

DITA for the Impatient

Hussein Shafie

Pixware

91, rue Gambetta

78120 Rambouillet

France

Phone: +33 (0)1 30 59 81 44

xmleditor-support@xmlmind.com

ditac-support@xmlmind.com

www.xmlmind.com/xmleditor/

www.xmlmind.com/ditac/

November 9, 2009

Table of Contents

Chapter 1. Introduction	1
Chapter 2. About topics and maps	2
Chapter 3. The topic element	3
1. The structure of the topic element	3
2. The most commonly used “block elements”	4
3. The most commonly used “inline elements”	7
4. The xref and link elements	8
Chapter 4. Specialized topic types	12
1. The concept element	12
2. The task element	12
3. The reference element	14
4. The glossentry element	15
Chapter 5. Topic maps	17
1. The map element	17
2. The bookmap element	19
Chapter 6. Conclusion	21

List of Figures

2-1. File layout of this tutorial	2
3-1. The photo converted to black and white	6
3-2. The TopicGo to dialog box	10

List of Tables

3-1. Sample CALS table 7

Chapter 1. Introduction

By reading this short tutorial, you'll get acquainted with the [DITA 1.1](#) markup and after that, you'll be able to author your first DITA document⁽¹⁾ right away.

This short tutorial will not discuss the DITA ``philosophy" or the advantages of the DITA vocabulary over other XML vocabularies (e.g. [DocBook](#)).

This article is published under the [Creative Commons "Attribution-Share Alike"](#) license.

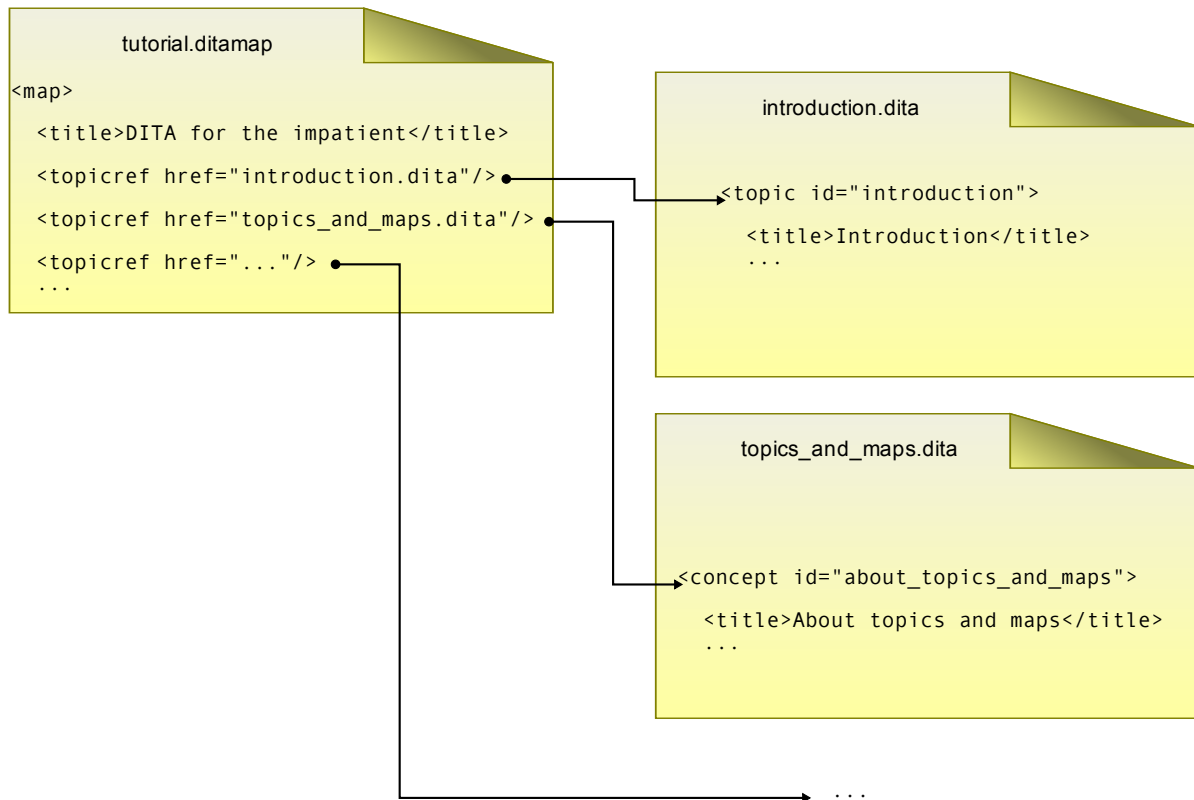
⁽¹⁾ Preferably using a DITA-aware XML editor such as [XMLmind XML Editor Personal Edition](#) (free to use).

Chapter 2. About topics and maps

A DITA document is necessarily modular. The information unit used to compose a DITA document is called a *topic*. As its name suggests it, a topic addresses a single subject.

The overall contents of a DITA document is specified using a *topic map* (also simply called a *map*). A map mainly contains a hierarchy of topic references.

Figure 2-1. File layout of this tutorial



- Remember**
- It is recommended to have a single topic per XML file.
 - The recommended filename extension for a topic file is ".dita".
 - The recommended filename extension for a map file is ".ditamap".
 - Topic and map files may be contained in different directories. You are free to organize the contents of these directories as you wish.

Chapter 3. The `topic` element

In this chapter,

we'll first explain what is the structure of a `topic` element;
 then we'll list the most useful “block elements” (paragraph, table, list, etc);
 then we'll list the most useful “inline elements” (bold, italic, etc);
 finally, we'll explain how to create internal and external links. Note that creating internal links in DITA, which is inherently modular, is slightly harder than in other, generally monolithic, document types. Therefore it is important to read this section carefully.

1. The structure of the `topic` element

Remember A `topic` element must always be given an ID. Use the `id` attribute to specify it.

The structure of the `topic` element is very simple:

1. A `title` element.
2. An optional `shortdesc` or `abstract` element.

The `shortdesc` element is longer and more descriptive than the `title` element. However, it is recommended to keep it short, approximately one paragraph long, because the contents of this element is often used during navigation (e.g. to generate a detailed table of contents).

The `abstract` element is intended to contain more information than the `shortdesc` element.

3. A `body` element⁽²⁾.
4. An optional `related-links` element.

This is a kind of “**See Also**” section containing a list of `link` elements. Topics being generally short, this section stands out clearly after the body of a topic. That's why it is often preferred to spreading links inside the body of a topic.

Example:

```
<topic id="docbook_or_dita">
  <title>DITA or DocBook?</title>

  <shortdesc>Both DITA and DocBook are both mature, feature rich, document types,
  so which one to choose?</shortdesc>

  <body>
    <p>DocBook 5 is a mature document type, well-documented (DocBook: The
    Definitive Guide, DocBook XSL: The Complete Guide), coming with decent XSL
    stylesheets allowing to convert it to a variety of formats, based on the
    best schema technologies: RELAX NG and Schematron.</p>

    <p>DITA concepts (topics, maps, specialization, etc) have an immediate
    appeal to the technical writer and indeed, makes this document type more
    attractive than DocBook. However the DocBook vocabulary is comprehensive
    and very well thought. So choose DITA if its standard vocabulary is
    sufficiently expressive for your needs or if, anyway, you intend to
    specialize DITA.</p>
  </body>
</topic>
```

⁽²⁾ The `body` element is optional too. However creating a `topic` element having no `body` child element is mainly a trick which is more simply implemented by adding a `topichead` to a map.

```

<related-links>
  <link format="html" href="http://www.docbook.org/" scope="external">
    <linktext>DocBook 5</linktext>
  </link>

  <link format="html"
    href="http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dita"
    scope="external">
    <linktext>DITA</linktext>
  </link>
</related-links>
</topic>

```

The above example is rendered as follows:

DITA or DocBook?

Both DITA and DocBook are both mature, feature rich, document types, so which one to choose?

DocBook 5 is a mature document type, well-documented (DocBook: The Definitive Guide, DocBook XSL: The Complete Guide), coming with decent XSL stylesheets allowing to convert it to a variety of formats, based on the best schema technologies: RELAX NG and Schematron.

DITA concepts (topics, maps, specialization, etc) have an immediate appeal to the technical writer and indeed, makes this document type more attractive than DocBook. However the DocBook vocabulary is comprehensive and very well thought. So choose DITA if its standard vocabulary is sufficiently expressive for your needs or if, anyway, you intend to specialize DITA.

Related information

- ➔ [DocBook 5](#)
- ➔ [DITA](#)

2. The most commonly used “block elements”

The most commonly used block elements are borrowed from HTML and as such, should be immediately familiar to the reader.

Paragraphs and lists

A paragraph is represented by the `p` element.

A preformatted paragraph is represented by the `pre` element.

An itemized list is represented by the `ul` element. As expected, it contains `li` list item elements.

An ordered list is represented by the `ol` element.

A variable list is represented by the `dl` element. Unlike HTML's `d1`, the `dt` (term being defined) and the `dd` (term definition) elements must be wrapped in a `dlentry` element.

Example:

```

<ul>
  <li>First item.
    <p>Continuation paragraph.</p>
  </li>

```

```

<li>Second item. This item contains an ordered list.
  <ol>
    <li>First do this.</li>
    <li>Then do that.</li>
    <li>Finally do this.</li>
  </ol>
</li>

<li>Third item. This item contains a variable list.
  <dl>
    <dentry>
      <dt>Term #1</dt>
      <dd>Definition of term #1.</dd>
    </dentry>

    <dentry>
      <dt>Term #2</dt>
      <dd>Definition of term #2.</dd>
    </dentry>
  </dl>
</li>
</ul>

```

The above example is rendered as follows:

- First item.
Continuation paragraph.
- Second item. This item contains an ordered list.
 1. First do this.
 2. Then do that.
 3. Finally do this.
- Third item. This item contains a variable list.

Term #1
Definition of term #1.

Term #2
Definition of term #2.

Sections

DITA has no h1, h2, h3, etc, heading elements. Instead it has the `section` element which generally always has a `title` child element. Note that `section` elements cannot nest. Example:

```

<section>
  <title>The customary "hello word" program in Tcl/Tk</title>

  <pre frame="all">button .hello -text "Hello, World!" -command { exit }
    pack .hello</pre>
</section>

```

The above example is rendered as follows:

The customary “hello word” program in Tcl/Tk

```

button .hello -text "Hello, World!" -command { exit }
pack .hello

```

Figures and examples

Of course, DITA has also figure, table and example elements.

The `example` element is just a specialized kind of `section`.

The `fig` element generally has a `title` and generally contains an image element.

Like `img`, its HTML counterpart, the `image` “inline element” may be contained in any “block element”. The graphics file which is to be displayed is specified using the `href` attribute. The `image` element also has `width`, `height`, `scale` and `align` attributes.

Example:

```
<example>
  <title>Converting a color image to black and white</title>

  <pre>$ convert -dither Floyd-Steinberg -monochrome photo.png bwphoto.gif</pre>

  <fig>
    <title>The photo converted to black and white</title>
    <image href="bwphoto.gif" align="center" />
  </fig>
</example>
```

The above example is rendered as follows:

Example 3-1. Converting a color image to black and white

```
$ convert -dither Floyd-Steinberg -monochrome photo.png bwphoto.gif
```

Figure 3-1. The photo converted to black and white



Tables

DITA has two kinds of table element: `simpletable` which is specific to DITA, and `table` which is in fact a `CALS` table (also know as a DocBook table).

`Simpletable` example:

```
<simpletable relcolwidth="1* 2* 3*">
  <thead>
    <stentry>A</stentry>
    <stentry>B</stentry>
    <stentry>C</stentry>
  </thead>

  <strow>
    <stentry>A,1</stentry>
    <stentry>B,1</stentry>
    <stentry>C,1</stentry>
  </strow>

  <strow>
    <stentry>A,2</stentry>
    <stentry>B,2</stentry>
    <stentry>C,2</stentry>
  </strow>
</simpletable>
```

A `simpletable` element contains an optional `sthead` element, followed by one or more `strow` elements. Both row elements, `sthead` and `strow`, contain `stentry` cell elements. The `relcolwidth` attribute may be used to specify the relative width of each column.

The above example is rendered as follows:

A	B	C
A,1	B,1	C,1
A,2	B,2	C,2

Same as above example but using a CALS table this time:

```
<table>
  <title>Sample CALS table</title>

  <tgroup cols="3">
    <colspec colwidth="1*" />
    <colspec colwidth="2*" />
    <colspec colwidth="3*" />

    <thead>
      <row>
        <entry align="center">A</entry>
        <entry align="center">B</entry>
        <entry align="center">C</entry>
      </row>
    </thead>

    <tbody>
      <row>
        <entry>A,1</entry>
        <entry>B,1</entry>
        <entry>C,1</entry>
      </row>

      <row>
        <entry>A,2</entry>
        <entry>B,2</entry>
        <entry>C,2</entry>
      </row>
    </tbody>
  </tgroup>
</table>
```

CALS tables are quite complex and explaining how they can be used is out of the scope of this tutorial. Our recommendation is to use CALS tables rather than `simpletables` only when you want a cell to span more than one row and/or more than one column.

The above example is rendered as follows:

Table 3-1. Sample CALS table

A	B	C
A,1	B,1	C,1
A,2	B,2	C,2

3. The most commonly used “inline elements”

The most generic inline elements are also borrowed from HTML: `i` (italic), `b` (bold), `tt` (teletype or monospaced text), `sub` (subscript), `sup` (superscript).

- Always specify the `format` attribute of the link element. Example: given an URL such as `"http://www.xmlmind.com/ditac/"`, the DITA processing software has no way to guess that this corresponds to an HTML file.
- Always specify attribute `scope="external"` for such links. By doing so, you instruct the DITA processing software to use the target location *as is* in the deliverable.

Internal links

The target of an internal link (`xref` or `link` element) is a DITA element belonging to the same DITA document. This target element may be found in the same XML file as the link element or, on the contrary, in a different XML file. The later case is still considered to be an internal link because both the link and its target belong to the same overall DITA document.

Of course, in order to use a DITA element as the target of a link, this element must have an `id` attribute.

The value of the `href` attribute of a link element is:

`"URL_of_the_target_DITA_file#qualified_ID_of_the_target_element"`, where:

- `URL_of_the_target_DITA_file` may be omitted when both the link and its target are found in the same XML file.
- `#qualified_ID_of_the_target_element` may be omitted if you want to address the first topic contained in an XML file.

What is the *qualified ID* of the target element?

- It's simply the value of the `id` attribute for a topic element (of any kind: `topic`, `concept`, `reference`, etc).
- It's the value of the `id` attribute of the target element prefixed by `"ID_of_the_topic_ancestor/"` for any descendant element of a topic.

Example: Let's suppose you want to add an `xref` element to `topic1.dita`:

```
<topic id="t1">
  <title>Title of topic 1</title>
  <body>
    <p id="p1">Paragraph inside topic 1.</p>
    <p>More information in <xref href="???"></p>
  </body>
</topic>
```

Let's suppose that you want to address elements contained in `topic2.dita`, this file being found in the same directory as `topic1.dita`.

```
<topic id="t2">
  <title>Title of topic 2</title>
  <body>
    <p id="p2">Paragraph inside topic 2.</p>
  </body>
</topic>
```

- If you want to address topic "t1", specify `href="#t1"`.
- If you want to address paragraph "p1", specify `href="#t1/p1"`.
- If you want to address topic "t2", specify `href="topic2.dita#t2"` or more simply `href="topic2.dita"`.
- If you want to address paragraph "p2", specify `href="topic2.dita#t2/p2"`.

Remember • There is generally no need to specify the text of internal links, as the DITA processing software can automatically generate this text.

Example: converting the following paragraph to XHTML

```
<p>More information in <xref href="topic2.dita"></p>
```

may result in something like:

```
<p>More information in
  <a href="page-23.html#t2">Title of topic 2</a>.</p>
```

This probably works fine for any element having a title: `topic`, `section`, `table`, `fig`, etc. However this cannot work for the other elements.

For example, do not expect the DITA processing software to generate some text for:

```
<xref href="topic2.dita#t2/p2"/>
```

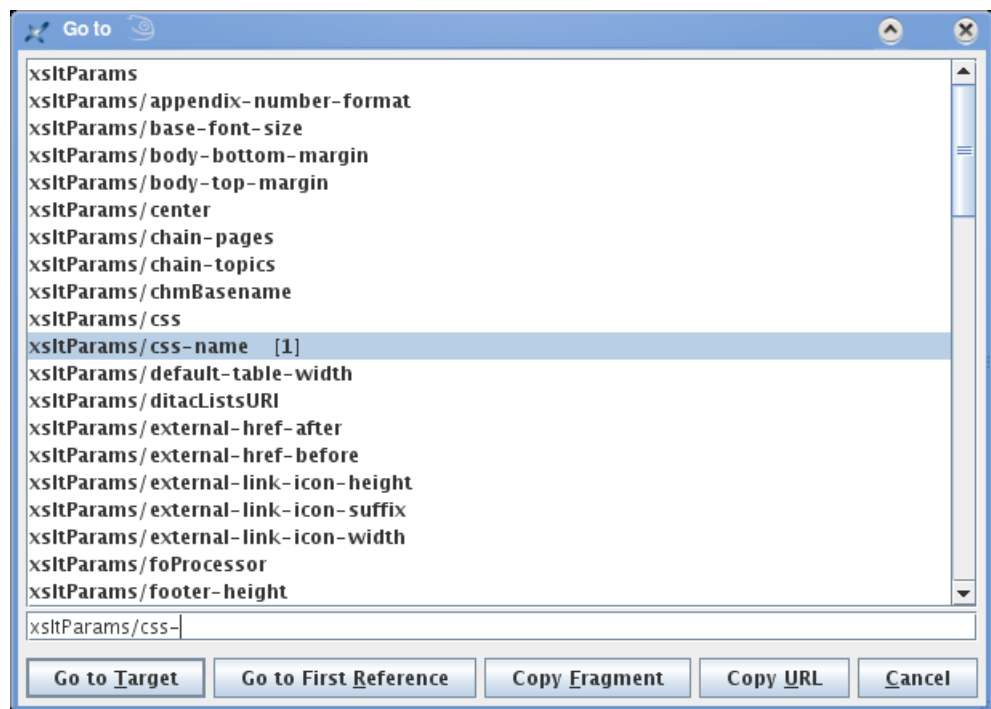
Instead explicitly specify some text:

```
<xref href="topic2.dita#t2/p2">this paragraph</xref>
```

- Link targets are tedious and error prone to specify by hand. Using a DITA-aware XML editor is therefore especially handy when it comes to inserting link elements.

XMLmind XML Editor example:

Figure 3-2. The **Topic** → **Go to** dialog box



You are inserting an `xref` element in `topic1.dita`.

- You want to address an element found in `topic1.dita`.
 1. Select menu item **Topic** → **Go to**.
 2. Select the ID of the target in the **Go to** dialog box. Note that this dialog box supports the auto-completion of ID values.
 3. Click **Copy Fragment**. This copies to the clipboard something like `#t1/p1`.
 4. Paste the copied fragment into the `href` attribute of the newly inserted `xref` element.
- You want to address an element found in `topic2.dita`.
 1. Open or switch to `topic2.dita`.
 2. Select menu item **Topic** → **Go to**.
 3. Select the ID of the target in the **Go to** dialog box.
 4. Click **Copy URL**. This copies to the clipboard something like `"topic2.dita#t2/p2"`.
 5. Switch back to `topic1.dita`.

6. Paste the copied URL into the `href` attribute of the newly inserted `xref` element.

Chapter 4. Specialized topic types

The `topic` element is the most generic topic type. There are four more specialized topic types: `concept`, `task`, `reference`, `glossentry`. When appropriate, use a specialized topic type rather than a plain `topic`.

1. The `concept` element

Create a `concept` element when you need to provide your reader with background information which must be absorbed in order to understand the rest of the document.

Example:

```
<concept id="what_is_a_cache">
  <title>What is a cache?</title>

  <shortdesc>Everything you'll ever need to know about
  <term>cache</term>s.</shortdesc>

  <conbody>
    <p>In computer science, a cache is a temporary storage area where
    frequently accessed data can be stored for rapid access.</p>
  </conbody>

  <related-links>
    <link format="html" href="http://en.wikipedia.org/wiki/Cache"
    scope="external">
      <linktext>Wikipedia definition of a cache</linktext>
    </link>
  </related-links>
</concept>
```

Notice how the structure of a `concept` element is similar to the structure of a `topic` element. Moreover, a `conbody` element has almost the same content model as a `body` element.

The above example is rendered as follows:

What is a cache?

Everything you'll ever need to know about *caches*.

In computer science, a cache is a temporary storage area where frequently accessed data can be stored for rapid access.

Related information

↳ [Wikipedia definition of a cache](#)

2. The `task` element

Create a `task` element when you need to explain step by step which procedure is to be followed in order to accomplish a given task.

Example:

```
<task id="install_emacs">
  <title>Installing GNU Emacs</title>
```

```

<taskbody>
  <prereq>Windows NT 4.0 or any subsequent version of Windows. 5Mb of free
  disk space.</prereq>

  <steps>
    <step>
      <cmd>Unzip the distribution anywhere.</cmd>

      <info>We recommend to use the free, open source, <xref format="html"
      href="http://www.info-zip.org/" scope="external">Info-ZIP</xref>
      utility to do so.</info>

      <stepxmp><screen>C:\&gt; unzip emacs-21.3-bin-i386.zip</screen></stepxmp>

      <stepresult><p>Doing this will create an
      <filepath>emacs-21.3</filepath> directory.</p></stepresult>
    </step>

    <step>
      <cmd>Go to the bin subdirectory.</cmd>

      <stepxmp><screen>C:\&gt; cd emacs-21.3\bin</screen></stepxmp>
    </step>

    <step>
      <cmd>Run <cmdname>addpm</cmdname>.</cmd>

      <stepxmp><screen>C:\emacs-21.3\bin&gt; addpm</screen></stepxmp>

      <stepresult>A confirmation dialog box is displayed.<fig>
        <image href="confirm_install_emacs.png"/>
      </fig></stepresult>
    </step>

    <step>
      <cmd>Click <uicontrol>OK</uicontrol> to confirm.</cmd>
    </step>
  </steps>
</taskbody>
</task>

```

Albeit being the most complex specialized topic type, the `task` element is also the most useful one. Its `taskbody` is mainly organized around the `steps` element. Other useful elements are `prereq` (pre-requisite section of a task), `context` (background information for the task), `result` (expected outcome of a task).

The `step` element has several useful child elements other than the required `cmd` element: `info` (additional information about the step), `stepxmp` (example that illustrates a step), `substeps`, `choices` (the user needs to choose one of several actions), `stepresult` (expected outcome of a step).

The above example is rendered as follows:

Installing GNU Emacs

Before you begin

Windows NT 4.0 or any subsequent version of Windows. 5Mb of free disk space.

Procedure

1. Unzip the distribution anywhere.

We recommend to use the free, open source, [Info-ZIP](#) utility to do so.

```
C:\> unzip emacs-21.3-bin-i386.zip
```

Doing this will create an `emacs-21.3` directory.

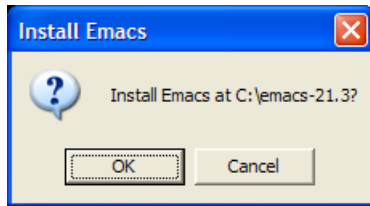
2. Go to the `bin` subdirectory.

```
C:\> cd emacs-21.3\bin
```

3. Run `addpm`.

```
C:\emacs-21.3\bin> addpm
```

A confirmation dialog box is displayed.



4. Click **OK** to confirm.

3. The reference element

Create a [reference](#) element when you need to add an entry to a reference manual. The [reference](#) element is typically used to document a command or a function.

Example:

```
<reference id="pwd_command">
  <title>The <cmdname>pwd</cmdname> command</title>

  <refbody>
    <refsyn><cmdname>pwd</cmdname></refsyn>

    <section><title>DESCRIPTION</title><p>Print the full filename of the
current working directory.</p><note>Your shell may have its own version of
<cmdname>pwd</cmdname>, which usually supersedes the version described
here.</note></section>

    <section><title>AUTHOR</title><p>Written by John Doe. </p></section>
  </refbody>

  <related-links>
    <link format="html" href="http://www.manpagez.com/man/3/getcwd/"
scope="external">
      <linktext><cmdname>getcwd</cmdname>(3)</linktext>
    </link>
  </related-links>
</reference>
```

The [refbody](#) child element of a [reference](#) can contain the following generic elements: [sections](#), [examples](#), [simpletables](#) and [tables](#) but also more specific elements: [refsyn](#) (contains the syntax of a command-line

utility or the prototype of a function) and `properties` (a special kind of table having 3 columns: type, value and description).

The above example is rendered as follows:

The `pwd` command

`pwd`

DESCRIPTION

Print the full filename of the current working directory.

Note Your shell may have its own version of `pwd`, which usually supersedes the version described here.

AUTHOR

Written by John Doe.

Related information

→ [getcwd\(3\)](#)

4. The `glossentry` element

Create a `glossentry` element when you need to add entry to a glossary.

The following example shows *three* glossary entries having the following IDs: `ajax`, `dhtml`, `javascript`.

```
<dita>
  <glossentry id="ajax">
    <glossterm>AJAX</glossterm>

    <glossdef><b>A</b>synchronous <b>Ja</b>vaScript and <b>X</b>ML. Web
    development techniques used on the client-side to create interactive web
    applications.</glossdef>
  </glossentry>

  <glossentry id="dhtml">
    <glossterm>DHTML</glossterm>

    <glossdef><b>D</b>ynamic <b>HTML</b>. Web development techniques used on
    the client-side to create interactive web sites.</glossdef>
  </glossentry>

  <glossentry id="javascript">
    <glossterm>JavaScript</glossterm>

    <glossdef>JavaScript is an object-oriented scripting language supported by
    all major web browsers. It allows the development of interactive web sites
    and web applications.</glossdef>

    <related-links>
      <link format="html" href="https://developer.mozilla.org/en/JavaScript"
        scope="external">
        <linktext>Mozilla's Official Documentation on JavaScript</linktext>
      </link>
    </related-links>
  </glossentry>
</dita>
```

```
</glossentry>  
</dita>
```

The `glossentry` element is the simplest specialized topic type. It contains a `glossterm` child element (the term being defined) followed by a `glossdef` (the definition of the term) child element, optionally followed by the `related-links` element common to all topic types.

- Remember**
- It is not recommended to create XML files containing several topics because if you do so, first this makes it harder reusing your topics in different documents and second, this makes the topic map slightly harder to specify⁽³⁾.

However if you need to create multi-topic files, you'll have to use the `dita` element as a wrapper for your topics. This is what we have done in the above example. Note that a file containing several `glossentry` elements is, in our opinion, the only case for which it makes sense creating a multi-topic file.

- A topic may contain other topics. The DITA grammar allows to add a topic child element after the `body` or `related-links` element (whichever is the last child) of the parent topic.

It is not recommended to use this nested topics facility which, in our opinion, is almost never useful. The hierarchy of topics (that is, chapters containing sections containing subsections, etc) is better expressed in a map using a hierarchy of `topicrefs`⁽⁴⁾.

The above example is rendered as follows:

Glossary

AJAX

Asynchronous **JavaScript** and **XML**. Web development techniques used on the client-side to create interactive web applications.

DHTML

Dynamic **HTML**. Web development techniques used on the client-side to create interactive web sites.

JavaScript

JavaScript is an object-oriented scripting language supported by all major web browsers. It allows the development of interactive web sites and web applications.

Related information

➔ [Mozilla's Official Documentation on JavaScript](#)

⁽³⁾You'll need to specify attribute `chunk="select-document"` in the corresponding `topicref` of your `map`. More about the `chunk` attribute later in this tutorial.

⁽⁴⁾More about topic maps later in this tutorial.

Chapter 5. Topic maps

1. The `map` element

A topic `map` mainly contains:

- A `title` child element.
- A `topicmeta` element where you can specify the author of the document, the date of publication, etc.
- A hierarchy of `topicref` elements.

The `href` attribute of a `topicref` element specifies the URL of a topic which is part of the DITA document.

Example:

```
<topicref href="samples/sample_glossary.dita"/>
```

If the target XML file contains several topics (not recommended), you'll have to use a fragment to specify the ID of the referenced topic.

```
<topicref href="samples/sample_glossary.dita#javascript"/>
```

A map contains a *hierarchy* of `topicref` elements. What does this mean?

```
<topicref href="topic.dita">
  <topicref href="topic_structure.dita">
    <topicref href="samples/sample_topic.dita"/>
  </topicref>
  <topicref href="block_elements.dita"/>
  <topicref href="inline_elements.dita"/>
  <topicref href="link_elements.dita"/>
</topicref>
```

In the case of the above example, this means two things:

1. The overall DITA document contains this sequence of topics: `topic.dita`, `topic_structure.dita`, `samples/sample_topic.dita`, `block_elements.dita`, `inline_elements.dita`, `link_elements.dita`.
2. Topics `topic_structure.dita`, `block_elements.dita`, `inline_elements.dita`, `link_elements.dita` are *subsections* of topic `topic.dita`. Topic `samples/sample_topic.dita` is a subsection of topic `topic_structure.dita`.

If you instruct the DITA processing software to generate a Table of Contents for your document and/or to number the topics, the hierarchy of topics appears very clearly.

What follows is the topic map actually used for this tutorial (contents of file `tutorial.ditamap`):

```
<map chunk="by-document">
  <title>DITA for the Impatient</title>

  <topicmeta>
    <author>Hussein Shafie</author>
    <publisher>Pixware</publisher>
    <critdates>
      <created date="October 7, 2009"/>
    </critdates>
  </topicmeta>

  <topicref chunk="to-content" href="introduction.dita"/>
  <topicref chunk="to-content" href="topics_and_maps.dita"/>
  <topicref chunk="to-content" href="topic.dita">
```

```

<topicref chunk="to-content" href="topic_structure.dita">
  <topicref href="samples/sample_topic.dita" toc="no"/>
</topicref>
<topicref chunk="to-content" href="block_elements.dita"/>
<topicref chunk="to-content" href="inline_elements.dita"/>
<topicref chunk="to-content" href="link_elements.dita"/>
</topicref>
.
.
.
<topichead chunk="to-content" navtitle="Topic maps">
  <topicref href="map.dita"/>
  <topicref href="bookmap.dita"/>
</topichead>
<topicref chunk="to-content" href="conclusion.dita"/>
</map>

```

The chunk attribute

Notice how attribute `chunk="to-content"` is specified on most `topicref` elements. This attribute means, to make it simple: create a separate HTML page containing this topic.

Specifying proper values for the `chunk` attribute is almost always required for all HTML based formats: HTML Help, Java™ Help, Eclipse Help, EPUB, etc. On the other hand, this attribute is completely ignored for all print formats: PDF, PostScript®, RTF, ODT, etc.

Also notice how attribute `chunk="by-document"` is specified on the `map` root element. This attribute means: a generated HTML page must have the same basename as the corresponding XML file. The other possible value is `chunk="by-topic"`. In this case, it is the the ID of the topic which is used to compute a filename for the HTML page.

Specifying attribute `copy-to` allows to override any automatically computed filename for an HTML page. Example:

```

<topicref chunk="to-content" copy-to="index.html"
  href="introduction.dita"/>

```

Without `copy-to="index.html"`, the contents of `introduction.dita` would have been found in `introduction.html`.

The toc attribute

Specifying attribute `toc="no"` for a `topicref` element allows to prevent it from appearing in the generated Table of Contents.

```

<topicref chunk="to-content" href="topic_structure.dita">
  <topicref href="samples/sample_topic.dita" toc="no"/>
</topicref>

```

The topichead element

The `topichead` element provides an author with a simple way to group several topics in the same HTML page and to give this HTML page a title⁽⁵⁾.

```

<topichead chunk="to-content" navtitle="Topic maps">
  <topicref href="map.dita"/>
  <topicref href="bookmap.dita"/>
</topichead>

```

⁽⁵⁾ A less convenient alternative would be to use an actual `topic` having no body at all, just a title.

2. The `bookmap` element

A `bookmap` element is just a more elaborate form of `map`. We recommend using a `bookmap` for anything more complex than an article.

Tip You don't need to create a `map` for the screen media and a `bookmap` for the print media. If you follow the “one topic per XML file” rule, a *single topic map* (`map` or `bookmap` depending on the complexity of the contents) is all what you need.

- A `bookmap` may have a `booktitle` rather than a `title`.
- Its metadata wrapper element, `bookmeta`, may contain richer information than the `topicmeta` element.
- A `bookmap` may contain specializations of the `topicref` element having stronger semantics: `part`, `chapter`, `appendix`.
- The hierarchy of references to topic elements which makes up the body of the document may be preceded by a `frontmatter` element and followed by a `backmatter` element.

These wrapper elements can contain references to actual, hand-written, topics: `bookabstract`, `preface`, `dedication`, `colophon`, etc.

However the most common use of `frontmatter` and `backmatter` is to contain the following, empty, placeholder elements: `toc`, `figurelist`, `tablelist`, `indexlist`. These placeholders instructs the DITA processing software to automatically generate: a Table of Contents, a List of Figures, a List of Tables, an Index.

What follows is a possible `bookmap` for this tutorial (contents of file `tutorial-book.ditamap`):

```
<bookmap chunk="by-document">
  <booktitle>
    <mainbooktitle>DITA for the Impatient</mainbooktitle>
  </booktitle>

  <bookmeta>
    <authorinformation>
      <personinfo>...</personinfo>
      <organizationinfo>...</organizationinfo>
    </authorinformation>
    <critdates>
      <created date="October 7, 2009"/>
    </critdates>
  </bookmeta>

  <frontmatter chunk="to-content">
    <booklists>
      <toc/>
      <figurelist/>
      <tablelist/>
    </booklists>
  </frontmatter>

  <chapter chunk="to-content" href="introduction.dita"/>
  <chapter chunk="to-content" href="topics_and_maps.dita"/>
  <chapter chunk="to-content" href="topic.dita">
    <topicref chunk="to-content" href="topic_structure.dita">
      <topicref href="samples/sample_topic.dita" toc="no"/>
    </topicref>
    <topicref chunk="to-content" href="block_elements.dita"/>
    <topicref chunk="to-content" href="inline_elements.dita"/>
    <topicref chunk="to-content" href="link_elements.dita"/>
  </chapter>
```

```
.  
. .  
.  
<chapter chunk="to-content" navtitle="Topic maps">  
  <topicref href="map.dita"/>  
  <topicref href="bookmap.dita"/>  
</chapter>  
<chapter chunk="to-content" href="conclusion.dita"/>  
</bookmap>
```

Chapter 6. Conclusion

This tutorial has just scratched the surface of DITA. We didn't discuss:

- A lot of useful elements such as: `screen`, `note`, `fn` (footnote), `lq` (long quote), `q` (quote), etc.
- The `reltable` child element of topic maps which allows to link different topics without explicitly adding a `related-links` section to each of them.
- Document meta-data: `topicmeta`, `prolog`, etc.
- Conditional processing.
- The `conref` transclusion mechanism which allows to reuse fine-grained content.

However we believe that what you have learned here is sufficient to start authoring your first DITA document.

Related information

- ↳ [DITA Language Specification v1.1 \(OASIS specification\)](#)
- ↳ [DITA Architectural Specification v1.1 \(OASIS specification\)](#)
- ↳ [DITA Open Toolkit, the reference DITA implementation](#)
- ↳ [XMLmind DITA Converter, an alternative to using the DITA Open Toolkit](#)