DITA/ XML Practical Guide

Structured Authoring for Technical Documentation

This guide provides a practical DITA XML example that demonstrates how structured authoring works using concept, task, and reference topics. It includes a real DITA map, content reuse patterns, metadata, and a diagram illustrating topic relationships within a scalable documentation system.

Darwin Information Typing Architecture (DITA) harnesses XML as the industry leading solution for structured content. It is both vendor and tool agnostic as DITA's a non-proprietary standard managed by the OASIS consortium. Therefore, content is not locked into a single software or tool.

This guide aims to provide you with a practical introduction to reusing content through DITA's core principles. Readers will acquire foundational knowledge to begin authoring high quality documentation.

Why DITA Matters

For those of you who have also spent time in development, like me…you know the importance of components. Modern web applications are comprised entirely of thereof. They're the building blocks of every sustainable feature because they adhere to reusability. Reuse a button component for instance everywhere there's a button. Same scenario with DITA technical writing.

Component based authoring enables writers to work with small, self-contained items vs massive files. Updates and maintenance efforts are reduced tremendously. In the event of having a topic component for instance used in multiple channels, updating it once will reflect every instance.

All of which emanate from one source file known as a single source of truth (SSOT). Paragraphs, procedural steps, or safety warnings for instance are stored as granular topics in a central repository or CMS. Similar to many technological concepts, reusing elements is achieved via referencing, not copying. Content Referencing (conref) and Key Referencing (keyref) establish links from elements to the master content.

Translation systems become more accurate, and as content is prepared for localization, its easier—leading to cost savings and faster production rates for global updates. Reviewers performing said updates can then focus solely on the actual changes rather than reviewing entire manuals. Content reuse with DITA standardizes output among:

- PDF
- HTML5
- Mobile
- EPUB
- Numerous other formats

DITA matters because we're able to utilize its fundamental concept of Write Once, Use Many. A concept that should undoubtedly be applied to an array of resources—including technical writing. If you encounter the term(s), *XML technical documentation* or *structured authoring example*, it's referring to this system of Write Once, Use many. XML is a format that's widely supported among an innumerable amount of business systems, CMSs, training platforms, and bug trackers.

Topic Types

Before viewing a structured authoring example, we must first understand topic types. This foundational concept simply means content is grouped into types of topics. Think about an informational document and you'll quickly conclude its made of specific building blocks. Together, these blocks make up the document structure.

A webpage for instance has a header, footer, navigation, main content, and perhaps secondary sections such as a sidebar. In a DITA documentation sample, we'd have a:

- Task topic
- Concept topic
- Reference topic

- Perhaps custom topic types

With these rudimentary building blocks, we can create elaborate and diverse documentation. The task topic type's purpose is to explain how to perform a procedure or complete a process. It's purpose is to guide users through a series of ordered steps such as a recipe or assembly. Task topics contain steps and actions to strictly enforce a process such as:

- Install the software
- Log into the system
- Change the battery

The concept topic type answers users' questions about, **"what is this?"**, or **"why should I care?"**. It's purpose is to provide background or introductory information via contextual understanding such as a dictionary definition or encyclopedia entry. This flexible topic type is built around paragraphs and lists. Of which, should never contain actionable steps. Examples of thereof are:

- New System Architecture Overview
- Best Practices for Database Backup
- Error Codes and Meanings
- Hardware Specifications Table

Reference types are for specs or details. They focus on specific items with a purpose of presenting detailed data that's looked up such as a data sheet, specs chart, or quick reference guide.

Topic Based Authoring Example

This XML snippet represents a real world scenario for resetting a password via a *task topic type*. Demonstrating content separation, the tags enforce structure for a process.

- <title> serves as the heading in a PDF, HTML, etc.
- <taskbody> holds all the procedural elements

- <context> optionally provides necessary background prior to starting the steps
- <steps> is the procedural container for sequential elemental actions
- <step> is a single item in an ordered process
- <cmd> represents the command—click, type, etc.
- <unicontrol> indicatively conveys text is a button in the UI
- <info> provides additional detail to an action
- <result> describes the successful outcome

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE task PUBLIC "-//OASIS//DTD DITA Task//EN" "task.dtd">
<task id="reset_password">
    <title>Resetting Your Account Password</title>
    <taskbody>
        <context>
            <p>You can reset your password using the "Forgot
Password" link on the login screen. Ensure you have access to the
email address associated with your account.</p>
        </context>
        <steps>
            <step>
                <cmd>Navigate to the application login page.</
cmd>
            </step>
            <step>
                <cmd>Click the <uicontrol>Forgot Password</
uicontrol> link located beneath the login button.</cmd>
            </step>
            <step>
                <cmd>Enter your registered email address in the
provided field.</cmd>
            </step>
            <step>
                <cmd>Click <uicontrol>Send Reset Link</
uicontrol>.</cmd>
                <info>A password reset link will be sent to your
email address.</info>
            </step>
            <step>
                <cmd>Open the email and click the provided reset
link.</cmd>
            </step>
            <step>
                <cmd>On the Reset Password screen, enter your new
```

```
password in both the <uicontrol>New Password</uicontrol> and
<uicontrol>Confirm Password</uicontrol> fields.</cmd>
            </step>
            <step>
                <cmd>Click <uicontrol>Update Password</uicontrol>
to finalize the change.</cmd>
            </step>
        </steps>
        <result>
            <p>Your password has been successfully updated. You
will be redirected to the login page.</p>
        </result>
    </taskbody>
</task>
```

Now let's examine an XML technical documentation sample for REST APIs as a *concept topic type*—what's a REST API and how does it work?

- <concept> is the root element defining primary information as explanatory
- <title> serves as the main heading in the final output
- <abstract> optionally provides an immediate high-level overview
- <conbody> is the main body wrapper
- <section> indicates a logical grouping of content—a subheading
- <p> is of course the standard paragraph element
- and <li> are used in lists
- <dl> and <dlentry> for present definition lists and entries

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE concept PUBLIC "-//OASIS//DTD DITA Concept//EN"
"concept.dtd">
<concept id="rest_api_overview">
    <title>Understanding RESTful APIs</title>
    <abstract>
        <p>A RESTful API (Representational State Transfer
Application Programming Interface) is an architectural style for
designing networked applications, central to modern web
services.</p>
    </abstract>
    <conbody>
        <section id="principles">
            <title>Core Architectural Principles</title>
            <p>REST relies on a few key constraints, which allow
```

```
services to be highly scalable and maintainable:</p>
            <ul>
                <li>**Client-Server:** Decoupling the user
interface (client) from the data storage (server).</li>
                <li>**Stateless:** Each request from the client
to the server must contain all the information needed to
understand the request.</li>
                <li>**Cacheable:** Responses can be cached to
improve performance.</li>
            </ul>
        </section>
        <section id="methods">
            <title>HTTP Methods and Resource Mapping</title>
            <p>REST uses standard HTTP methods to interact with
resources (e.g., a user record, a product item). These are often
mapped directly to CRUD (Create, Read, Update, Delete)
operations:</p>
            <dl>
                <dlentry>
                    <dt>GET</dt>
                    <dd>Retrieves data (Read).</dd>
                </dlentry>
                <dlentry>
                    <dt>POST</dt>
                    <dd>Creates new data (Create).</dd>
                </dlentry>
                <dlentry>
                    <dt>PUT/PATCH</dt>
                    <dd>Modifies existing data (Update).</dd>
                </ddlentry>
                <dlentry>
                    <dt>DELETE</dt>
                    <dd>Removes data (Delete).</dd>
                </dlentry>
            </dl>
        </section>
    </conbody>
</concept>
```

This *reference topic type* specifies an API endpoint's parameters signifying content reuse with DITA. Applicable shall the documentation cover integration with multiple languages or frameworks.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE reference PUBLIC "-//OASIS//DTD DITA Reference//EN"
"reference.dtd">
<reference id="user_endpoint_parameters">
```

```
    <title>/users Endpoint Parameters</title>
    <refbody>
        <section>
            <title>Query Parameters</title>
            <properties>
                <proptable>
                    <prophead>
                        <proptypehd>Parameter</proptypehd>
                        <propvaluehd>Data Type</propvaluehd>
                        <propdeschd>Description</propdeschd>
                    </prophead>
                    <properties>
                        <proptype>id</proptype>
                        <propvalue>Integer</propvalue>
                        <propdesc>The unique user identifier
(optional).</propdesc>
                    </properties>
                    <properties>
                        <proptype>status</proptype>
                        <propvalue>String</propvalue>
                        <propdesc>Filter by user status: 'active'
or 'inactive'. Default is 'active'.</propdesc>
                    </properties>
                </proptable>
            </properties>
        </section>
        <refsyn>
            <title>Example Syntax</title>
            <codeblock>GET /users?status=active</codeblock>
        </refsyn>
    </refbody>
</reference>
```

- <reference> is the root element in this case—defines the file's primary type
- <title> serves as the main heading (e.g., "/users Endpoint Parameters")
- <refbody> wraps the content
- <section> groups the content
- <properties> contains property value tables
- <proptable> is then used within <properties>
- Just like <thead> in HTML, we have <prophead> to define the header row
- <proptype>, <propvalue>, <propdesc> are table column elements used to define type, value, and description respectively
- <refsyn> alludes to sectional syntax/ shows the code snippet
- <codeblock> displays the code (`GET /users?status=active`)

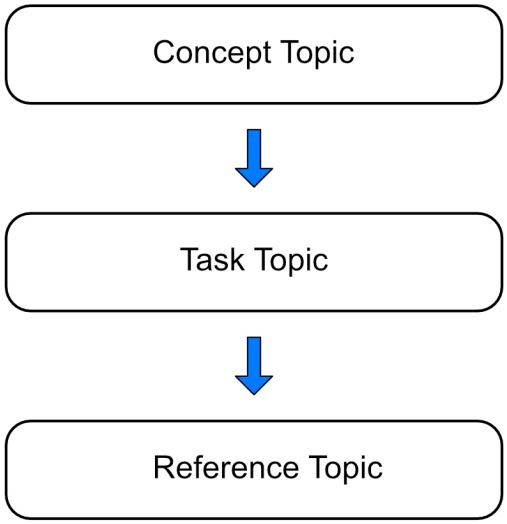
Do take note however of the DOCTYPE in each of these code snippets:

```
<!DOCTYPE reference PUBLIC "-//OASIS//DTD DITA Reference//EN"
   "reference.dtd">
```

We're saying, the declaration identifies this document as a DITA [Reference] topic which enforces the OASIS-defined structure and elemental rules for reference content. OASIS is the nonprofit, international standards organization that creates and maintains open technology standards used across the web and enterprise software.

Topic Relationship Diagram

This flow diagram illustrates how DITA topic types—concept, task, and reference work together in a structured authoring workflow. Concept topics introduce the subject, task topics provide procedural steps, and reference topics supply supporting technical detail.



DITA Map Example and Content Organization

In order to produce an output from a variety of content and components, authors of thereof must first create a new file. One that ends in .ditamap. This file serves as the blueprint—*not the content itself*. The DITA Map specifies which links to include and the order in which they appear. Acting as the output Table of Contents (TOC), it establishes relationships between topics.

A nested topic for instance is a child. DITA map contents are translated into chapters, sections, etc. In software and rudimentary web applications, there's always one file that's referenced when all the code is compiled and the actual project is built. Whether its XML technical documentation or complex web applications, there's always some type of config file powering the output process. In this case, it's the DITA map (example shown below).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE map PUBLIC "-//OASIS//DTD DITA Map//EN" "map.dtd">
<map>
    <title>CloudConnect REST API Guide</title>

    <topicref href="concepts/api_overview.dita"/>
    <topicref href="concepts/authentication_modes.dita"/>

    <topicref href="concepts/getting_started_intro.dita">
        <topicref href="tasks/generate_api_key.dita"/>
        <topicref href="tasks/first_api_call.dita"/>
    </topicref>

    <topicref href="reference/endpoint_users.dita"/>
    <topicref href="reference/error_codes.dita"/>
</map>
```

- <map> is the root element
- <title> sets the documentation title
- <topicref> links to a topic (.dita or .xml)
    - a nested <topicref> creates hierarchy, creating sub-sections

Multiple DITA maps can reference the same topics to create different output channels for perhaps, basic and advanced. Version 1.1 and version 2.1, how to use with a CMS, UI framework, etc. Or create one Manual.ditamap file containing:

```
<topicref href="intro.dita"/>
<topicref href="installation.dita"/>
<topicref href="parameters.dita"/>
```

Maps can be created rather easily via drag and drop editors found in Oxygen

XML, XMetal, or Adobe FrameMaker. These tools enable users to utilize their DITA files to build tree structures while XML is automatically written. Linking is done via basic HTML href attributes that simply point to the files location.

Authors create user experiences via DITA maps. Deciphering whether one section should be read before another. Organize content and location of the actual files however you'd like, but the Map is where reader organization occurs.

XML Technical Documentation Best Practices

We've already touched on reusability several times in this document—directly and indirectly. But the underlying concept here is to reuse topics in multiple outputs, for numerous reasons, or even in special use cases.

Reuse topics in fragments and subsections or as standalone chapters in other cases. In the event of an automobile manual for instance, the manufacturer may produce six models each with varying degrees of dashboard features, power, and safety. One manual may be confusing as not every model is equipped with the same features—so numerous manuals area created by reusing topics.

Specifically, content references can be utilized by pulling a unique piece of copy (single step, paragraph, or a table) into a different topic. *<step id="safety-step">* can be pulled in via the *conref* attribute. Further reusability can be accomplished by key references (keyref attribute). Let's clarify this in the following example with an API call document.

- Begin by defining variables in the Map and Library file
- Use both variables in the Delete User task
- `<step conref="shared_steps..."/>` gets replaced with the Library file contents
- `<ph keyref="api_version"/>` acts as a placeholder for the version number
  - Make version changes in the Map file

api_project.ditamap

```
<map>
  <keydef keys="api_version">
    <topicmeta>
      <keywords><keyword>v2.4.0</keyword></keywords>
    </topicmeta>
```

```
  </keydef>
</map>
```

shared_steps.dita

```
<task id="library">
  <title>Shared Steps</title>
  <taskbody>
    <steps>
      <step id="auth_step">
        <cmd>Header: Set <uicontrol>Authorization</uicontrol> to
your Bearer Token.</cmd>
      </step>
    </steps>
  </taskbody>
</task>
```

delete_user.dita

```
<task id="delete_user">
    <title>Deleting a User</title>
    <taskbody>
        <context>
            <p>This endpoint is only available in API version
               <ph keyref="api_version"/>.
            </p>
        </context>
        <steps>
            <step conref="shared_steps.dita#library/auth_step"/>

            <step>
                <cmd>Send a DELETE request to /users/{id}.</cmd>
            </step>
        </steps>
    </taskbody>
</task>
```

Metadata and Conditional Attributes

In the same way details are extracted from a photo, DITA uses metadata and conditional attributes to describe what content is and who views it. Metadata is data about data. Its information that's primarily used for categorization purposes which is why it's never viewed on a page. Let's illustrate metadata.

- <prolog> is the metadata container
- <keywords> and <metadata> help a docs website find the right topic
- Product and audience attributes help show parts of the file for select

audiences

- This API doc has free and paid versions with paid features hidden to free users
- audience="administrator" needs direction during the publishing process which occurs in the DITAVAL file—a tiny XML file created prior to publishing which would exclude content where product="pro-version". Swapping the DITAVAL file enables authors to simply generate alternate versions. A sample use case is a *Standard User Guide* and an *Admin Guide.* An ideal topic based authoring example.

```
<task id="api_setup">
    <title>Setting Up Your API Key</title>
    <prolog>
        <metadata>
            <keywords>
                <keyword>Authentication</keyword>
                <keyword>Security</keyword>
            </keywords>
        </metadata>
    </prolog>
    <taskbody>
        <steps>
            <step>
                <cmd>Log in to your dashboard.</cmd>
            </step>
            <step audience="administrator" product="pro_version">
                <cmd>Enable "High-Priority Throughput" in the
settings.</cmd>
            </step>
            <step>
                <cmd>Copy your API Key to your clipboard.</cmd>
            </step>
        </steps>
    </taskbody>
</task>
```

Versioning/ Review Workflow

Versioning and review workflows are significantly different than simply tracking changes. Because content is housed in small reusable chunks, workflow focuses on managing the lifecycle of these objects. Versioning ensures we can maintain documentation as the software evolves. Thus, maintaining

documentation for an older version while working on a new one.

Since we're versioning hundreds of small components, we need to harness the inbuilt version history in Topics and Map files. Of which can be accomplished either via a Component CMS or a version control system such as Git. Via Git, branching and merging occurs in relation to the Main Branch—containing the latest documentation.

Release branches occur upon new software version development—allowing users to update the "Installation" task for v2.0 without changing the live v1.0. Snapshots are a baseline, frozen, version of a DITA Map and all its referenced topics at a specific point in time. And because auditing is an inbuilt feature in version control systems, we can see who changed a step or tag, and why.

No DITA XML tutorial would be complete if it failed to mention the review process—a collaborative and asynchronous phase. In the drafting stage, topics are assigned a status of *Draft*, *In Review*, or *Approved*. Topics In Review are often locked to prevent writers from making changes while Subject Matter Experts (SME) review it.

SMEs are not looking at the XML tags—but a simplified and rendered version of the content where comments or suggestions can be added. This is achieved via modern DITA tools with web based interfaces for reviewers. In avoidance of emailing Word attachments back and forth, multiple reviewers can comment on the same DITA topic simultaneously.

Topic reviews focus on technical accuracy of a single component while Map reviews focus on the flow, hierarchy, and so called story of the entire document. An automated validation process is then conducted looking for XML errors.

Skills Demonstrated

This documentation sample highlights my ability to create structured, reusable, and scalable content using DITA/XML. It demonstrates proficiency in topic-based authoring, information architecture, content engineering, and visual communication. The work reflects enterprise-level documentation practices, including modular design, version management, metadata usage, and multi-channel publishing workflows—all expressed through clean, readable XML and consistent visual styling.

- Developed a structured DITA XML example using concept, task, and reference topics.
- Applied topic-based authoring principles for clarity and modular reuse.
- Built a DITA map to define hierarchy and support multi-format publishing.
- Utilized metadata and conditional attributes for versioning and audience variants.
- Demonstrated content reuse through conref/keyref implementation.
- Designed scalable information architecture to organize and maintain complex docs.
- Produced clean, standards-aligned XML following structured authoring best practices.
- Applied content engineering methods, including modularity, governance, and version control.
- Created a visual diagram to illustrate topic relationships and publishing flow.
- Maintained consistent branding and readability using Web Dev Unlimited's design system.