in fig/table/linklist, for example. In xref/link, it provides a description of the target; processors that support it may choose to display this as hover help. In object, it contains alternate content for use when in contexts that cannot display the object.

  Figure 2.12-1 Flower

- **%display-atts; sample**

  Figure 2.12-2 Flower frame='all'

  Figure 2.12-3 Flower frame='topbot'

- **image attribute sample**

  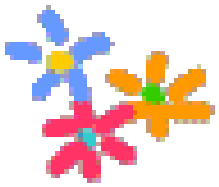

  Figure 2.12-4 Flower width="3cm"

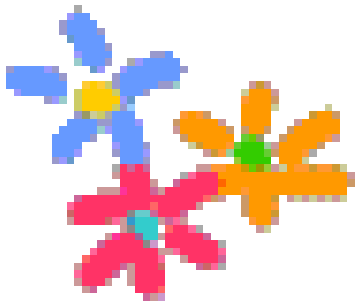

Figure 2.12-5 Flower height="5cm"

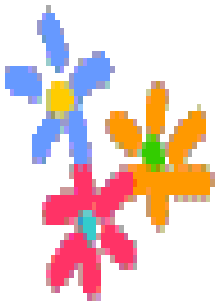

Figure 2.12-6 Flower width="3cm", height="5cm"

Figure 2.12-7 Flower scale="200"

Figure 2.12-8 Flower placement="break"&align="center"

Figure 2.12-9 Flower placement="break"&align="rifht"

Figure 2.12-10 Flower placement="break"&align="left"

Figure 2.12-11 Flower placement="break"&align="current"

- **Jpeg sample**

Figure 2.12-12 Dragon

- **figgroup sample from dita1.2**

These elementsare grouped togetherfor some purpose

First group

name="MetaItem" value="13"name="MetaThing" value="31"These elementsare grouped with associated metadata

Second group

Figure 2.12-13 Sample complex figure

These elementsare grouped togetherfor some purpose

First group

name="MetaItem" value="13"name="MetaThing" value="31"These elementsare grouped with associated metadata

Figure 2.12-14 Sample complex figure (2)

These elementsare grouped togetherfor some purpose

First group

name="MetaItem" value="13"name="MetaThing" value="31"These elementsare grouped with associated metadata

Second group

Figure 2.12-15 Sample complex figure (3): no frame

■ Fig testing in nested topic

● Figure in the nested topic

Figure 2.12-16 Flower

■ Image testing

● Image that have @longdescref

Figure 2.12-17 Link to Wikipedia

2.12.2 Table testing

● Table testing

The <table> element organizes arbitrarily complex relationships of tabular information. This standard table markup allows column or row spanning and table captions or descriptions. An optional title allowed inside the table element provides a caption to describe the table.

See simpletable for a simplified table model that can be specialized to represent more regular relationships of data.

The DITA table is based on the OASIS Exchange Table Model, augmented with DITA attributes which enable it for specialization, conref, and other DITA processing. In addition, the table includes a desc element, which enables table description that is parallel with figure description.

In DITA tables, in place of the expanse attribute used by other DITA elements, the pgwide attribute is used in order to conform with the OASIS Exchange Table Model. This attribute has a similar semantic (1=page width; 0=resize to galley or column).

Note

The scale attribute represents a stylistic markup property that is maintained for now in tables for legacy purposes. External stylesheets should enable less dependency on this attribute. You should use the scale attribute judiciously in your topics.

Table 2.12-1 Normal Table Sample

This table shows the relationships of table items between Microsoft Word, DocBook and XSL-FO.

Item	Word	DocBook	XSL-FO	Notes
Table Style	w:tbl/w:tblPr/ w:tblStyle/@w:val	table@style	-	-
Table width	w:tbl/w:tblPr/ w:tblw	table/@pgwide(?)	fo:table/@width	-
Column width	w:tblGrid/w:grid-Col/@w:w	tgroup/colspec/ @colwidth	fo:table-column/ @column-width	-
Header row	w:tr/w:trPr/ @w:tblHeader	tgroup/thead	fo:table-header	-
Body row	-	tgroup/tbody	fo:table-body	-

Table 2.12-2 Normal Table Sample with scale="140" frame="none"

Item	Word	DocBook	XSL-FO	Notes
Table Style	w:tbl/ w:tblPr/ w:tblStyle/ @w:val	table@style	-	-
Table width	w:tbl/ w:tblPr/ w:tblw	table/ @pgwide(?)	fo:table/ @width	-
Column width	w:tblGrid/ w:gridCol/ @w:w	tgroup/ colspec/ @colwidth	fo:table-col-umn/@col-umn-width	-
Header row	w:tr/w:trPr/ @w:tblHead-er	tgroup/thead	fo:table-header	-
Body row	-	tgroup/tbody	fo:table-body	-

Table 2.12-3 Normal Table Sample with rowheader='firstcol'

Item	Word	DocBook	XSL-FO	Notes
Table Style	w:tbl/w:tblPr/ w:tblStyle/@w:val	table@style	-	-

Item	Word	DocBook	XSL-FO	Notes
Table width	w:tbl/w:tblPr/ w:tblw	table/@pgwide(?)	fo:table/@width	-
Column width	w:tblGrid/w:grid-Col/@w:w	tgroup/colspec/ @colwidth	fo:table-column/ @column-width	-
Header row	w:tr/w:trPr/ @w:tblHeader	tgroup/thead	fo:table-header	-
Body row	-	tgroup/tbody	fo:table-body	-

Table 2.12-4 Normal Table with rowsep="0" colsep="0"

Item	Word	DocBook	XSL-FO	Notes
Table Style	w:tbl/w:tblPr/ w:tblStyle/@w:val	table@style	-	-
Table width	w:tbl/w:tblPr/ w:tblw	table/@pgwide(?)	fo:table/@width	-
Column width	w:tblGrid/w:grid-Col/@w:w	tgroup/colspec/ @colwidth	fo:table-column/ @column-width	-
Header row	w:tr/w:trPr/ @w:tblHeader	tgroup/thead	fo:table-header	-
Body row	-	tgroup/tbody	fo:table-body	-

Table 2.12-5 Normal Table with pgwide="1"

Item	Word	DocBook	XSL-FO	Notes
Table Style	w:tbl/w:tblPr/ w:tblStyle/@w:val	table@style	-	-
Table width	w:tbl/w:tblPr/w:tblw	table/@pgwide(?)	fo:table/@width	-
Column width	w:tblGrid/w:grid-Col/@w:w	tgroup/colspec/ @colwidth	fo:table-column/ @column-width	-
Header row	w:tr/w:trPr/ @w:tblHeader	tgroup/thead	fo:table-header	-
Body row	-	tgroup/tbody	fo:table-body	-

Table 2.12-6 dl in table sample

Name	Description	Data Type	Default Value	Required?
frame	Specifies which portion of a border should surround the element. Allowable values are: **top** Draw a line before the element **bottom** Draw a line after the element	(top | bottom | topbot | all sides | none | -dita-use-conref-target)	#IMPLIED	No

Name	Description	Data Type	Default Value	Required?
	topbot Draw a line both before and after the element **all** Draw a box around the element **sides** Draw a line at each side of the element **none** Don't draw any lines around this element **-dita-use-conref-target** See "using the -dita-use-conref-target value" for more information. Some DITA processors or output formats may not able to support all values.			
colsep	Column separator. A value of 0 indicates no separators; 1 indicates separators.	NMTOKEN	#IMPLIED	No
rowsep	Row separator. A value of 0 indicates no separators; 1 indicates separators.	NMTOKEN	#IMPLIED	No
pgwide	Determines the horizontal placement of the element. Supported values are 1 and 0, although these are not mandated by the DTD. For PDF, the value ″1″ places the element on the left page margin; ″0″ aligns the element with the left margin of the current text line and takes indention into account. For XHTML, the table surrounds the table data. Either value sets the table width to 100%.	NMTOKEN	#IMPLIED	No
rowheader	This attribute specifies whether the content of the first column in a table contains row headings. In the same way that a column header introduces a table column, the row header introduces the table row. This attribute makes tables whose first column contains row headings more readable on output. Allowable values are:	(firstCol | norowheader| -dita-use-conreftarget)	#IMPLIED	No

Name	Description	Data Type	Default Value	Required?
	firstcol The first column contains the row headings. **norowheader** Indicates that no column contains row headings. This is the processing default. **-dita-use-conref-target** See “Using the -dita-use-conref-target value” for more information.			

Table 2.12-7 sl in table sample

sl
Messages from the ags_open module are identical with messages from: ags_read ags_write ags_close sl/@compact='yes' Messages from the ags_open module are identical with messages from: ags_read ags_write ags_close

Table 2.12-8 ol in table sample

ol
Here are the colors of the rainbow in order of appearance from top to bottom: 1. Red a. Blood red b. Thin red 2. Orange a. Blood orange b. Thin orange 3. Yellow 4. Green 5. Blue 6. Indigo 7. Violet 8. Red 9. Orange 10. Yellow 11. Green

ol
12. Blue 13. Indigo 14. Violet

Table 2.12-9 ul in table sample

ul

Here are the colors of the rainbow in order of appearance from top to bottom:
• Red
  • Blood red
  • Thin red
• Orange
  • Blood orange
  1. Blood orage #1
  2. Blood orage #2
  3. Blood orage #3
  a. Blood orage #3-1
  b. Blood orage #3-2
  c. Blood orage #3-3
  i. Blood orage #3-3-1
  ii. Blood orage #3-3-2
  iii. Blood orage #3-3-3
  • Thin orange
• Yellow
• Green
• Blue
• Indigo
• Violet
• Red
  1. Blood red
  2. Thin red
• Orange
  1. Blood orange
  2. Thin orange
• Yellow
• Green
• Blue
• Indigo
• Violet

Stylesheet user can customize <ol> number formats by modifying config/[langage-code]_style.xml.

Table 2.12-10 fig in table sample

fig
This is the tiff image of book&bird. Figure 2.12-18 Book & bird

Table 2.12-11 Valign sample

	aligned top	aligned middle	aligned bottom

Table 2.12-12 Cell span sample

horizontally spanned		vertically spanned
vertically spanned		
	horizontally spanned	

Table 2.12-13 Table align sample (No cell rule)

NO.1	NO.2	NO.3
a1	b1	1.0

NO.1	NO.2	NO.3
a2		0.0001
a3		111.01
		12.34

Table 2.12-14 Table align sample

NO.1	NO.2	NO.3
a1	b1	1.0
a2		0.0001
a3		111.01
		12.34

- ## Simple table testing

The <simpletable> element is used for tables that are regular in structure and do not need a caption. Choose the simple table element when you want to show information in regular rows and columns. For example, multi-column tabular data such as phone directory listings or parts lists are good candidates for simpletable. Another good use of simpletable is for information that seems to beg for a ″three-part definition list″—just use the keycol attribute to indicate which column represents the ″key″ or term-like column of your structure.

The keycol attribute is not specified.The styleshhet will apply font-weight="bold" & background-color="rgb(217,217,217)" to the header row.

Type style	Elements used
Bold	b
Italic	i
Underlined	u

keycol="1": The styleshhet will apply font-weight="bold" & background-color="rgb(217,217,217)" to the column number 1. Then the first row's background-color will become white.

Type style	Elements used
Bold	b
Italic	i
Underlined	u

keycol="2": The styleshhet will apply font-weight="bold" & background-color="rgb(217,217,217)" to the column number 2. Then the first row's background-color will become white.

Type style	Elements used
Bold	**b**

Type style	**Elements used**
Italic	**i**
Underlined	**u**

scale="150": The styleshhet will apply font-size 1.5 * 9pt for this table.

Type style	**Elements used**
Bold	b
Italic	i
Underlined	u

relcolwidth="90* 150*": The styleshhet will apply fixed table layout and width will be 100%. The column width will be 37.5% and 62.5%.

Type style	**Elements used**
Bold	b
Italic	i
Underlined	u

frame="topbot": The styleshhet will apply top/bottom border for table

Type style	**Elements used**	**FO description**
Bold	b	font-weight="bold"
Italic	i	font-style="italic"
Underlined	u	text-decoration="underline"

frame="none": The styleshhet will remove all surrounding border from table

Type style	**Elements used**	**FO description**
Bold	b	font-weight="bold"
Italic	i	font-style="italic"
Underlined	u	text-decoration="underline"

expanse="page": The styleshhet will apply start-indent="0mm", end-indent="0mm" to surrounding fo:block and the width of table will reach to page margin.

Type style	**Elements used**	**FO description**
Bold	b	font-weight="bold"
Italic	i	font-style="italic"
Underlined	u	text-decoration="underline"

■ Additional table testing in nested topic

Table 2.12-15 Normal Table Sample (Again)

Item	Word	DocBook	XSL-FO	Notes
Table Style	w:tbl/w:tblPr/ w:tblStyle/@w:val	table@style	-	-
Table width	w:tbl/w:tblPr/ w:tblw	table/@pgwide(?)	fo:table/@width	-
Column width	w:tblGrid/w:grid-Col/@w:w	tgroup/colspec/ @colwidth	fo:table-column/ @column-width	-
Header row	w:tr/w:trPr/ @w:tblHeader	tgroup/thead	fo:table-header	-
Body row	-	tgroup/tbody	fo:table-body	-

2.12.3 Properties

- **No @relcolwidth**

Visual Element	Value	Implication
color	red	depicts anger
	green	depicts permission

- **@relcolwidth="1* 2* 3*"**

Visual Element	Value	Implication
color	red	depicts anger
	green	depicts permission

- **scale="200"**

Visual Element	Value	Implication
color	red	depicts anger
	green	depicts permission

- **keycol="1"**

Visual Element	Value	Implication
color	red	depicts anger
	green	depicts permission

- **keycol="2"**

Visual Element	Value	Implication
color	red	depicts anger

Visual Element	**Value**	Implication
	green	depicts permission

- **frame="none"**

Visual Element	**Value**	**Implication**
color	red	depicts anger
	green	depicts permission

- **expanse="page"**

Visual Element	**Value**	**Implication**
color	red	depicts anger
	green	depicts permission

2.12.4 Foreign testing

The foeign (<foreign>) element allows the introduction of non-DITA content, for example, MathML, SVG, or Rich Text Format (RTF). This test data is made for fig/foreign elements.

- **fig/foreign for MathML example**

This example is downloaded from http://www.antenna.co.jp/AHF/FOsample/FOsample.htm#EB.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Figure 2.12-19 MathML example (1)

This example is also downloaded from http://www.antenna.co.jp/AHF/FOsample/FOsample.htm#EB.

$$J = \left| \frac{\partial(f_1, f_2, \ \ldots, f_n)}{\partial(x_1, x_2, \ \ldots, x_n)} \right|$$

Figure 2.12-20 MathML example (2)

- **fig/foreign for SVG example**

This example is also downloaded http://www.antenna.co.jp/AHF/FOsample/foGraphic/SVG/MP1.svg.

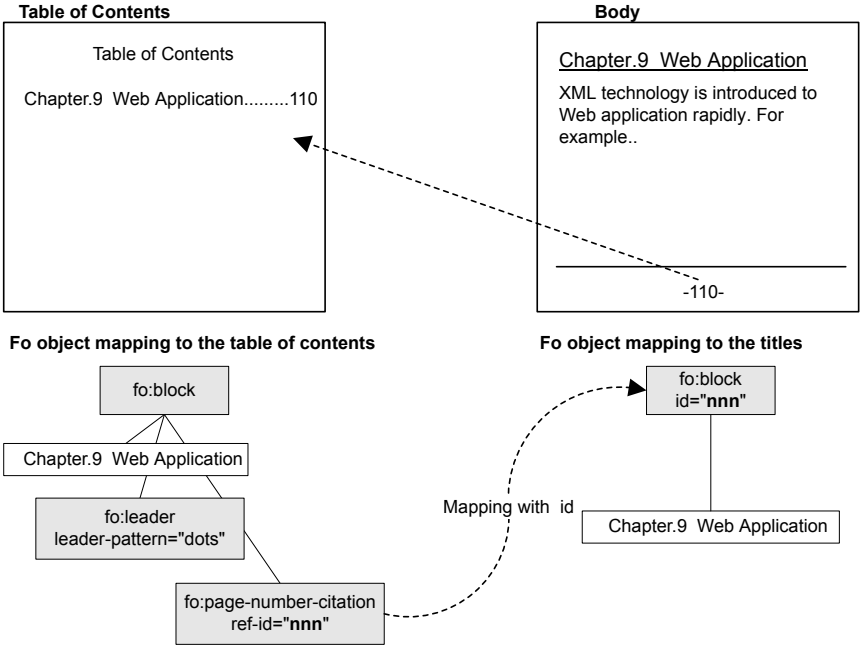

Figure 2.12-21 SVG example

Chapter 3. Logging in to DB3 Client

You must log in to the DB3 Client to access any DB3 functions.

The system administrator must set up a DB3 account for you to use.

DB3 supports two types of authentication: Windows Authentication and DB3 Authentication. Use the appropriate type, depending on the type of user account you have. Contact your DB3 system administrator if you are not sure. If you are logging in to DB3 Client for the first time after a new installation of DB3, use DB3 Authentication.

1. From your desktop, select [**Start**]>[**DB3**]>[**DB3 Manager**] from the menu.

   The following DB3 - Login dialog box appears.

The next step explains how to authenticate your login.

2. Authenticate your login in one of the following ways:

   - Windows authentication: Select [**Windows Authentication**] from the Authentication drop-down list. Both the User name and Password text boxes become greyed out, with the text box User name automatically displaying your Windows login name. You do not need to edit the contents.

     Tip

     If your windows user account belongs to more then one DB3 group, a dialog appears, asking you to select a DB3 group. In this case, select the DB3 group of your choice, and click [**OK**].

   - DB3 Authentication:

     a. Select [**DB3 Authentication**] from the [**Authentication**] drop-down list.

     b. Enter your user name and password.

     If you are logging in for the first time after a new installation of DB3, you can use one of the following default user accounts. Log in using the ADMINISTRATOR account if you require a full set of read and write permissions for all features.

     Type "edit" after the command line prompt and press Enter. The following editing interface will be displayed.

```
File Edit Search View Options Help
+--------------------------------- UNTITLED1 ----------------------------------+
¦                                                                             ¦
¦                                                                             ¦
¦                                                                             ¦
¦                                                                             ¦
¦ Line:1 Col:1 F1=Help                                                        ¦
+-----------------------------------------------------------------------------+
```

```
File Edit Search View Options Help
+------------------------------- UNTITLED1 ---------------------------------+
¦                                                                           ¦
¦                                                                           ¦
¦                                                                           ¦
¦                                                                           ¦
¦ Line:1 Col:1 F1=Help                                                      ¦
+---------------------------------------------------------------------------+
```

If DB3 Client is configured to use multiple server sets, the dialog box shows an additional Server set list. In this case, select the server set you want to connect to from the list.

3. Click [**OK**]

   For example, `OK`

   Tip

   If the number of history records stored in DB3 Server exceeds 9999999, a dialog box appears, asking you to optimize the database on DB3 server. Click [**OK**] to close the dialog box. If you do not have adminstrative rights, contact the DB3 administrator.

   DB3 Client opens.

4. Then this

   a. which is done by doing this

   b. and then this.

Do something	Or Else this
Do this	and this will happen
Do that	and that will happen

Option	Description
Do this	and this will happen
Do that	and that will happen

Do something	Or Else this
Do this	and this will happen
Do that	and that will happen

Do something	Or Else this
Do this	and this will happen
Do that	and that will happen

Do something	Or Else this
Do this	and this will happen

Do something	Or Else this
Do that	and that will happen

Do something	Or Else this
Do this	and this will happen
Do that	and that will happen

Do something	Or Else this
Do this	and this will happen
Do that	and that will happen

Do something	Or Else this
Do this	and this will happen
Do that	and that will happen

Do something	Or Else this
Do this	and this will happen
Do that	and that will happen

Do something	Or Else this
Do this	and this will happen
Do that	and that will happen

Do something	**Or Else this**
Do this	**and this will happen**
Do that	**and that will happen**

Do something	Or Else this
Do this	and this will happen
Do that	and that will happen

5. Choose a server.
   - If you have a remote server you want to test on, type the IP address or hostname of the server here.
   - If you want to do local testing, just type localhost.
6. Do this

   In your editor, open the first element and click on the dialog.
7. Do that

   Move the framulator into the foobar box.

The SQLJ file is successfully created when the SQLJ server displays the "File Created" dialog.

Notify the proctor upon completing this self-test.

Chapter 4. DITA 1.2 element test

The Darwin Information Typing Architecture (DITA) 1.2 specification defines a set of document types for authoring and organizing topic-oriented information, as well as a set of mechanisms for combining, extending, and constraining document types.

The DITA 1.2 specification consists of the following components:

Document Type Definitions (DTDs) and XML Schemas (XSDs)

The DTDs and XSDs – along with the catalog files – define DITA markup for the DITA vocabulary modules and DITA document types. While the DTDs and XSDs should define the same DITA elements, the DTDs are normative if there is a discrepancy. If there is a discrepancy between the written specification (this document) and the DTDs, the written specification takes precedence.

DITA 1.2 written specification

While the DITA 1.2 documentation does contain some introductory information, it is intended neither as an introduction to DITA nor as a users guide. The intended audience of this documentation consists of implementers of the DITA standard, including tool developers and XML architects who develop specializations. The documentation contains several parts:

- Architectural specification
- Language reference
- Conformance statement
- Appendices

The DITA 1.2 written specification is available in the following formats: XHTML, CHM, PDF, and DITA source. The XHTML version is authoritative.

4.1 Longdescref testing

The <longdescref> element supports a reference to a text description of the graphic or object. This element replaces the deprecated longdescref attribute on image and object elements.

PDF output does not generate any results for <longdescref>.

- **Image example (1)**

- **Image example (2)**

4.2 Longquoteref testing

The <longquoteref> element provides a reference to the source of a long quote. The long quote (<lq>) element itself allows an href attribute to specify the source of a quote, but it does not allow other standard linking attributes such as keyref, scope, and format. The <longquoteref> element should be used for references that make use of these attributes.

Rendering of this element is left up to DITA processors. Depending on the presentation format, it may be appropriate to ignore the element, present it as a link, use it to turn the entire quote into a link, or do something else.

PDF output will generate link for <longdescref> if <lq> has no @reftitle attribute.

- **Longquoteref example (1)**

  Scorpa is a manufacturer of trials motorcycles based near Alès, France. It was founded in 1993 by Marc Teissier and Joël Domergue. The first model produced by the company was the WORKS 294 in 1994, powered by a single-cylinder, two-stroke Rotax engine. In 1998, Scorpa signed an agreement with Yamaha Motor Company to use its engines in subsequent models. The final range included trials bikes ranging from 125 cc to 249 cc, in both two- and four-stroke variants. Scorpa also produced "long ride" models, that were mechanically similar to the SY trials range, but with a larger fuel tank, and a seat to allow longer trekking use. Non-traditional models included the BMX-inspiried 4-Tricks and the trials/trail hybrid T-Ride. On July 13, 2009, Scorpa announced on its website that it was going into bankruptcy liquidation. [1] In September 2009 it was announced that original owner Marc Tessier had purchased the company. Operations have moved from Ales to Nimes, France, and a full complement of Trials models will be produced for 2010. These will include the TYS125 four-stroke, a 125 and 250 cc two-stroke SY, and a T-Ride model.

  Scorpa motorcycles

- **Longquoteref example (3)**

  Scorpa is a manufacturer of trials motorcycles based near Alès, France. It was founded in 1993 by Marc Teissier and Joël Domergue. The first model produced by the company was the WORKS 294 in 1994, powered by a single-cylinder, two-stroke Rotax engine. In 1998, Scorpa signed an agreement with Yamaha Motor Company to use its engines in subsequent models. The final range included trials bikes ranging from 125 cc to 249 cc, in both two- and four-stroke variants. Scorpa also produced "long ride" models, that were mechanically similar to the SY trials range, but with a larger fuel tank, and a seat to allow longer trekking use. Non-traditional models included the BMX-inspiried 4-Tricks and the trials/trail hybrid T-Ride. On July 13, 2009, Scorpa announced on its website that it was going into bankruptcy liquidation. [1] In September 2009 it was announced that original owner Marc Tessier had purchased the company. Operations have moved from Ales to Nimes, France, and a full complement of Trials models will be produced for 2010. These will include the TYS125 four-stroke, a 125 and 250 cc two-stroke SY, and a T-Ride model.

  Scorpa motorcycles

4.3 Sectiondiv testing

The <sectiondiv> element allows logical grouping of content within a section. There is no additional meaning associated with the sectiondiv element, aside from its function as a container for other content. The sectiondiv element does not contain a title; the lowest level of titled content within a topic is the section itself. If additional hierarchy is required, nested topics should be used in place of the section.

PDF output will generate fo:wrapper for <sectiondiv>.

- **Nice pets**

  Cats are nice.

  Dogs are nice.

  Friends of mine really love their hedgehogs.

  Lots of people want ponies when they grow up.

  Llamas are also popular.

Appendices

Appendix 1. XSL Transformations

XSL Transformations (XSLT) is a declarative XML-based language used for the transformation of XML documents into other XML documents. The original document is not changed; rather, a new document is created based on the content of an existing one. The new document may be serialized (output) by the processor in standard XML syntax or in another format, such as HTML or plain text. XSLT is often used to convert XML data into HTML or XHTML documents for display as a web page: the transformation may happen dynamically either on the client or on the server, or it may be done as part of the publishing process. It is also used to create output for printing or direct video display, typically by transforming the original XML into XSL Formatting Objects to create formatted output which can then be converted to a variety of formats, a few of which are PDF, PostScript, AWT and PNG. XSLT is also used to translate XML messages between different XML schemas, or to make changes to documents within the scope of a single schema, for example by removing the parts of a message that are not needed.

■ Origins

XSLT is developed by the World Wide Web Consortium (W3C). The most recent version is XSLT 2.0, which reached W3C recommendation status on 23 January 2007. As of 2008, however, XSLT 1.0 is still more widely used and implemented.

Originally, XSLT was part of the W3C's Extensible Stylesheet Language (XSL) development effort of 1998–1999, a project that also produced XSL Formatting Objects and the XML Path Language, XPath. The editor of the first version was James Clark. XSLT 1.0 was published as a Recommendation by the W3C on 16 November 1999. After an abortive attempt to create a version 1.1 in 2001, the XSL working group joined forces with the XQuery working group to create XPath 2.0, with a richer data model and type system based on XML Schema. XSLT 2.0, developed under the editorship of Michael Kay, was built on this foundation in 2002–2006.

As a language, XSLT is influenced by functional languages, and by text-based pattern matching languages like SNOBOL and awk. Its most direct predecessor was ISO DSSSL, a language that performed the same function for full SGML that XSLT performs for XML. Many of the standards committee that developed XSLT had previously worked on DSSSL, including James Clark. XSLT can also be considered Turing-complete template processor.

Most of this article is applicable to both XSLT versions; any differences are noted in the text.

■ Overview

The XSLT processing model involves:

- one or more XML source documents;

- one or more XSLT stylesheet modules;
- the XSLT template processing engine (the processor); and
- one or more result documents.

The XSLT processor ordinarily takes two input documents—an XML source document, and an XSLT stylesheet—and produces an output document. The XSLT stylesheet contains a collection of template rules: instructions and other directives that guide the processor in the production of the output document.

Template rule processing

The XSLT language is declarative—rather than listing an imperative sequence of actions to perform in a stateful environment, template rules only define how to handle a node matching a particular XPath-like pattern, if the processor should happen to encounter one, and the contents of the templates effectively comprise functional expressions that directly represent their evaluated form: the result tree, which is the basis of the processor's output.

The processor follows a fixed algorithm: Assuming a stylesheet has already been read and prepared, the processor builds a source tree from the input XML document. It then starts by processing the source tree's root node, finding in the stylesheet the best-matching template for that node, and evaluating the template's contents. Instructions in each template generally direct the processor to either create nodes in the result tree, or process more nodes in the source tree in the same way as the root node. Output is derived from the result tree.

Processor implementations

XSLT processor implementations fall into two main categories: server-side, and client-side.

Although client-side XSLT processing has been available in Microsoft's Internet Explorer since 1999 (or even earlier, but in a form that was incompatible with the W3C specifications), adoption has been slower because of the widespread deployment of older and alternative browsers without XSLT support. For similar reasons, adoption of XSLT 2.0 in such environments remains limited.

XSLT processors may be delivered as standalone products, or as components of other software including web browsers, application servers, frameworks such as Java and .NET, or even operating systems. For example, Windows XP comes with the MSXML3 library, which includes an XSLT 1.0 processor. Earlier versions may be upgraded and there are many alternatives. See the external links section.

Performance

The performance of XSLT processors has steadily improved as the technology has become more mature, although the very first processor, James Clark's xt, was unbeaten for several years.

Most of the earlier XSLT processors were interpreters; in more recent products, code generation is increasingly common, using portable intermediate languages such as Java bytecode or .NET Common Intermediate Language as the target. However, even the

interpretive products generally offer separate analysis and execution phases, allowing an optimized expression tree to be created in memory and reused to perform multiple transformations: this gives substantial performance benefits in online publishing applications where the same transformation is applied many times per second to different source documents. This separation is reflected in the design of XSLT processing APIs such as JAXP (Java API for XML Processing).

Early XSLT processors had very few optimizations; stylesheet documents were read into Document Object Models and the processor would act on them directly. XPath engines were also not optimized. Increasingly, however, XSLT processors use the kind of optimization techniques found in functional programming languages and database query languages, notably static rewriting of the expression tree for example to move calculations out of loops, and lazy pipelined evaluation to reduce the use of memory for intermediate results and allow "early exit" when the processor can evaluate an expression such as `following-sibling::*[1]` without a complete evaluation of all subexpressions. Many processors also use tree representations that are much more efficient (in both space and time) than general purpose DOM implementations.

The performance of XSLT processors has steadily improved as the technology has become more mature, although the very first processor, James Clark's xt, was unbeaten for several years.

Most of the earlier XSLT processors were interpreters; in more recent products, code generation is increasingly common, using portable intermediate languages such as Java bytecode or .NET Common Intermediate Language as the target. However, even the interpretive products generally offer separate analysis and execution phases, allowing an optimized expression tree to be created in memory and reused to perform multiple transformations: this gives substantial performance benefits in online publishing applications where the same transformation is applied many times per second to different source documents. This separation is reflected in the design of XSLT processing APIs such as JAXP (Java API for XML Processing).

Early XSLT processors had very few optimizations; stylesheet documents were read into Document Object Models and the processor would act on them directly. XPath engines were also not optimized. Increasingly, however, XSLT processors use the kind of optimization techniques found in functional programming languages and database query languages, notably static rewriting of the expression tree for example to move calculations out of loops, and lazy pipelined evaluation to reduce the use of memory for intermediate results and allow "early exit" when the processor can evaluate an expression such as `following-sibling::*[1]` without a complete evaluation of all subexpressions. Many processors also use tree representations that are much more efficient (in both space and time) than general purpose DOM implementations.

The performance of XSLT processors has steadily improved as the technology has become more mature, although the very first processor, James Clark's xt, was unbeaten for several years.

Most of the earlier XSLT processors were interpreters; in more recent products, code generation is increasingly common, using portable intermediate languages such as Java bytecode or .NET Common Intermediate Language as the target. However, even the interpretive products generally offer separate analysis and execution phases, allowing an optimized expression tree to be created in memory and reused to perform multiple

transformations: this gives substantial performance benefits in online publishing applications where the same transformation is applied many times per second to different source documents. This separation is reflected in the design of XSLT processing APIs such as JAXP (Java API for XML Processing).

Early XSLT processors had very few optimizations; stylesheet documents were read into Document Object Models and the processor would act on them directly. XPath engines were also not optimized. Increasingly, however, XSLT processors use the kind of optimization techniques found in functional programming languages and database query languages, notably static rewriting of the expression tree for example to move calculations out of loops, and lazy pipelined evaluation to reduce the use of memory for intermediate results and allow "early exit" when the processor can evaluate an expression such as `following-sibling::*[1]` without a complete evaluation of all subexpressions. Many processors also use tree representations that are much more efficient (in both space and time) than general purpose DOM implementations.

The performance of XSLT processors has steadily improved as the technology has become more mature, although the very first processor, James Clark's xt, was unbeaten for several years.

Most of the earlier XSLT processors were interpreters; in more recent products, code generation is increasingly common, using portable intermediate languages such as Java bytecode or .NET Common Intermediate Language as the target. However, even the interpretive products generally offer separate analysis and execution phases, allowing an optimized expression tree to be created in memory and reused to perform multiple transformations: this gives substantial performance benefits in online publishing applications where the same transformation is applied many times per second to different source documents. This separation is reflected in the design of XSLT processing APIs such as JAXP (Java API for XML Processing).

Early XSLT processors had very few optimizations; stylesheet documents were read into Document Object Models and the processor would act on them directly. XPath engines were also not optimized. Increasingly, however, XSLT processors use the kind of optimization techniques found in functional programming languages and database query languages, notably static rewriting of the expression tree for example to move calculations out of loops, and lazy pipelined evaluation to reduce the use of memory for intermediate results and allow "early exit" when the processor can evaluate an expression such as `following-sibling::*[1]` without a complete evaluation of all subexpressions. Many processors also use tree representations that are much more efficient (in both space and time) than general purpose DOM implementations.

■ XSLT and XPath

XSLT relies upon the W3C's XPath language for identifying subsets of the source document tree, as well as for performing calculations. XPath also provides a range of functions, which XSLT itself further augments. This reliance upon XPath adds a great deal of power and flexibility to XSLT.

XSLT 1.0 uses XPath 1.0. Similarly, XSLT 2.0 relies on XPath 2.0; both specifications were published on the same date.

XSLT and XQuery compared

XSLT capabilities overlap with XQuery, which was initially conceived as a query language for large collections of XML documents.

The XSLT 2.0 and XQuery 1.0 standards were developed by separate working groups within W3C, working together to ensure a common approach where appropriate. They share the same data model, type system, and function library, and both include XPath 2.0 as a sublanguage.

The two languages, however, are rooted in different traditions and serve the needs of different communities. XSLT was primarily conceived as a stylesheet language whose primary goal was to render XML for the human reader on screen, on the web (as web template language), or on paper. XQuery was primarily conceived as a database query language in the tradition of SQL.

Because the two languages originate in different communities, XSLT is stronger in its handling of narrative documents with more flexible structure, while XQuery is stronger in its data handling, for example when performing relational joins.

XSLT media types

As of 2009, there is no MIME/Internet media type registered for XSLT.

The XSLT 1.0 Recommendation (1999) says "The MIME media types text/xml and application/xml should be used for XSLT stylesheets. It is possible that a media type will be registered specifically for XSLT stylesheets; if and when it is, that media type may also be used." It goes on to use text/xml in an example of how to embed a stylesheet with the xml-stylesheet processing instruction.

RFC 3023 points out potential technical problems with text/* types in general, and proposes application/xslt+xml as an ideal media type for XSLT. The XSLT 2.0 Recommendation (January 2007) includes a formal application to register this media type. However, at the time of writing (January 2009) the process of registration has not yet been completed, and RFC 3023 warns that "... this media type should not be used until such registration has been completed."

Pre-1.0 working drafts of XSLT used text/xsl in their embedding examples, and this type was implemented and continues to be promoted by Microsoft in Internet Explorer and MSXML. It is also widely recognized in the xml-stylesheet processing instruction by other browsers. In practice, therefore, users wanting to control transformation in the browser using this processing instruction are obliged to use this unregistered media type.

Above contents are the excerpts from Wikipedia to test range indexterm. (http://en.wikipedia.org/wiki/XSLT)

Backmatter testing

The <backmatter> element contains the material that follows the main body of a document and any appendices. It may include items such as a colophon, legal notices, and various types of book lists such as a glossary or an index.

The backmatter has following content model

```
(amendments or booklists or colophon or dedication or notices or to
picref or anchorref or keydef or mapref or topicgroup or topichead
or topicset or topicsetref) (any number)
```

Glossary

Document Object Model The Document Object Model (DOM) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents. Aspects of the DOM (such as its "Elements") may be addressed and manipulated within the syntax of the programming language in use. The public interface of a DOM is specified in its application programming interface (API).

Document Type Definition Document Type Definition (DTD) is a set of markup declarations that define a document type for SGML-family markup languages (SGML, XML, HTML). DTDs were a precursor to XML schema and have a similar function, although different capabilities. DTDs use a terse formal syntax that declares precisely which elements and references may appear where in the document of the particular type, and what the elements' contents and attributes are. DTDs also declare entities which may be used in the instance document. XML uses a subset of SGML DTD.

HTML Hypertext Markup Language (HTML) is the predominant markup language for web pages. HTML elements are the basic building-blocks of webpages.

HTML is written in the form of HTML elements consisting of tags, enclosed in angle brackets (like <html>), within the web page content. HTML tags most commonly come in pairs like <h1> and </h1>, although some tags, known as empty elements, are unpaired, for example <img>. The first tag in a pair is the start tag, the second tag is the end tag (they are also called opening tags and closing tags). In between these tags web designers can add text, tags, comments, and other types of text-based content.

Simple API for XML SAX (Simple API for XML) is an event-based sequential access parser API developed by the XML-DEV mailing list for XML documents. SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the Document Object Model (DOM). Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially.

XML Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form. It is defined in the XML 1.0 Specification[*1] produced by the W3C, and several other related specifications, all gratis open standards.[*2]

*1 "XML 1.0 Specification". W3.org. Retrieved 2010-08-22.

*2 "W3C DOCUMENT LICENSE".

XML Schema (W3C) XML Schema, published as a W3C recommendation in May 2001, is one of several XML schema languages. It was the first separate schema language for XML to achieve Recommendation status by the W3C. Because of confusion between XML Schema as a specific W3C specification, and the use of the same term to describe schema languages in general, some parts of the user community referred to this language as WXS, an initialism for W3C XML Schema, while others referred to it as XSD, an initialism for XML Schema Document—a document written in the XML Schema language, typically containing the "xsd" XML namespace prefix and stored with the ".xsd" filename extension. In Version 1.1 (currently in July 2011 a Candidate Recommendation), the W3C has chosen to adopt XSD as the preferred name, and that is the name used in this article.

XSL Formatting Objects XSL Formatting Objects, or XSL-FO, is a markup language for XML document formatting which is most often used to generate PDFs. XSL-FO is part of XSL (Extensible Stylesheet Language), a set of W3C technologies designed for the transformation and formatting of XML data. The other parts of XSL are XSLT and XPath. As of December 12, 2006, the current version of XSL-FO is v1.1

XSLT XSLT (Extensible Stylesheet Language Transformations) is a declarative, XML-based language used for the transformation of XML documents. The original document is not changed; rather, a new document is created based on the content of an existing one.[*1] The new document may be serialized (output) by the processor in standard XML syntax or in another format, such as HTML or plain text.[*2] XSLT is most often used to convert

data between different XML schemas or to convert XML data into web pages or PDF documents.

*1 XSL Transformations (XSLT)

*2 See e. g., http://www.w3.org/TR/xslt#output, specifying alternate output methods.

Index

Symbols

#all..... 3
#default 3
see also Mode parameter (template)
<dita> sample indexterm in <metadata>..... i
<dita> sample range indexterm in <metadata> .....i,ii
<dita> sample range indexterm in <metadata> that have @end.....i,ii
<index-base>.....13

C

Carp 3
see also Goldfish
Change the license..... 3
cheese 3
goats milk 3
chevre..... 3
see also cheese , sheeps milk , pecorino
see also Cheese maker
sheeps milk
pecorino.....3
see also cheese , goats milk , chevre
Cheese maker 3

D

<data>..... 3
DITA 1.2 Testing in <metadata>..... 47-51
DITA 1.2 Testing in <topicmeta>.....47-57
DITA sample indexterm <topicmeta>..... i
DITA sample range indexterm <topicmeta>..... .....i-57
DITA sample range indexterm <topicmeta> that has @end..... i,ii

E

error1
`error2`
`error3`
`error4`
`error5` 3

F

Feeding 3
see also Goldfish , feeding
Feeding goldfish
see Carp
see Goldfish , feeding

G

Goldfish 3
feeding..... 3

I

Indexterm of Note Section Title.....6
Indexterm of Note Shortdesc..... 6
Indexterm of Note Title.....6

L

license..... 3
License.....3
Developer license.....3
For Java..... 3
For Visual C++..... 3
Server license.....3
Standalone license..... 3
License control PC.....3
License Control PC.....3
License control software.....3

M

Mode parameter (template).....3

R

Range indexterm of Note Section Title..... 6-8
Range indexterm of Note Shortdesc.....6-8
Range indexterm of Note title..... 6-8

T

Testing Examples 3-42
Title
testing.....1
between page..... 1,2
Title testing..... 1
Title testing(2).....2

W

W3C
World Wide Web Consortium.....52

X

XSLT.....52-56
Performance.....53-55
XSL Transformations (in title).....52
XSLT testing in appendices <topicmeta>52-57
XSLT testing in appendix <topicmeta>...52-57

Index