Arbortext Authoring Guide for DITA

**E53075–01**

ORACLE

Arbortext Authoring Guide for DITA,

E53075–01

# Contents

## 6 Creating and Managing Publications in Architect



## Part II The Arbortext Editor

## 7 Getting Started with Arbortext Editor



## 8 Customizing the Arbortext Editor Interface



## 9 Arbortext Editor Fundamentals

# 1

# Introduction

This is your guide to authoring DITA documents with the Arbortext XML editor.

To write documentation for Oracle Database, you must be familiar with the following tools and standards:

- XML
- DITA
- Arbortext Editor
- SDL LiveContent Architect

## About XML

Extensible Markup Language (XML) is a metalanguage that enables users to define their own customized markup languages. A markup language enables authors to enclose content in elements (or **tags**) that provide metadata characterizing the enclosed content. XML tags contain **attributes** that provide additional properties characterizing the content.

A markup language enables content to be read and interpreted by computers. For example, for data transfer between computers, each item of data might be enclosed in tags such as <product_id>, <customer_id>, or <order_number>. In documents, XML tags can assign meaning to document parts, such as paragraphs, lists, and tables, enabling documents to have a rigorous structure for machine processing. Finally, an XML-based markup language enables computers to perform transformations on the data. The most well-known transformation is the rendering of a structured document into HTML.

XML is defined in a specification produced by the World Wide Web Consortium (W3C), and in several other related specifications, all free and open standards.

XML documents can contain row-oriented data, similar to records in a database, or non-row-oriented data, such as the content of an article, white paper, or book. An XML document is constructed of a sequence of **elements**. An element consists of an opening tag, followed by content, followed by a closing tag. The content can be text or other **child** elements. The following is a simple element:

```
<ordernumber>89393</ordernumber>
```

A tag is text enclosed in angle brackets. In the previous example, <ordernumber> is the opening tag and </ordernumber> is the closing tag. Closing tags are indicated by a slash after the initial angle bracket.

The following example shows an element (`order`) with the child elements `order`, `ordernumber`, `orderdate`, and `customer_id`:

```
<order>
    <ordernumber>89393</ordernumber>
    <orderdate>May 11 2013</orderdate>
    <customer_id>233</customer_id>
</order>
```

Elements can have zero, one, or more **attributes**. Attributes define properties for elements, Attributes in a parent element can characterize sibling elements. Markup language designers can choose to characterize the nature of information contained within an element, or use attributes in an element to characterize the same element differently, by setting different attribute values . The following example is a variation of the previous example, this time using the `id` attribute of the `order` element to define the order number:

```
<order id="89393">
    <orderdate>May 11 2013</orderdate>
    <customer_id>233</customer_id>
</order>
```

The following example shows how attributes of the `table` element define the table properties:

```
<table width="fullpage" border_color="black">
    <row>...</row>
    <row>...</row>
</table>
```

Elements can be **empty**, meaning that they have no content. However, an empty element may have attributes. An empty element is indicated by a slash before the ending angle bracket. One example of a well-known empty element is the line-break element in XHTML:

```
<br/>
```

**Note:** XHTML is HTML that follows the rules of XML.

# About DITA

Darwin Information Typing Architecture (DITA) is an XML-based architecture designed for writing, organizing, and linking topic-based content. DITA maps organize topics according to their content model to create different kinds of documents.

In DITA, the basic unit of information is the topic. There are three kinds of DITA topics: concept, task, and reference. An InfoDev DITA customization adds a fourth topic type: the orientation topic. Each topic must stand on its own and must be reusable in multiple contexts, such as different documents or different output formats.

Each DITA topic must follow a defined structure, and the elements that form the topic must be placed in the correct locations. The element structure of all topics must adhere to a content model. This content model dictates the relationships between the DITA elements in the topic (such as parent/child) as well as the allowed attributes and values for each kind of tag. InfoDev DITA templates enable you to create different types of topics that have the correct structure.

**See Also:**

InfoDev DITA Information Types
: InfoDev DITA information types, also called topic types, are the building blocks for documentation. The types are task, concept, reference, and orientation.

Topic-Based Writing Concepts
: Writers separate content into topics of the following types: concept, task, reference, and orientation. Topic-based writing enables topic reuse and promotes consistency.

Creating Maps
: Use maps to logically organize objects, including other maps and topics, in a publication.

Formatting Content with Semantic Tags
: You use semantic tags to mark up inline text (phrases, names, file paths, and so on) and impart both meaning and implied formatting to the marked up text. Semantic tags are preferred over typographic tags.

Creating a Publication
: You create a publication to assemble the content in a document for output.

# About SDL LiveContent Architect

SDL LiveContent Architect (Architect) is a content management system (CMS) for planning, creating, reviewing, publishing, and releasing publications.

Architect provides a central repository with an Oracle database backend that stores publications and the objects that comprise publications, such as topics and images. Writers, managers, architects, graphic designers, and others use Architect to create, manage, modify, and review these objects in a collaborative environment.

Architect provides the following benefits:

- Supports DITA-based content
- Integration with Arbortext Editor
- Storage in a CMS
- Collaborative authoring
- Reuse and sharing of content
- Version control of all objects
- Unique identification of each object with a globally unique ID (GUID)
- Support for complex conditional text
- Object lifecycle management through workflows
- Robust search capabilities
- Support for variable libraries

**See Also:**

Architecture of the SDL LiveContent Architect System
: SDL LiveContent Architect (Architect) is a content management system (CMS) that consists of an Oracle database repository, the SDL Application Server , batch servers, and two clients: Publication Manager and the Web Client.

## About Arbortext Editor

Information Development (InfoDev) has selected Arbortext Editor as the authoring tool to create and edit documents in DocBook and other XML languages. Arbortext Editor has an interface that facilitates creating structured documents, and fully supports topic-based writing.

Arbortext Editor includes the following features:

- XML authoring and editing

  With Arbortext Editor, you can create and edit documents, edit tables in table or tagged mode, create and edit equations, import and export documents to and from other formats, personalize content through profiling, and perform several other tasks.

- Native DITA support

  Arbortext Editor supports the OASIS Darwin Information Typing Architecture (DITA) standard by providing customized configurations for editing under this architecture.

- Arbortext Command Language (ACL)

  ACL is a scripting and customization tool that enables you to write simple macros, which can increase productivity, modify key mappings in specific windows, and change the behavior of editor events.

- SDL LiveContent Architect compatibility

  Arbortext Editor supports compatibility with SDL LiveContent Architect through the Authoring Bridge module. The Authoring Bridge embeds the SDLLiveContent menu into the default Arbortext Editor window, it is used to access the repository and perform the most recurrent operations such as checking objects in and out of the repository, adding variables, conditionalizing elements, and so on.

**See Also:**

Getting Started with Arbortext Editor
: Installing Arbortext Editor is the first step to set up your authoring environment. After you install Arbortext Editor, configure it to access the Information Development custom document types to use the InfoDev DITA element specializations.

## About DocBuilder

DocBuilder is the mechanism with which Oracle documentation will be produced, similar to the Review Site Builder has been doing up until now.

DocBuilder produces HTML, PDF, and will publish content to Oracle Review from the source files you create using the SDL LiveContent Architect and Arbortext.

# About the CCMS and Arbortext Authoring Portal

The CCMS and Arbortext Authoring Portal contains resources for using XML, DocBook, DITA, SDL LiveContent Architect, Arbortext Editor, and DocBuilder.

The CCMS and Arbortext Authoring Portal's resources include documentation, training materials, videos, how-to articles, and online support forum. Post questions about CCMS and Arbortext Authoring to the support forum, and experts will answer your questions.

**See Also:**

CCMS and Arbortext Authoring Portal

CCMS and Arbortext Authoring Support Forum

TEST CCMS and Arbortext Authoring Support Forum

# Part I

## SDL LiveContent Architect

# 2

# Overview of SDL LiveContent Architect

SDL LiveContent Architect (Architect) is a content management system (CMS) for creating, storing, and managing publications and the objects that comprise publications.

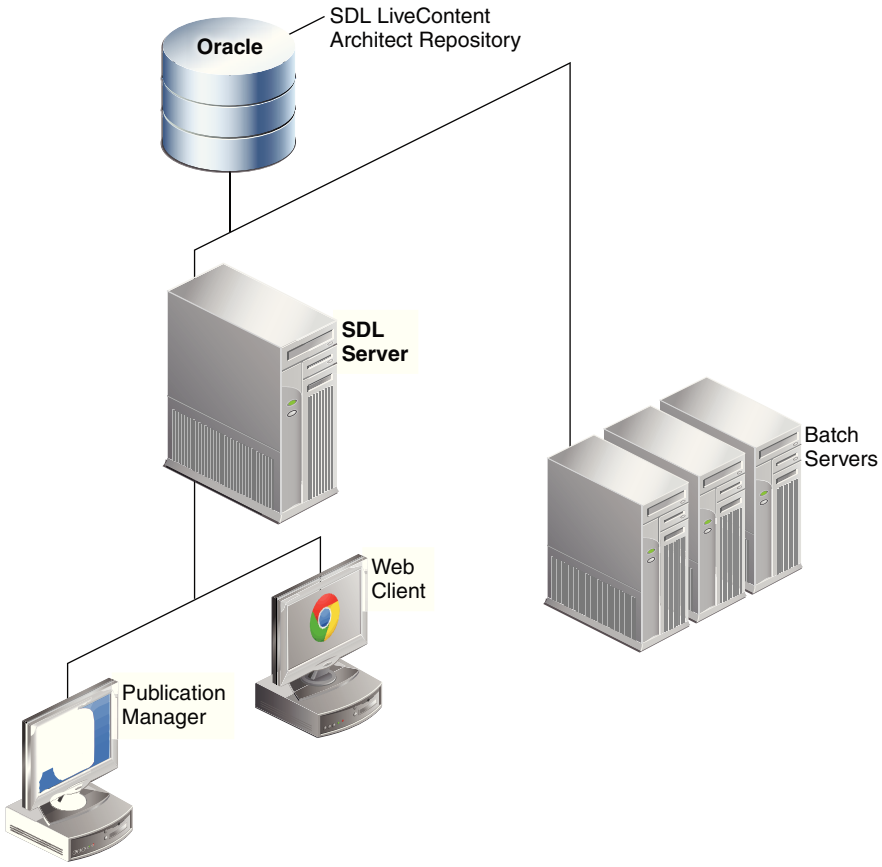
Architect provides the architecture and features that enable a collaborative, topic-based writing process.

## Architecture of the SDL LiveContent Architect System

SDL LiveContent Architect (Architect) is a content management system (CMS) that consists of an Oracle database repository, the SDL Application Server , batch servers, and two clients: Publication Manager and the Web Client.

This diagram shows the components of Architect:

**SDL LiveContent Architect**



| Component | Description |
|---|---|
| The SDL LiveContent Architect Repository | The repository uses an Oracle database (running on Exadata!) to store all of the objects in the CMS, including XML files, image files, and output such as PDF and XHTML files. |
| SDL Application Server | This server contains the application logic for the CMS. It manages the content in the repository and enables clients to consume its content. |
| Batch servers | A number of batch servers dedicated to publishing output. |
| Publication Manager | Publication Manager is a desktop client used by writers and architects to create and manage publications in the repository. In Publication Manager, writers access their **inboxes**, which contain objects, such as topics, that require their attention. Writers, architects, and others can also use Publication Manager to search the repository for specific objects. |
| Web Client | The Web Client provides access to the repository through a web browser. It provides tools that are used by both writers and administrators. |

In addition, an **Authoring Bridge** for Arbortext Editor provides access to the repository through a customized menu. This menu enables writers to perform several CMS tasks from Arbortext Editor, such as checking topics out and in.

## About Publication Manager

Publication Manager is a desktop client used by writers and architects to create and manage publications in the repository. It is your main tool for managing publications and the components that comprise publications.

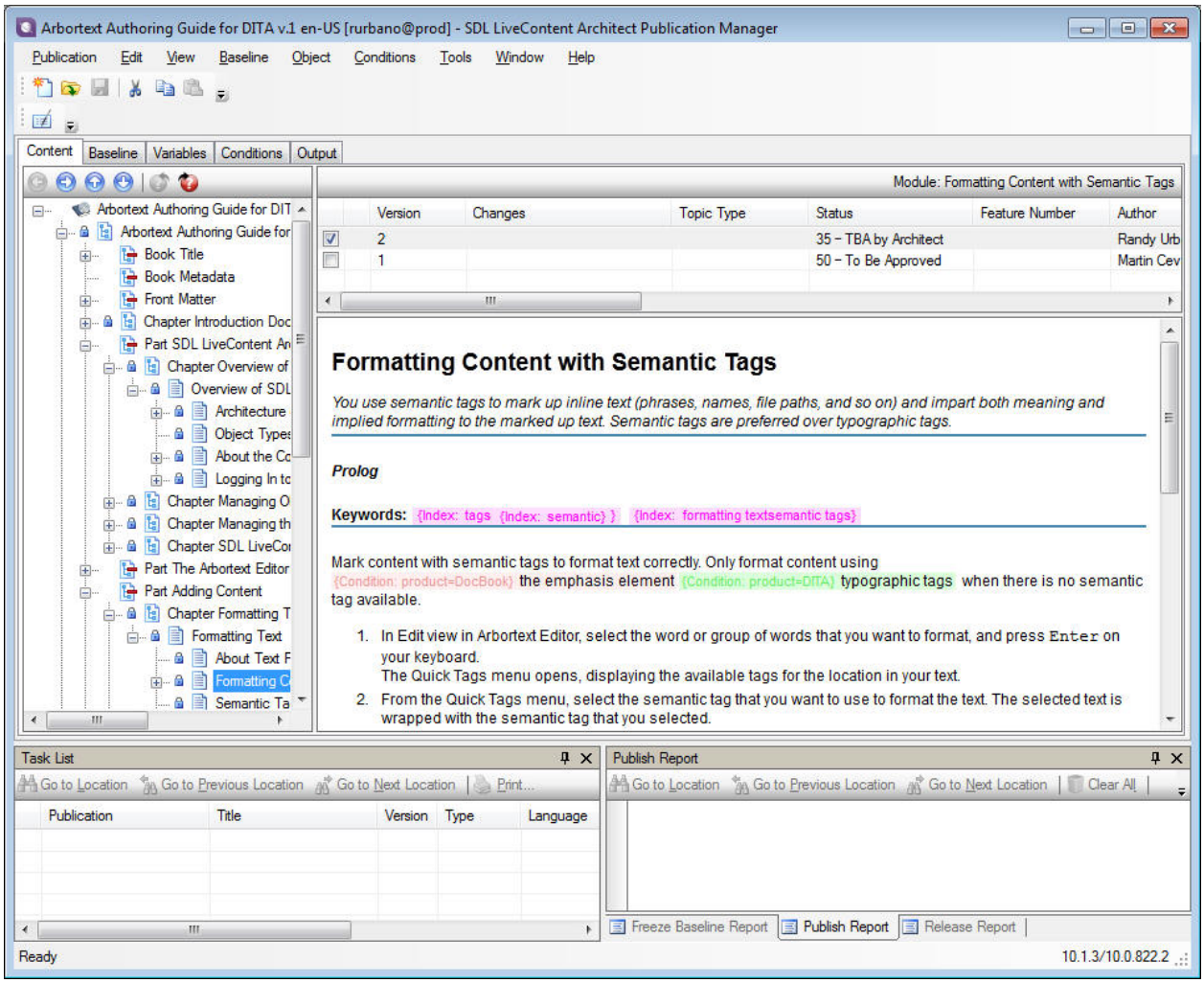
A publication is a CMS-specific object that defines a document so that it can be published. It does so by associating a **master document** (ditamap), a **baseline**, the **publication context** (variables and conditions), and multiple **output format** objects.

You use Publication Manager to complete many of the tasks described in this guide. For example, in Publication Manager, you can open a publication to modify its master document and submaps, and you can select the versions of individual objects to use in publications.

You can also open the Browse Repository window from Publication Manager. Using the Browse Repository window, writers access their **inboxes**, which contain objects, such as topics, that require their attention. Writers, architects, and others can also use the Browse Repository dialog box to search the repository for specific objects.

The following figure shows the Publication Manager interface displaying an open publication.

***Figure 2-1 Publication Manager Interface***




**See Also:**

Logging In to Publication Manager

Publication Manager is able to connect to the SDL server using different credentials. You can create different accounts to store credentials associated with different users. Furthermore, you can set a particular account as the default for the client so it is automatically used every time it makes a request to the server.

The Browse Repository Dialog Box

The Browse Repository dialog box provides a graphical user interface to the repository. It enables you to complete several tasks, such as creating folders and objects, previewing objects, and checking objects out.

## SDL LiveContent Architect Features

Architects includes a number of features that enable it to accomplish its authoring, reviewing, and publishing tasks.

| Feature | Description |
|---|---|
| Version control | Architect manages objects (maps, topics, illustrations, and unstructured documents) |

| Feature | Description |
|---|---|
| | and all its versions in a single repository. The system provides the standard versioning and branching features. It also creates a new revision when an object is modified enabling you to trace changes. |
| Link management | In Architect, a link pointing to an identifier represents all relationships between objects, this enables to move them around the folder hierarchy in the repository while guaranteeing document integrity. Architect doesn't allow deleting an object that is being referenced some other (for example, a map that includes a topic or a topic that references a graphic), this prevents broken links in your web sites or missing pages in your published output. |
| Workflow management | Architect supports lifecycle management. Each objects has a certain workflow that tracks the object from creation to when a publication is released. Each type of object has a specific workflow and different stages of the workflow involve different members of the content team. |
| Conditional content | SDL LiveContent Architect's Condition Manager feature enables information architects to define and manage a list of valid condition names and condition values. Condition Builder provides a graphical interface that authors use to select among the valid conditions and apply them to specific content. |
| Variables | The Architect interface facilitates managing and implementing variables in a publication. Authors may insert references to previously defined variables instead of hard coded XML content. These variables are resolved during processing time and the appropriate output is included in the publication. For example, instead of hard coding the product name throughout the XML content, a variable can be used. This makes it possible to immediately update a publication with a new product name by simply changing the variable definition. |
| Search and taxonomies | There are two ways of locating content in the repository: search and navigation. Authors can search for content using metadata and/or full text search. Also, users can navigate the repository's folder structure to locate content objects. |
| Reporting capabilities | Architect provides different reporting capabilities:<br>• Publication report: It helps to identify the differences between two versions of a publication, including for example which |

| Feature | Description |
|---|---|
| | modules were added and which modules were changed. It enables you to export the publication out of the repository.<br>• Workflow report: it provides a quick overview of the status of a publication indicating which components are missing, which components need to be approved, which components need to be translated.<br>• Where used report: it gives an overview of where a component is used within publication or at the entire repository level.<br>All reports can be downloaded in Comma Separated Values (CSV) file format. |
| Publishing | Authoring all the source documents in DITA enables Architect to produce multiple output formats from a single source. Besides the traditional formats (PDF, XHTML, CHM, etc.) integrating the repository with different publishing engines helps to deliver content in new, more dynamic formats, such as customized webpages. |

# Object Types Managed by SDL LiveContent Architect

All of the components that make up a document are objects in the SDL LiveContent Architect (Architect) repository.

***Table 2-1 Object Types***

| Architect Term | Equivalent DITA Term | Description |
|---|---|---|
| Publication | (None) | The definition of a document by associating a master document, a baseline, the publication context (variables and conditions), and several output formats. |
| Module | Topic | An XML document that stands on its own to cover a single topic. InfoDev DITA supports the following topic types: concept, task, reference, and orientation. |
| Master document | Ditamap | A document that collects and organizes references to DITA topics to indicate the relationships among the topics. It can be used to identify the topics you want to include in a deliverable and to create tables of contents and related links for the information.<br>In the CMS, a master document is similar to the table of contents for a |

| Architect Term | Equivalent DITA Term | Description |
|---|---|---|
| | | publication. A master document can include submaps that contain the structure for a part of the publication, such as one chapter. |
| Library | (None) | Special topics that store a set of related variable definitions or elements. The variables can then be referenced from multiple topics, and the elements can be referenced from multiple topics as content references (conrefs). |
| Graphic | Graphic that is referenced by an image or figure element | An image that can be inserted into one or more topics |
| Template | (None) | Non-XML files such as PDF files, spreadsheets, and other documents |
| Folder | (None) | A logical construct that organizes objects. A folder only can contain objects of the same kind. |

**See Also:**

Object Types Managed by SDL LiveContent Architect
: All of the components that make up a document are objects in the SDL LiveContent Architect (Architect) repository.

Viewing Objects in the Repository Browser Dialog Box
: You can preview an object in Publication Manager without going through the process of checking it out. The objects contents appear in a new pane of the repository browser, they can bee seen in HTML or XML, but cannot be modified unless the object is checked out.

Checking Out Objects
: To edit objects locally, you must first check them out of the repository. When you check an object out of the repository, you lock the object, preventing someone else from making changes to the same object.

Metadata for Objects
: SDL LiveContent Architect tracks and maintains metadata for each object in the repository. The metadata contains relevant information about each object. It is used to describe each object in the repository, to search for objects, and to track each object through its workflow.

# About the Collaborative Writing Process with DITA and Architect

Content teams decide which topics are required to document a feature, which writers will write specific topics, and which topics will be reused in different publications.

Using DITA and LiveContent Architect, writers, functional area owners, architects, editors, and product managers collaborate to document product features efficiently and fully.

## About Content Teams

The content team works together to determine the topics that must be written to document a specific product feature.

The leader of the team is the functional area owner for the feature in the content management system (CMS), which is roughly equivalent to the primary documentation mapping writer in ST Project. When the content team is finished determining the topics that are required to document a feature, it determines which writer will write each topic.

The functional area owner optionally can create a feature map, which serves as an outline for the feature. The functional area owner creates empty topics for the feature and adds these topics to the feature map. An architect must approve the feature map. After approval, the functional area owner assigns the topics to writers in the CMS. The feature map usually is not published for any deliverable to customers, but it might be published for review by a product manager.

The content team also determines which topics can be reused in different publications. The content team might also determine whether any libraries of variables are required and if conditional text is needed. The team also determines whether some content can be placed in a library to be referenced by multiple topics in content references (conrefs).

**See Also:**

Reusing Small Chunks of Content with Conrefs
: A content reference (conref) is a reference to a reusable element in a library or topic. Conrefs enable writers to share small chunks of content in multiple topics.

About Reusing Topics
: Topic reuse improves efficiency and documentation quality. Collaboration and planning result in high levels of topic reuse.

Creating Feature Maps
: Create a map in the repository, and add the topics that will cover a specific feature to the map.

Conditionalizing Content Using Variables and Variable Libraries
: Implementing variables in topics provides a mean of reusing them in different publications with minimum changes involved. For example, when writing two documents referring to related products that differ only in small chunks of content (such as product name or screenshots) variables allow to generate different publications from the same DITA source.

Conditionalizing Content Using Conditions
: Conditions enable you to mark a DITA element, topic, or submap as conditionalized. Conditionalized content is either included or excluded from a publication, depending on publication settings.

ADMIN

A Comparison of Simple and Complex Materialized Views

## About Reusing Topics

Topic reuse improves efficiency and documentation quality. Collaboration and planning result in high levels of topic reuse.

One of the goals of writing in DITA and using a content management system (CMS) is high levels of topic reuse. Topic reuse provides the following benefits:

- **Efficiency:** A topic is created once and used where it is appropriate instead of multiple writers covering the same content in different publications. Reuse eliminates duplication of effort in both topic creation and future topic maintenance. When updates to the content are needed because of product changes or documentation bugs, it is easier to update a single topic than it is to update multiple topics.
- **Quality:** When content is covered in a single topic instead of multiple topics, writers and reviewers can focus on the single topic to improve its quality. Also, when multiple topics cover the same content, the topics tend to diverge over time as different writers update the content of different topics. By ensuring that the content is covered in a single topic, inconsistency is avoided.

Planning and collaboration are required for a high level of topic reuse. Ad hoc reuse, which involves searching the repository for topics that can be reused with no planning, typically does not result in a high level of reuse.

The following strategies increase the reuse of topics:

- Content teams must analyze the content required for new features and identify new content that can be reused. For example, the content team might identify a concept topic that is required in several different publications.
- Writers must assume that topics will be read out of context and write topics that can stand on their own. Topic-based writing is key for reusability.
- Writers, architects, and editors must identify duplication in existing topics, and writers must consolidate this content into a single topic.

Writers can also reuse chunks of content with content references (conrefs). A conref references an element and places the element's content in a topic. For example, a content team might identify a subtask related to a new feature that must be repeated in several larger tasks.

**See Also:**

Reusing Small Chunks of Content with Conrefs
: A content reference (conref) is a reference to a reusable element in a library or topic. Conrefs enable writers to share small chunks of content in multiple topics.

About Content Teams
: The content team works together to determine the topics that must be written to document a specific product feature.

Topic-Based Writing Concepts
: Writers separate content into topics of the following types: concept, task, reference, and orientation. Topic-based writing enables topic reuse and promotes consistency.

## Logging In to Publication Manager

Publication Manager is able to connect to the SDL server using different credentials. You can create different accounts to store credentials associated with different users. Furthermore, you can set a particular account as the default for the client so it is automatically used every time it makes a request to the server.

1. From the **Tools** menu, select the **Accounts** option.
2. Click the **Add** button.
3. In the Add Account dialog box, enter the name you want to give to the new account in the **Account name** field.

> **Note:** Follow the *user@host* convention when assigning account names. For example, John@Production.

4. In the **SDL LiveContent Architect web service** field, enter one of the following URLs depending on the server that you want the new account to connect:

| | |
|---|---|
| Production server | `https://dadvip0032.us.oracle.com/InfoShareWS/` |
| Development server | `https://dadvip0015.us.oracle.com/InfoShareWS/` |

> **Note:** Login to the development server only for training purposes. Always use your production account when writing official documentation.

5. Click **Next**.
6. In the **User name** and **Password** fields, enter your account credentials.

> **Note:** Selecting the **Remember password** option avoids repeatedly entering your credentials every time Publication Manager makes a request to the SDL server.

7. Click **Next**.

   Your Publication Manager instance synchronizes with the server by downloading the necessary files.

8. In the Synchronizer dialog box, click **Close** after the synchronization is finished.

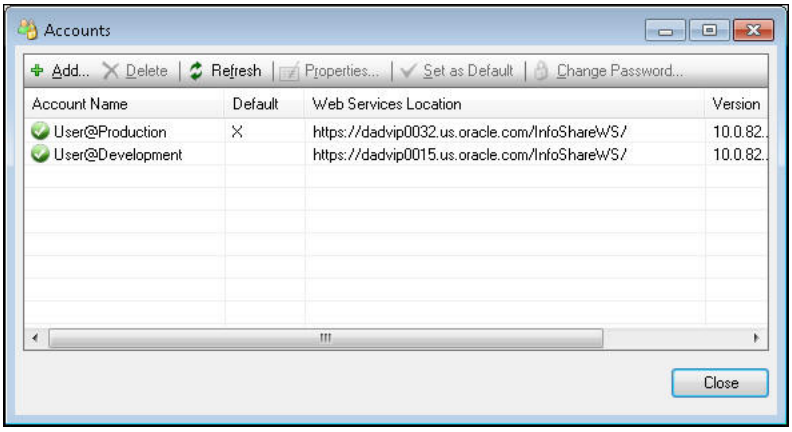
   A validation successful message is shown in the Add Account dialog box.

9. Click the **Finish** button.

After you account is created, it is listed in the Accounts dialog box. When you create more than one account in your Publication Manager client, you can set one of them as the default account so it is automatically used to log in to the repository.

Figure 2-2 shows two accounts used by Publication Manager to log into different servers. The `User@Production` account is used by default every time a request action, like opening a publication, is performed.

***Figure 2-2 The Accounts Dialog Box Displaying Different Accounts***




**See Also:**

About Publication Manager
: Publication Manager is a desktop client used by writers and architects to create and manage publications in the repository. It is your main tool for managing publications and the components that comprise publications.

# 3

# Managing Objects and Folders in SDL LiveContent Architect

Use the Browse Repository dialog box to manage objects and folders in SDL LiveContent Architect, search objects, and access your inbox.

## The Browse Repository Dialog Box

The Browse Repository dialog box provides a graphical user interface to the repository. It enables you to complete several tasks, such as creating folders and objects, previewing objects, and checking objects out.

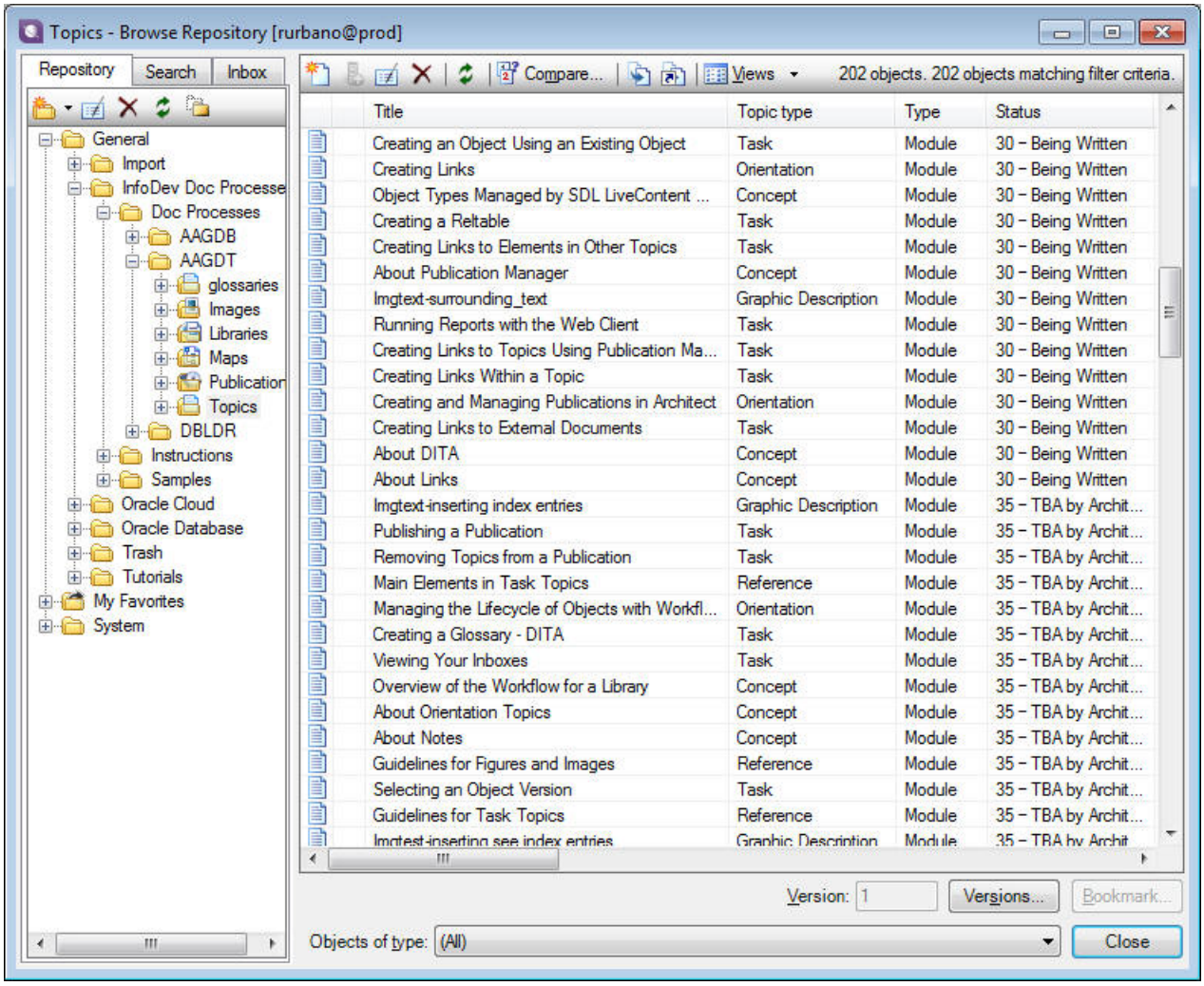
You access the Browse Repository dialog box by selecting **Browse Repository** from the **Tools** menu in Publication Manager. The Browse Repository interfaces is divided in panes. The left pane includes the following tabs:

- The **Repository** tab provides a tree-view of the folders in the repository. You can expand and contract folders to view the folder hierarchy.
- The **Search** tab enables you to search the repository for specific objects.
- The **Inbox** tab provides access to your inboxes, which contain objects that require your attention.

In its right pane, the Browse Repository dialog box shows objects in the repository, metadata related to the displayed objects, and graphical user interface controls that enable you to complete tasks.

> **Tip:** You can filter the type the objects displayed in the right pane by using the **Objects of type** drop-down list at the bottom of the right pane.

***Figure 3-1 The Browse Repository Dialog Box***




### Object Tasks Available from the Browse Repository Dialog Box

Here are some of the tasks you can complete using the dialog box:

- Create and delete folders and objects
- View and change the properties of a selected object
- Preview a selected object in either HTML or XML
- Check out an object
- Import graphics
- Move objects from one folder to another

### Specializations of the Browse Repository Dialog Box

Similar dialog boxes appear while performing other tasks in publication manager such as the following:

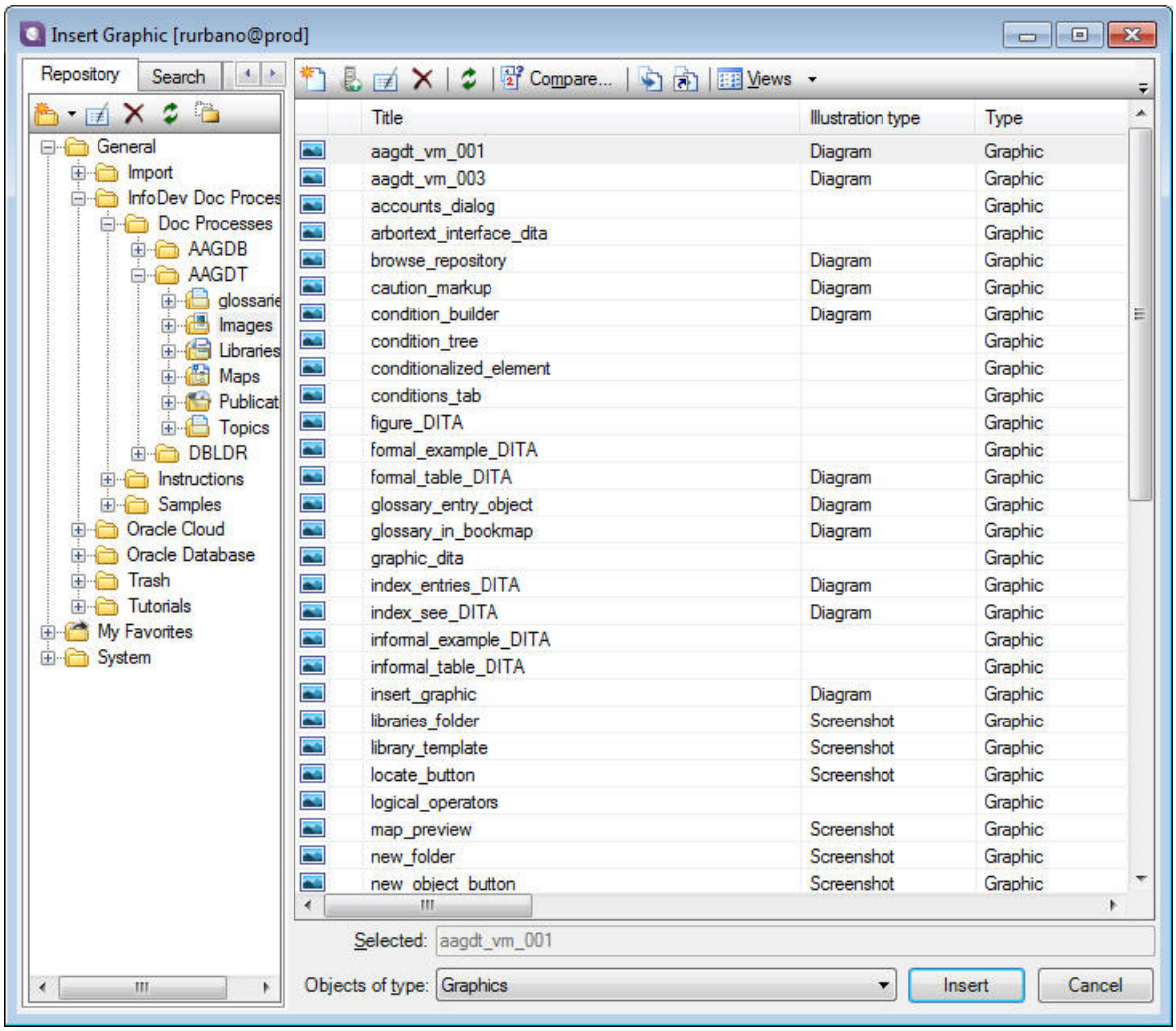
- The Insert Link dialog box appears when you insert a topic reference (topicref) in a map or a topic.

- The Insert Hyperlink dialog box appears when you insert an xref in a topic.
- The Insert Graphic dialog box appears when you insert a graphic in a topic.
- The Check Out dialog box appears when you select **Check Out** from the **SDL LiveContent** menu in Arbortext Editor.
- The Select Target Folder dialog box appears when you select **Check In As** from the **SDL LiveContent** menu in Arbortext Editor.
- The View dialog box appears when you select **View** from the **SDL LiveContent** menu in Arbortext Editor.
- The Insert Conref dialog box appears when you insert a content reference (conref) in a topic.
- The Insert Variable dialog box appears when you insert a variable in a topic.

The new dialogs are specializations of the Browse Repository one and differ in their title, the purpose of their action button, and the default value for the Objects of type drop down list. For example, the Insert Graphic dialog box includes controls that enable you to insert a graphic.

Figure 3-1 shows an instance of this dialog box, notice how the Objects of type default value is set to Graphics, this allows only images to be displayed in the right pane, and the Insert button at the bottom of the window.

***Figure 3-2 Insert Graphic Dialog Box***




## Opening the Browse Repository Dialog Box

Open the Browse Repository dialog box to create objects and folders, search the repository, and access your inbox.

1. In Publication Manager, open the **Tools** menu.
2. Select the **Browse Repository** option.

The Browse Repository dialog box displays. The left pane includes the **Repository**, a tree view of repository, **Search** and **Inbox** tabs.

## Creating Folders

Use folders to organize objects of the same type in a repository location. Folders also enable you to restrict access to its object to a particular user group.

1. Open the Browse Repository dialog box.
2. In the **Repository** tab, navigate to the folder into which you want to create a folder.

3. Click the new folder icon.

4. In the Add Folder dialog box and under the General tab, enter a title for the folder.
5. From the **Content type** drop-down list, select one of the following content types:
   - **Graphic**: Choose this content type for a folder to hold images.
   - **Library**: Choose this content type for a folder to hold libraries.
   - **Master Document**: Choose this content type for a folder to hold DITA maps.
   - **Module**: Choose this content type for a folder to hold DITA topics.
   - **Publication**: Choose this content type for a folder to hold publications.
   - **Template**: Choose this content type for a folder to hold non-XML files, such Microsoft Word or Excel files.

   **Note:** **Note:** You cannot change the content type after you create the folder.

6. Optional: Change the folder security settings:
   a. In the **Security** tab, in the **Owner** drop-down list, select the appropriate user group.

      The Grant read access to pane becomes enabled. By default, all user groups have now access to this folder

   b. Select the **Members of the following user groups** option and click **Change**.
   c. In the User Groups dialog box, select the user groups that will have read access to the folder.

      **Note:** Any group that is not included will not be able to see the folder in the repository.

   d. Click **OK**.
7. Click **OK**.

The new folder appears in Publication Manager.

**See Also:**

Object Types Managed by SDL LiveContent Architect
: All of the components that make up a document are objects in the SDL LiveContent Architect (Architect) repository.

Modifying Folder Properties
: You modify the properties of a folder to change the folder's metadata. You can change the name of a folder and define user restrictions but you can't modify it's content type.

## Modifying Folder Properties

You modify the properties of a folder to change the folder's metadata. You can change the name of a folder and define user restrictions but you can't modify it's content type.

1. In Publication Manager, from the **Tools** menu, select the **Browse Repository** option.
2. In the **Repository** tab, locate and select the folder which properties you want to modify.
3. Right-click the folder, and select the **Properties** option.
4. In the Folder Properties dialog box, under the General tab, modify the folder's name.
5. Optional: Change the folder security settings:
   a. In the **Security** tab, in the **Owner** drop-down list, select the appropriate user group.

      The Grant read access to pane becomes enabled. By default, all user groups have now access to this folder

   b. Select the **Members of the following user groups** option and click **Change**.
   c. In the User Groups dialog box, select the user groups that will have read access to the folder.

      **Note:** Any group that is not included will not be able to see the folder in the repository.

   d. Click **OK**.
6. Click the **OK** button to save your changes.

**See Also:**

Creating Folders
: Use folders to organize objects of the same type in a repository location. Folders also enable you to restrict access to its object to a particular user group.

## Creating Topics from Publication Manager

Create a topic from the Browse Repository dialog box in Publication Manager.

Before creating a topic, you should understand the purposes of different topic types and topic-based writing principles. You should also know the correct metadata for the topic, such as the topic title, the product and product component related to the topic, and the author.

---

**Note:** You can also create topics in Arbortext Editor using the **SDL LiveContent** menu.

---

1. In Publication Manager, from the **Tools** menu, select **Browse Repository**.
2. In the **Repository** tab, locate and select the folder that will contain the new topic.

---

**Note:** Topics can only be created in folders of the "Module" content type. To check the content type of a folder, right-click the folder, and select **Properties**. The content type is displayed on the **General** tab.

---

3. Click the **New Object** button.

4. In the Select Template dialog box, under the tab for your group, select the template for the type of topic that you want to create.

   For example, if you are a database writer, then select a template under the Database Topics tab.

5. Click the **Next** button.
6. In the Add Object dialog box, fill in the metadata fields.
7. Click the **OK** button.

   The topic appears in the **Browse Repository** dialog box.

**See Also:**

DITA Skill Set Requirements
: To write effective documentation in DITA, writers need minimalist writing skills and topic-based writing skills.

Metadata for Topics
: Metadata for maps aids in searching for topics and tracking a topic's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

DITA Topic Templates
: InfoDev supplies a set of templates to use as the basis for new topics. It is strongly recommended that you use these templates rather than creating topics from scratch. The DITA Standards document is intended as accompanying documentation for the templates, and assumes that you are familiar with the templates.

DITA Standards for Topic Types
: All InfoDev documentation is built using only four topic types: task (`<task>`), concept (`<concept>`), reference (`<reference>`), and orientation (which uses `<topic>`). A glossary uses a different element (`<glossgroup>`), but is not considered a separate topic type.

Creating Topics from Arbortext Editor
: Create a topic from the **SDL LiveContent** menu in Arbortext Editor as an alternative to the Browse Repository dialog box in Publication Manager.

Adding Topics to a Publication
: You add an existing topic to a publication by inserting a topic reference (topicref) to the topic in the publication's bookmap. The topicref points to the topic in the repository using its GUID.

## Importing Graphics Using the Browse Repository Dialog Box

You import a graphic by dragging it from storage on your local computer to a folder in the **Browse Repository** dialog box. For a graphic with a separate EPS version for PDF output, use the Web Client to import the EPS file.

When you create a screen shot, there is only one file to import into the repository. When a graphic artist creates a graphic, the graphic artist typically creates two versions of the image: one version for XHTML output and one version for PDF output.

---

**Note:** The folder that will contain the graphic must already exist in the repository. The content type of the folder must be "Graphic". To verify the content type of a folder, right-click the folder, and select **Properties**. The content type is on the **General** tab.

---

1. In Publication Manager, from the **Tools** menu, select **Browse Repository**.
2. In the **Repository** tab, locate and select the folder that will contain the graphic.
3. Locate the graphic source file in your computers file system.
4. Position the Browse Repository dialog box and your local file system window side by side on your computer screen so that both are visible.

5. Drag the graphic file from your local file system to the right pane of the **Browse Repository** dialog box.

   Only drag and drop the screenshot version of the image, not the EPS file.

6. Complete the following steps to upload an EPS version of the image:

   a. Using your browser, go to the Web Client, and log in if necessary.

      Access one of the following URLs in your web browser:

| Application Instance | URL |
|---|---|
| Production | `https://dadvip0032.us.oracle.com/InfoShareAuthor/` |
| Development | `https://dadvip0015.us.oracle.com/InfoShareAuthor/` |

   b. Click the **Repository** tab.

   c. Locate and select the graphic file you imported in the previous steps.

   d. In the bottom pane, select the check box on the right that corresponds to the version of the graphic you want to modify, and then click the**New Resolution** button.

   e. In the **Add Illustration Document Language** dialog box, from the **Resolution** list, select **Print**.

   f. Select **File**, click the **Choose File** button, and then select the EPS version of the graphic in your local file system.

   g. In the **Add Illustration Document Language** dialog box, click **OK**.

Connect to the following URL in a browser:

**See Also:**

Inserting Figures and Graphics
: In DITA, images are inserted either as figures or graphics. Use figures for reference topics and concept topics. Use graphics when you need to insert images to refer to user interface features in a task or in a reference table. For both figures and graphics, you must always include a graphic description to make the image accessible.

Updating Multiple Images in a Publication

## Viewing Objects in the Repository Browser Dialog Box

You can preview an object in Publication Manager without going through the process of checking it out. The objects contents appear in a new pane of the repository browser, they can bee seen in HTML or XML, but cannot be modified unless the object is checked out.

**Note:** The HTML formatting might not reflect the final XHTML output or PDF output.

1. In Publication Manager, from the **Tools** menu, select the **Browse Repository** option.
2. In the **Repository** tab, locate and select the object you want to preview.
3. Click the **Views** button

   The preview pane appears at the right side of the Browse Repository dialog box.
4. Optional: Change the format of the preview pane:
   a. Right-click anywhere in the preview pane.
   b. From the context menu, select **View HTML/XML**.

**See Also:**

Object Types Managed by SDL LiveContent Architect
: All of the components that make up a document are objects in the SDL LiveContent Architect (Architect) repository.

Viewing an Object in Arbortext Editor
: You can view a topic, map, publication, or library in Arbortext Editor. Although viewing an object in Arbortext Editor limits your ability to modify its contents it is useful when you want to review the contents of an object that you don't have permissions to modify.

## Moving Objects

You can re-organize objects in the repository by moving them from one folder to another. An object must be placed in a folder of the correct content type.

Changing an objects location in the repository does not affect references to the object. After moving an object, references to the object remain valid because they rely on the object's GUID, not a "filepath". You do not need to modify maps or objects that reference the one that you relocated.

1. In Publication Manager, from the **Tools** menu, select **Browse Repository**.
2. In the **Repository** tab, locate the object that you want to move.

   The object appears in the right pane of Publication Manager.
3. In the **Repository** tab, locate the folder to which you want to move the object.
4. Drag the object from the right pane to the correct folder in the **Repository** tab.

---

**Note:** The target folder must be of the same content type as the object that you are moving. For example, topics must be placed in folders of content type "Module," and images must be placed in folders of content type "Graphic."

---

**See Also:**

Object Types Managed by SDL LiveContent Architect

## Modifying an Object's Metadata

You modify an object's metadata by using the Properties dialog box. A common reason to modify the properties of an object, such as a topic or an image, is to update the object's workflow or relate it to a particular product component.

1. In Publication Manager, from the **Tools** menu, select **Browse Repository**.
2. In the **Repository** tab, locate and select the object you want to modify.
3. Right-click the object, and select **Properties**.
4. In the **Properties** dialog box, modify the object's properties as needed..

> **Note:** You might need to select a specific tab to modify a property. For example, select the **Workflow** tab to modify a topic's workflow.

**See Also:**

Object Types Managed by SDL LiveContent Architect

Metadata for Objects

SDL LiveContent Architect tracks and maintains metadata for each object in the repository. The metadata contains relevant information about each object. It is used to describe each object in the repository, to search for objects, and to track each object through its workflow.

## Deleting an Object

Deleting an object is restricted most of the time because of the dependencies introduced by topic-based writing: you cannot delete an object that is referenced by another one. Run a Where Used report to verify if any reference targets the object you want to delete.

If it is not reasonable to remove all of the references to an object that must be deleted, then move the object to the Trash folder in the repository hierarchy.

> **Important:** Only delete objects from the Repository tab in the Browse Repository dialog box, never from the Inbox or Search tabs

1. In Publication Manager, from the **Tools** menu, select **Browse Repository**.
2. In the **Repository** tab, locate the object that you want to delete.
3. Do one of the following:
   - Right-click the object, and select **Delete**
   - Click the **Delete Object** button.
4. In the Confirm Object Delete dialog box, click the **Yes** button.
5. Optional: If a warning dialog box appears stating that the object cannot be deleted, then run a Where Used report to track and delete all references to the topic.

**See Also:**

Determining Where an Object is Referenced
: You determine where an object is referenced by running a Where Used report. This feature is available in Arbortext Editor or Publication Manager.

## Searching the Repository

You search for objects in the repository using the Search tab in the Browse Repository dialog box. When you search the repository, the system searches the metadata of all objects and the full text of all XML objects.

**Tip:**

Consider the following items when you are performing a search:

- Use the assist button for a field where it is available to ensure that you enter the correct criteria in the field. An assist button has three dots on it.
- You can use operators, such as =, >, and <, in appropriate search fields.
- You can use * as a wildcard character in single-word searches. Do not enter * as the first character in the word, and do not use * as a wildcard in phrase searches.
- You can surround text with quotation marks to search for an exact phrase.
- You can use Boolean operators, such as `AND`, `OR`, and `NOT`, when you enter multiple search criteria in a field. When you enter a comma-delimited list in a search field, the search uses the equivalent of an `AND` operator in a Boolean search.

1. In Publication Manager, from the **Tools** menu, select **Browse Repository**.
2. In the Browse Repository dialog box, click the **Search** tab.
3. Enter search criteria in one or both of the following ways:
   - In the **Find what** field, enter your general search criteria.

     **Note:** You can list and select recent search criteria using this field.

   - Expand one or more search criteria categories by clicking the + sign next to a category, and enter the additional search criteria in the appropriate field.
4. Click **Search**.

   Search results appear in the right pane.

**See Also:**

Object Types Managed by SDL LiveContent Architect

Metadata for Objects
: SDL LiveContent Architect tracks and maintains metadata for each object in the repository. The metadata contains relevant information

about each object. It is used to describe each object in the repository, to search for objects, and to track each object through its workflow.

## Determining Where an Object is Referenced

You determine where an object is referenced by running a Where Used report. This feature is available in Arbortext Editor or Publication Manager.

Writers and architects might want to determine where an object is referenced to learn more about the object's context. For example, a writer might want to know which maps references a topic to learn more about a feature described in the topic. In addition, it is important to determine where an object is used before deleting an object. An object that is referenced at least once cannot be deleted.

Do one of the following to run a Where Use report for an object:

- Use the SDL LiveContent menu in Arbortext Editor:

  1. Load the object in Arbortext Editor either by viewing it (**SDL LiveContent** > **View**) or checking it out (**SDL LiveContent** > **Check Out**).
  2. From the **SDL LiveContent** menu, select **Where Used**.

     The Where Used dialog box appears. It shows all of the object where the object is referenced in the entire repository.

- Use the Object menu in Publication Manager:

  1. In Publication Manager, open a publication that includes the object.
  2. In the publication tree, locate and select the desired object.
  3. From the **Object** menu, select **Where Used**.

     The Where Used pane appears at the right side of Publication Manager.
  4. In control bar, at the top of the Where Used pane, select the scope of the report from the drop-down list:

     - **Current Publication**: The report only includes objects that compose the current publication.
     - **Entire Repository**: The report includes all objects in the repository that target the selected object.

**See Also:**

Checking Out Objects
: To edit objects locally, you must first check them out of the repository. When you check an object out of the repository, you lock the object, preventing someone else from making changes to the same object.

Viewing an Object in Arbortext Editor
: You can view a topic, map, publication, or library in Arbortext Editor. Although viewing an object in Arbortext Editor limits your ability to

modify its contents it is useful when you want to review the contents of an object that you don't have permissions to modify.

## Viewing an Object in Arbortext Editor

You can view a topic, map, publication, or library in Arbortext Editor. Although viewing an object in Arbortext Editor limits your ability to modify its contents it is useful when you want to review the contents of an object that you don't have permissions to modify.

Arbortext Editor displays a gray background when you are viewing an object. You cannot modify an object unless you check it out.

1. In Arbortext Editor and from the **SDL LiveContent** menu, select the **View** option.
2. In the View window, locate and select the object that you want to view.
3. Click the **View** button.

   Arbortext Editor displays the object.
4. Optional: Select items from the **View** menu to choose various view options.

   For example, you can view an object with full tags, partial tags, or no tags.
5. Optional: When you are done viewing the object, select **Close** from the **SDL LiveContent** menu.

**See Also:**

Object Types Managed by SDL LiveContent Architect
: All of the components that make up a document are objects in the SDL LiveContent Architect (Architect) repository.

Viewing Objects in the Repository Browser Dialog Box
: You can preview an object in Publication Manager without going through the process of checking it out. The objects contents appear in a new pane of the repository browser, they can bee seen in HTML or XML, but cannot be modified unless the object is checked out.

Checking Out Objects
: To edit objects locally, you must first check them out of the repository. When you check an object out of the repository, you lock the object, preventing someone else from making changes to the same object.

Metadata for Objects
: SDL LiveContent Architect tracks and maintains metadata for each object in the repository. The metadata contains relevant information about each object. It is used to describe each object in the repository, to search for objects, and to track each object through its workflow.

## Creating Topics from Arbortext Editor

Create a topic from the **SDL LiveContent** menu in Arbortext Editor as an alternative to the Browse Repository dialog box in Publication Manager.

Before creating a topic, you should understand the purposes of different topic types and topic-based writing principles. You should also know the correct metadata for the

topic, such as the topic title, the product and product component related to the topic, and the author.

1. In Arbortext Editor, from the **SDL LiveContent** menu, select **New**.
2. In the Select Template dialog box, select the tab that corresponds to your content group.

   For example, if you are a database writer, then select the **Database Topics** tab.
3. Select the template for the type of topic that you want to create.
4. In the Select Target Folder dialog box, locate and select the folder that will contain the new topic.

   **Note:** Topics can only be created in folders of the "Module" content type. To check the content type of a folder, right-click the folder, and select **Properties**. The content type is displayed on the **General** tab.
5. In the Select Target Folder dialog box, click the **Next** button.
6. In the Add Object dialog box, fill in the metadata fields.
7. Optional: To check out the new topic and open it in read/write mode in Arbortext Editor, ensure that **Immediately check out the object** is selected.

   **Note:** When **Immediately check out the object** is not selected, the new topic opens in Arbortext Editor in read-only mode.
8. Click the **OK** button.

The new topic is created in the repository and loaded in Arbortext Editor.

**See Also:**

DITA Skill Set Requirements
: To write effective documentation in DITA, writers need minimalist writing skills and topic-based writing skills.

Metadata for Topics
: Metadata for maps aids in searching for topics and tracking a topic's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

DITA Topic Templates
: InfoDev supplies a set of templates to use as the basis for new topics. It is strongly recommended that you use these templates rather than creating topics from scratch. The DITA Standards document is intended as accompanying documentation for the templates, and assumes that you are familiar with the templates.

DITA Standards for Topic Types
: All InfoDev documentation is built using only four topic types: task (`<task>`), concept (`<concept>`), reference (`<reference>`), and

orientation (which uses <topic>). A glossary uses a different element (<glossgroup>), but is not considered a separate topic type.

Creating Topics from Publication Manager
: Create a topic from the Browse Repository dialog box in Publication Manager.

Adding Topics to a Publication
: You add an existing topic to a publication by inserting a topic reference (topicref) to the topic in the publication's bookmap. The topicref points to the topic in the repository using its GUID.

## Checking Out Objects

To edit objects locally, you must first check them out of the repository. When you check an object out of the repository, you lock the object, preventing someone else from making changes to the same object.

You can check out objects from Publication Manager's publication tree (inside the Content tab), the Browse Repository dialog box, or directly from Arbortext Editor.

**Note:** Checking a topic out from the publication tree opens the topic in Arbortext Editor, but checking a map out only enables you to modify its contents from the Publication Manager interface. To edit a map in Arbortext Editor, check the map out from the **Browse Repository** dialog box.

- To check out an object from the publication tree:

  1. Open your publication in the Publication Manager.
  2. In the tree-view of the Publication Manager, select the object you want to check out.
  3. Do one of the following:
     - From the **Object** menu, select **Check Out**.
     - Right-click the object, and select the **Check Out** option.

  The object opens in Arbortext Editor, ready for editing.

- To check out an object from the Browse Repository dialog box:

  1. In Publication Manager, from the **Tools** menu, select the **Browse Repository**option.
  2. In the **Repository** tab of the **Browse Repository** dialog box, select the folder that contains the object you want to check out.
  3. From the right pane, right-click the object you want to check out, and select the **Check Out** option.

  The object opens in Arbortext Editor.

- To check out an object from Arbortext Editor:

  1. In Arbortext Editor and from the **SDL LiveContent** menu, select the **Check Out** option.

2. In the **Repository** tab of the Check Out dialog box, select the folder that contains the object you want to check out.
3. Click the **Check Out** button to open the object in Arbortext Editor.

   **Tip:** If you recently checked out, edited, and checked in a topic and decide you want to check it out again, then you can select it from Arbortext Editor's **File** menu, which shows the most recently edited topics. The topic opens in view (read-only) mode, and you can select **Check Out** from the **SDL LiveContent** menu to check it out.

**See Also:**

Object Types Managed by SDL LiveContent Architect
: All of the components that make up a document are objects in the SDL LiveContent Architect (Architect) repository.

Viewing an Object in Arbortext Editor
: You can view a topic, map, publication, or library in Arbortext Editor. Although viewing an object in Arbortext Editor limits your ability to modify its contents it is useful when you want to review the contents of an object that you don't have permissions to modify.

## Checking In Objects

When you finish editing an object, you must check it back into the Repository to upload the latest data and to unlock the object, which makes it available for checkout by other users.

You can check in all kinds of objects from within Arbortext Editor; use the main Publication Manager interface only to check in maps that you previously modified

**Tip:** If the object is a topic, then review it after selecting **No Tags** under the **View** menu before checking it in. Reviewing the topic with the tags hidden might reveal formatting errors, extra spaces, empty elements, and other problems.

- To check in objects from within Arbortext Editor:
  1. From the **SDL LiveContent** menu, select **Check In**.
  2. Optional: In the **Check In** dialog, check the **Change Status** box and select the end status from the drop-down list.
  3. Click **OK**.
- To check in maps from Publication Manager:
  1. Select the map you want to check in from the publication tree (within the Content tab).

     Maps that are checked out have an open lock icon next to them. 
  2. From the **Object** menu, select **Check In**.
  3. Optional: In the **Check In** dialog, check the **Change Status** box and select the end status from the drop-down list.

4. Click **OK**.

## Duplicating an Existing Object

You can create a new object by copying an existing object in the repository using Arbortext Editor. After you copy the object, there are two independent objects, and each object has its own GUID.

1. Load the existing object into the Arbortext Editor by using one of the following **SDL LiveContent** menu options:

   - **Check Out**
   - **View**

   Arbortext Editor loads the object.

2. From the **SDL LiveContent** menu, select the **Check In As** option.
3. In the Select Target Folder dialog box, locate and select the folder that will contain the new object.

   **Note:** The folder's content type must be the same as the object you are trying to replicate. For example, for a topic, the folder type must be `Module`.

4. Click the **Next** button.
5. In the Add Object dialog box, enter the metadata for the object..
6. Click the **OK** button.

The new object is available in the specified folder in the repository.

**See Also:**

Object Types Managed by SDL LiveContent Architect

Metadata for Objects
: SDL LiveContent Architect tracks and maintains metadata for each object in the repository. The metadata contains relevant information about each object. It is used to describe each object in the repository, to search for objects, and to track each object through its workflow.

Checking Out Objects
: To edit objects locally, you must first check them out of the repository. When you check an object out of the repository, you lock the object, preventing someone else from making changes to the same object.

Viewing an Object in Arbortext Editor
: You can view a topic, map, publication, or library in Arbortext Editor. Although viewing an object in Arbortext Editor limits your ability to

modify its contents it is useful when you want to review the contents of an object that you don't have permissions to modify.

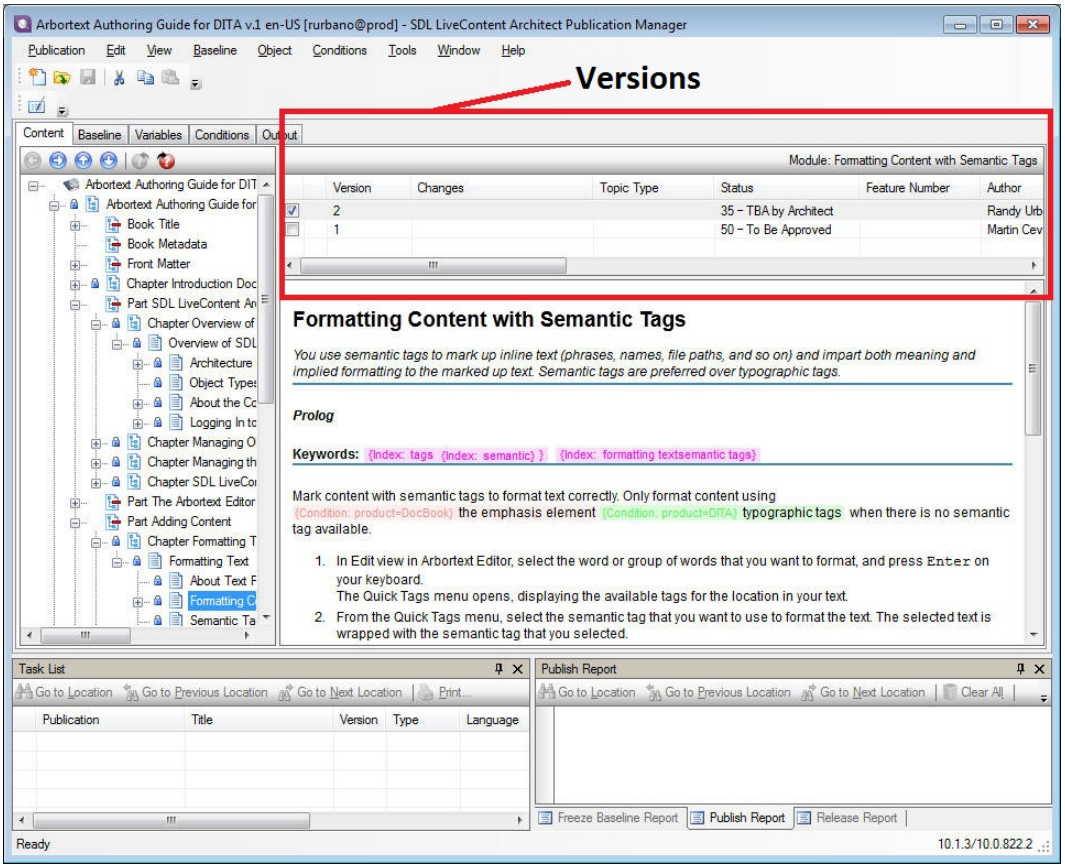
## About Object Versions

Object versions store changes over time. Although logically the same object, each version adheres to its own lifecycle (workflow). For example, after version 1 is released, you need to create version 2 to make changes to the object.

Several versions of an object can exist. All versions of an object have the same GUID, but each version has a different version number.

You can include the correct version of each object in a publication baseline. The object versions in a publication do not need to match the version of the publication. For example, version 2 of a publication can include objects of several different versions, such as 1, 2, 3, or any other object version number.

The following image shows Publication Manager displaying the versions of a selected topic:




### Versions and Revisions

It is important to understand the difference between versions and revisions. A new revision of an object is created automatically every time the object is checked into the repository. These revisions enable you to go back to a previous incarnation of the object if necessary. However, you must create a new version of an object explicitly. For example, you might create a new version of an object if changes are required to the object for a new release of a product.

**Note:** Revisions can only be seen and compared from the Web Client.

## Selecting an Object Version

Several versions of an object can exist. Select the correct version of each object in a publication. The selected version is included in the publication's baseline.

1. In Publication Manager, open the publication that contains the object.
2. Locate and select the object in the tree view of the Publication Manager.
3. In the top-right pane, select the check box next to the desired version of the object.
4. Save the publication.

The selected version is in the publication's baseline and appears in the publication's output.

**See Also:**

Managing Publication Baselines
: A baseline lists the versions of the objects that are used in a publication. Each publication has exactly one baseline. You manage baselines with the **Baseline** tab in Publication Manager.

## Creating a New Version of an Object

You can create a new version of an existing object, this enables you to use those objects in other documents or other versions of documents without changing the original publication.

A typical scenario for using versions is when you create a new version of a publication where you need to add new chapters or topics. You can also create versions of topics that enable you to edit existing topics without changing the original. This feature enhances the reuse of existing content.

Do one of the following to create a new version of an object

- From Publication Manager:
  1. Open the appropriate publication and locate the object for which you want to create a new version.
  2. In the right pane of the Publication Manager, right-click an existing version of the object and click **New Version**.
  3. In the Properties dialog box, edit the metadata of the object and click **OK**.
- From the Browse Repository dialog box:
  1. Do one of the following to locate the object for which you want to create a new version:
     - In the **Repository** tab, browse the repository folder structure.
     - In the **Search** tab, search for the object by its metadata.
     - In the **Inbox** tab, locate the object from one of your role inboxes.

2. In the dialog's right-pane, select the desired object and click **Versions**.
3. In the Select Versions dialog box, click **New Version**
4. In the Properties dialog box, edit the metadata of the object and click **OK**.
5. Click **OK** to dismiss the Select Versions dialog box.

You can now check out the object and edit it, accordingly.

## Viewing a Version History of an Object

The version history of an object is a record of each version of the object. The list of all versions of an object displays in the Content tab of Publication Manager.

1. In Publication Manager, open the publication by checking it out.
2. Locate and select the topic in the tree view of the Publication Manager.

The version history of the selected object appears in the top of the right pane in Publication Manager. You can select each of them to preview them in the bottom right pane.

## Comparing Different Versions of an Object

Comparing different versions of an object shows the differences between the selected versions of the object. You might want to compare versions of an object to ensure that you have the correct version selected in a publication.

You can compare different versions of an object in Arbortext Editor or in Publication Manager. The comparison can include local versions of the object on your computer system with versions of the object in the repository.

Do one of the following to compare versions of a topic:

- From Arbortext Editor:
  1. Open the object in Arbortext Editor by checking it out.
  2. From the **SDL LiveContent** menu, select the **Compare** option.

     The Differences dialog box appears, showing the differences in the versions. By default, it shows the differences between the current version of the local object and the same version of the object in the repository. If there are no differences, then the Differences dialog box displays the message "The objects are the same."
  3. Optional: To change the versions that are compared, change the version selected in the **Compare Version** and **To Version** lists.
  4. Click **Close** when you are done.
- From Publication Manager:
  1. In Publication Manager, open a publication that includesthe object.
  2. Locate and select the topic in the tree view of Publication Manager.
  3. In the top of the right pane of Publication Manager, select the versions you want to compare.

**Note:** Use the Shift key on your keyboard to select more than one version.

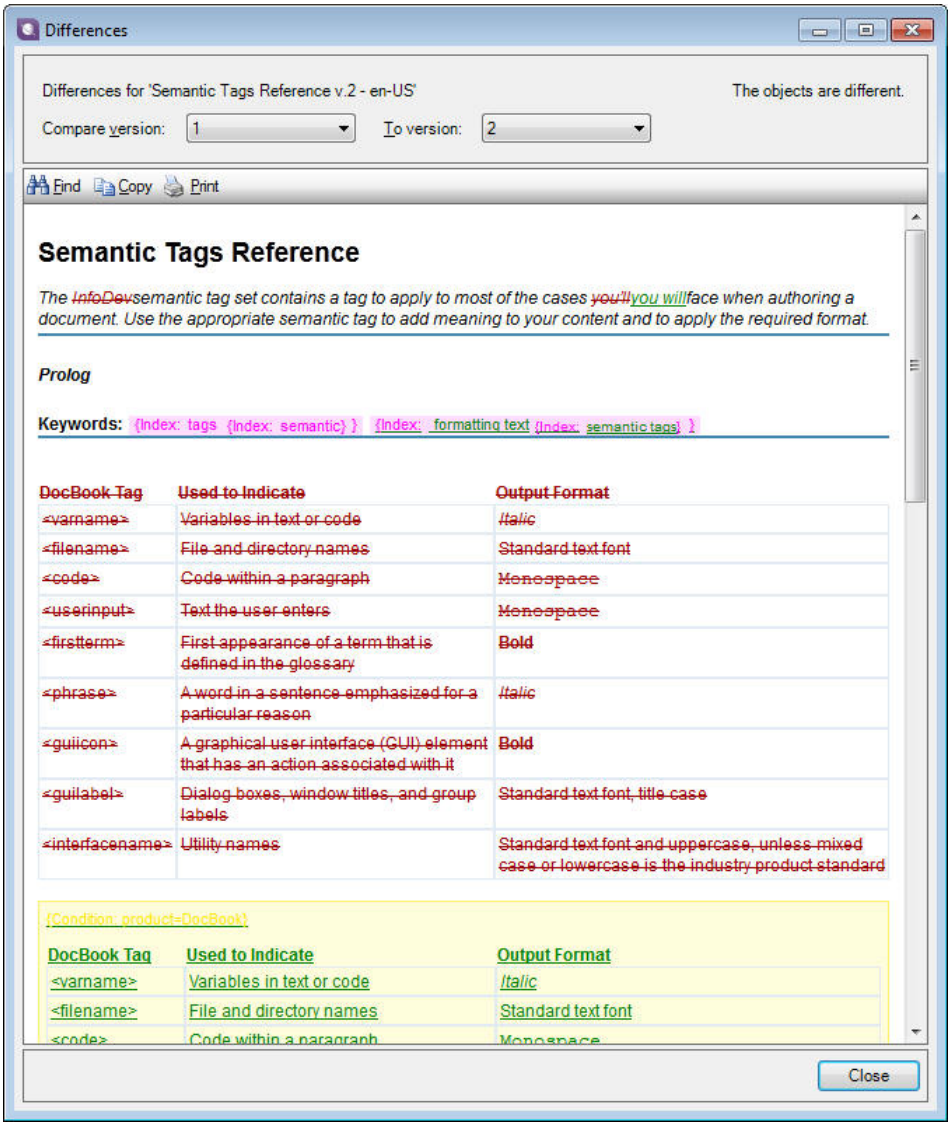
4. From the **Object** menu, select **Compare**.

   The Differences dialog box appears, showing the differences in the versions. If there are no differences, then the Differences dialog box displays the message "The objects are the same."

5. To change the versions that are compared, change the version selected in the **Compare Version** and **To Version** lists.

6. Click **Close** when you are done.

The following is a sample Differences dialog box:



## Viewing the Revision History of Objects

You use the Web Client to view the revision history of an object.

A revision of an object is created when the object is checked out, changed, and checked in. A revision is an intermediary version of an object that is automatically stored in the repository without creating a new version number. You can restore or revert to a previous revision if necessary.

1. In a web browser, go to the Web Client and log in if necessary.

| | |
|---|---|
| Production server | `https://dadvip0032.us.oracle.com/InfoShareWS/` |
| Development server | `https://dadvip0015.us.oracle.com/InfoShareWS/` |

> **Note:** Login to the development server only for training purposes. Always use your production account when writing official documentation.

2. Click the **Repository** tab to locate and select the desired object.
3. In the bottom pane, select the check box of a version of the object on the right.

> **Note:** Each version of an object has a different revision history.

4. Click **Show History**.

The History dialog box shows the revision history of the object. You can download, preview, and restore a selected revision.

**See Also:**

Restoring a Revision of an Object
: You restore a revision of an object when you want to dismiss recent changes therefore reverting to a previous state of its version.

## Restoring a Revision of an Object

You restore a revision of an object when you want to dismiss recent changes therefore reverting to a previous state of its version.

1. View the revision history of the object in the Web Client.
2. In the History dialog box, select the revision to which you want to revert.
3. Click the **Restore** button.

**See Also:**

Viewing the Revision History of Objects
You use the Web Client to view the revision history of an object.

# Modifying an Earlier Release of a Publication

You modify an earlier release of a publication by branching objects in the publication. A common reason to modify an earlier release of a publication is to correct documentation bugs.

## About Branching

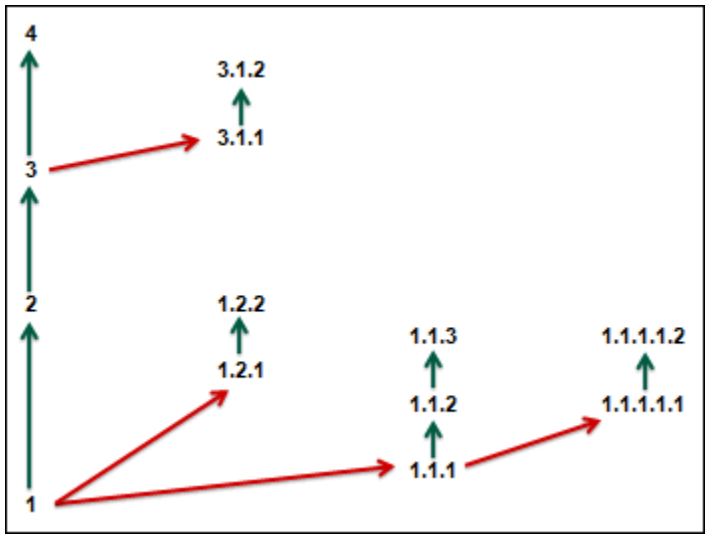
You branch an object when a newer version of an object exists, and you must change a previous version of the object. A new branch adds two digits to the version number.

Every object has linear version numbering, and each version number represents a lifecycle of the object that ends in the release of the version. For example, version 1 of an object might represent the lifecycle of an object for release 1.1 of a product, and version 2 of an object might represent the lifecycle of the same object for release 1.2 of a product. After a version of an object is released, that version of the object cannot be changed.

The following figure shows branches from objects. Notice that more than one branch can be created from a single version. The figure shows two branches (starting with release 1.1.1 and release 1.2.1) from version 1 of the object. Also, notice you can create a branch from a branch. The branch starting with release 1.1.1.1.1 was created from the branch starting with release 1.1.1.

**Note:** It is not required that branch numbers for objects match product version numbers or product release numbers.

***Figure 3-3 Branching***




It is sometimes necessary to change an older version of an object after it is released. For example, you might need to change an older version of an object to correct a documentation bug in that version of the object. In this case, you can create a new branch from the version. You can modify the branch of the object, and republish the publication.

With branching, writers can continue to work on a newer version of an object while you correct errors in an older version of the object. A branch of an object proceeds through its own lifecycle and cannot be merged with the main version of the object.

**See Also:**

Branching an Object
: When a higher version of an object exists, and you must make a change to a previous version of the object, you can branch the object. For example, if you must correct a documentation bug in a topic that was released, then you can branch the topic to edit it.

About Object Versions
: Object versions store changes over time. Although logically the same object, each version adheres to its own lifecycle (workflow). For example, after version 1 is released, you need to create version 2 to make changes to the object.

## Branching an Object

When a higher version of an object exists, and you must make a change to a previous version of the object, you can branch the object. For example, if you must correct a documentation bug in a topic that was released, then you can branch the topic to edit it.

1. In Publication Manager, open a publication that contains the object you want to branch.
2. In the publication tree, locate and select the object.
3. In the right pane of the Publication Manager, right-click an existing version of the object, and select **New Branch**.
4. In the Properties dialog box, edit the metadata of the object, and click **OK**.

   The new branch is created and is selected for the publication automatically.

You can now select the new branch, check out the object, and edit it.

**See Also:**

About Branching
: You branch an object when a newer version of an object exists, and you must change a previous version of the object. A new branch adds two digits to the version number.

# Updating Multiple Images in a Publication

filenames required to import images

1. Create a new version of the images to be updated
   a. can this be done by bulk in the baseline tab?
2. Determine the necessary filenames for images
   a. Export the publication using the web client (do we have a procedure to reference? no, we need the procedure in chapter 6, below running workflows reports)

   b. locate the images that you want to update (identify them using the naming convention)

   c. rename your source files

3. Generate the images locally

4. Update Version N using the web client's batch import featuer

# Working Offline

If you need to work on objects without network access to the repository, then you can plan to work offline by making the objects available in local storage. When you have access to the repository again, you can commit the objects with your offline changes to the repository.

## About Local Storage

You can make objects such as topics accessible in local storage and work on them offline.

When you check an object out of the repository, the object is downloaded to a local storage folder on your computer system. Local storage includes objects that are checked out currently and objects that were checked out previously and then checked in.

When you are working on an object that you have checked out and save the object with the Arbortext Save command, the object is read/write in local storage, and the changes are saved in the local storage copy of the object. When you check in the object, the changes are saved in the object in the repository, and the copy of the object in local storage becomes read-only.

If you plan to work offline (with no access to the repository), then you can borrow an Arbortext Editor license and make objects accessible in local storage. While you are working on the objects offline, they remain checked out in the repository. Therefore, they are locked, and other users cannot make changes to the objects.

If you want to add many objects to local storage for a long offline work session, then you must add them one at a time.

---

**Note:** After you make changes to an object in local storage while you are working offline, and you reconnect so that you have access to the repository, it is best practice to check in the object as soon as possible. If an object that is in local storage is checked out, and you attempt to check the same object out of the repository, then the system asks you if you want to overwrite the object in local storage with the object from the repository. If you do overwrite the object, then the most recent changes in your local copy are lost.

---

**See Also:**

Borrowing an Arbortext License for Offline Work
: You must borrow an Arbortext license if you plan to work offline.

Making Objects Available in Local Storage
: When an object is checked out and available in local storage, you can work on the object offline.

Opening Objects in Local Storage
: When an object is checked out, you can open it in local storage to work on it offline. If the object is not checked out, then you can view a read-only version of the object in local storage.

Checking In Objects in Local Storage
: When you have access to the repository after you have worked on an object offline in local storage, you can check the object in to record your changes in the repository.

Removing Objects From Local Storage
: Over time local storage can contain a large number of objects. To simplify file management, you can remove objects from local storage.

## Borrowing an Arbortext License for Offline Work

You must borrow an Arbortext license if you plan to work offline.

If you plan to work offline, then you must borrow an Arbortext Editor license to modify files in local storage while you have no access to the repository.

1. From the **Tools** menu, select **Administrative Tools** > **Borrow Licenses**.
2. In the Borrow Licenses dialog box, complete the following steps:
   a. Move **Arbortext Editor** from the **Available** list to the **Borrowed** list.
   b. In the **Borrow license for** list, specify the number of days that you want to borrow the license.
   c. Click the **Apply** button.

You have a license for working offline.

**See Also:**

About Local Storage
: You can make objects such as topics accessible in local storage and work on them offline.

## Making Objects Available in Local Storage

When an object is checked out and available in local storage, you can work on the object offline.

An object is downloaded to local storage when you check it out. When you make the object available in local storage, you also download all references to the object so that you can work on the object offline.

1. Check out the object.

2. With the object open in Arbortext Editor, select **Make Available in Local Storage** from the **SDL LiveContent** menu.

You can work on the object offline. When you have access to the repository, you can check in the object.

**See Also:**

About Local Storage
: You can make objects such as topics accessible in local storage and work on them offline.

Checking Out Objects
: To edit objects locally, you must first check them out of the repository. When you check an object out of the repository, you lock the object, preventing someone else from making changes to the same object.

## Opening Objects in Local Storage

When an object is checked out, you can open it in local storage to work on it offline. If the object is not checked out, then you can view a read-only version of the object in local storage.

1. Open Arbortext Editor.
2. From the **SDL LiveContent** menu, select **Open Local Storage**.
3. In the Local Storage dialog box, select the object, and click **View**.

**Note:** You can sort the objects by selecting a column title.

If the object is checked out, then it opens in Arbortext Editor in read/write mode, and you can work on the object and save changes to it in local storage. If the object is checked in, then it opens in Arbortext Editor in read-only mode.

**See Also:**

About Local Storage
: You can make objects such as topics accessible in local storage and work on them offline.

## Checking In Objects in Local Storage

When you have access to the repository after you have worked on an object offline in local storage, you can check the object in to record your changes in the repository.

It is best practice to check in an object as soon as possible after you finish making changes to it. Checking in the object enables other users to make changes to the object and ensures that your changes are present in output that includes the object.

If the object is a topic, then, before checking it in, it is best practice to review the topic after selecting **No Tags** under the **View** menu. Reviewing the topic with the tags hidden might reveal formatting errors, extra spaces, empty elements, and other problems.

You must have access to the repository to check in an object.

1. Open the object in local storage.
2. From the **SDL LiveContent** menu, select **Check In**.

**See Also:**

About Local Storage
: You can make objects such as topics accessible in local storage and work on them offline.

Opening Objects in Local Storage
: When an object is checked out, you can open it in local storage to work on it offline. If the object is not checked out, then you can view a read-only version of the object in local storage.

## Removing Objects From Local Storage

Over time local storage can contain a large number of objects. To simplify file management, you can remove objects from local storage.

When you remove an object from local storage, it does not affect the object in the repository. Instead, it removes the object from the local storage folder on your computer system.

> **Note:** If an object is checked out, and you remove it from local storage, then any changes you made to the object are lost.

You can remove an object from local storage using either Arbortext Editor or Publication Manager.

- To remove an object from local storage using Arbortext Editor:
  1. Open Arbortext Editor.
  2. From the **SDL LiveContent** menu, select **Open Local Storage**.
  3. In the Local Storage dialog box, select the object and click **Delete**.

     > **Note:** You can sort the objects by selecting a column title.

  4. In the Confirm Object Delete dialog box, click **Yes**.
  5. Click **Cancel** to close the Local Storage dialog box.
- To remove an object from local storage using Publication Manager:
  1. In Publication Manager, select **Local Storage** from the **Tools** menu.
  2. In the Local Storage dialog box, select the object, and click **Delete**.

     > **Note:** You can sort the objects by selecting a column title.

  3. In the Confirm Object Delete dialog box, click **Yes**.
  4. Click **Cancel** to close the Local Storage dialog box.

**See Also:**

About Local Storage
: You can make objects such as topics accessible in local storage and work on them offline.

# 4

# Managing the Lifecycle of Objects with Workflows

Use workflow to assist in the collaborative writing process and to track the progress of objects through their lifecycles.

All objects that you create in SDL LiveContent Architect have a certain workflow that tracks the objects from creation to when a publication is released. Each type of object has a specific workflow and different stages of the workflow involve different members of the content team.

## About Workflows

Using workflows, you can send topics and illustrations to others in your group for authoring, reviewing, translating, or any other task required to complete the work.

Every object in the repository is subject to workflow, where the object is transferred from one step, or user, to another. When an object is at the stage in the workflow where you must do something to it (either provide content or a review), the object displays in your inbox.

Workflow in SDL LiveContent Architect depends on the following:

- **The Status of an Object**

  All objects in the repository have a status that indicates where the object is relative to its lifecycle. For example, if an object has a status of **Draft**, then the object is being authored or reviewed and is not ready to be released for publication.

- **User Role**

  The user role indicates who has permissions to perform a particular action on the object. It also defines who is assigned to complete the action or task. For example, the **Author** role is assigned to objects that are in **Draft** status, and the **Reviewer** role is assigned to objects that are in a **To be reviewed** (**TBR**) status. Architects assign writers to roles when they create objects and, when an object is in a specific status, writers assigned a particular role are then able to perform the action or task on the object.

- **Status Transition**

  When the status of an object changes, such as from **Draft** to **TBR**, indicating that the object is ready to be reviewed by another user (writer or subject matter expert), the author of the object changes the status, which sends the object to the appropriate user for the next step in the flow.

- **States**

  Generally speaking, objects are either in the *Draft* or *Released* state. The *Draft* state indicates that an object is not yet ready for the intended audience, while the

*Released* state indicates that an object is ready for the intended audience, and the object can only be modified by an administrator.

**See Also:**

Viewing Your Inboxes

Objects that are related to you, such as objects that you own or objects that require your attention, are organized in the inboxes in the Browse Repository dialog box.

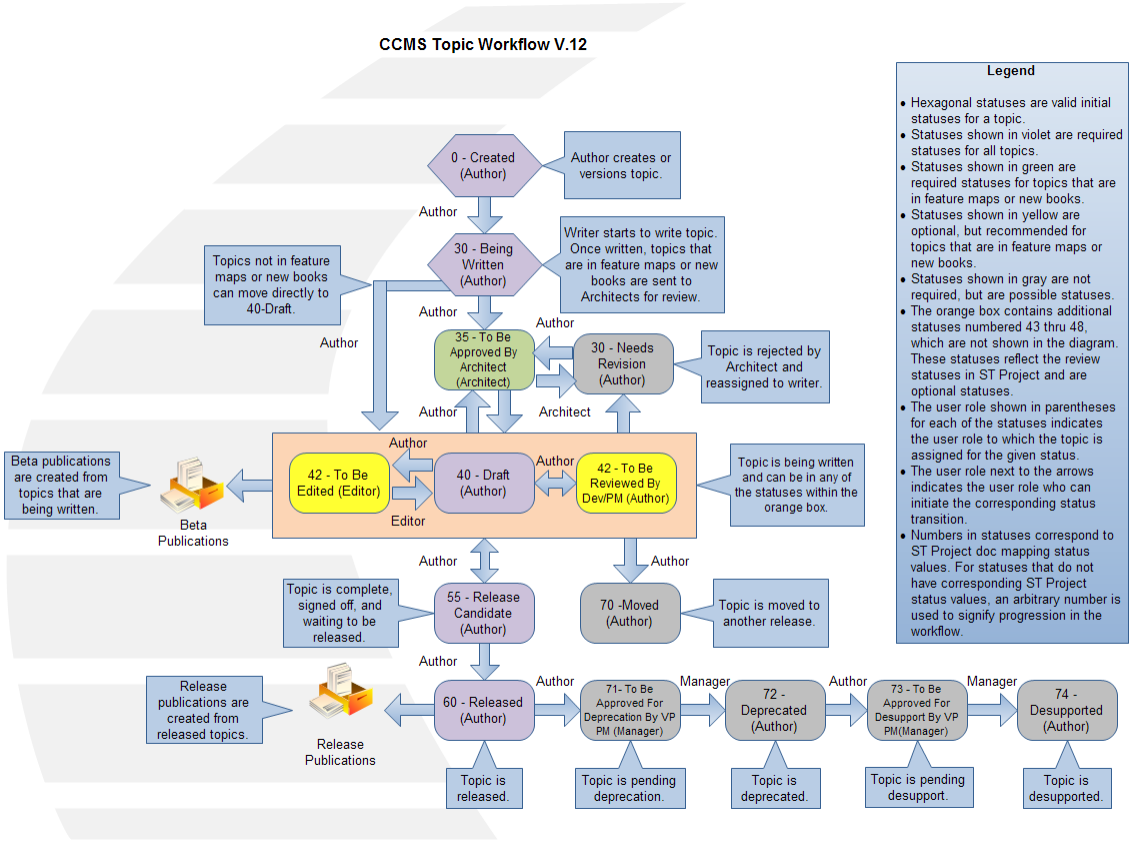
## Overview of the Workflow for a Topic

The workflow for a topic tracks the status of the topic from creation to release.

This workflow diagram for topics includes the following information:

- Each valid status for a topic.
- The user role assigned to complete an action at a particular status in parentheses.
- Status transitions in blue arrows labeled with the user role that is authorized to make the transition.
- Brief descriptions of some of the statuses.

The legend in the workflow diagram contains more information about the diagram.



**Tip:** You can enlarge the diagram in one of the following ways:

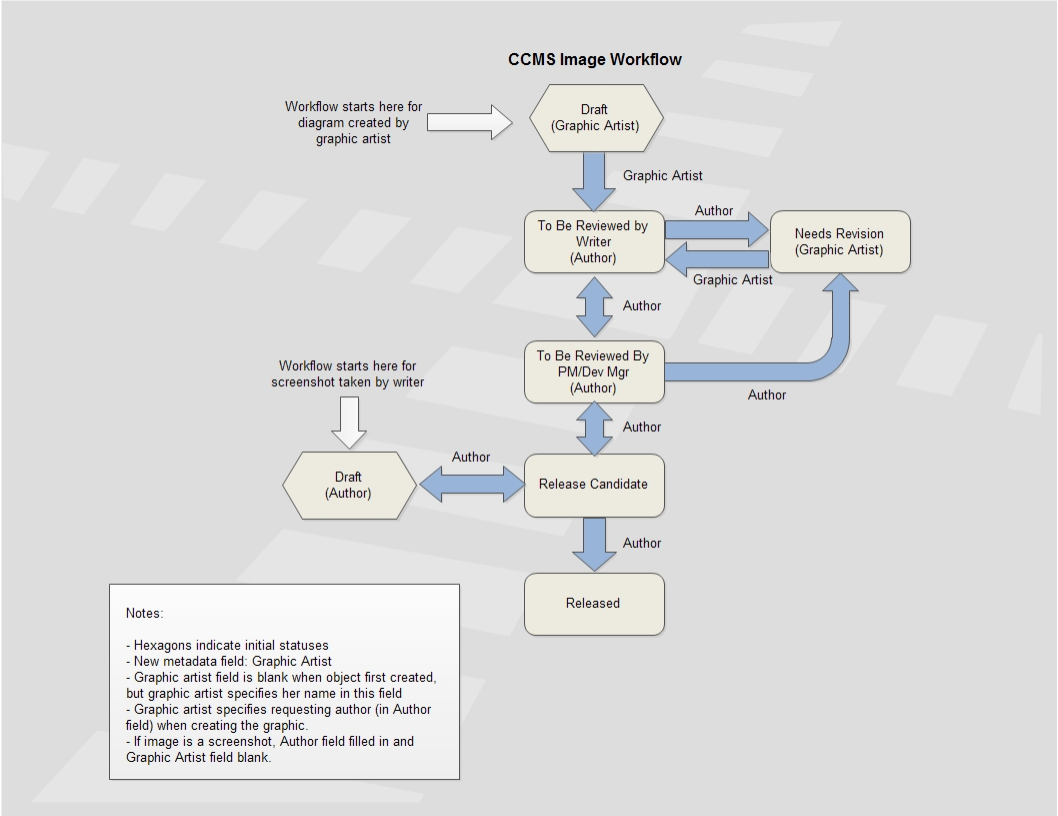
- In the PDF output, zoom in.
- In the HTML output, right-click the diagram and open it in a new tab or window.

## Overview of the Workflow for an Image

The workflow for an image tracks the status of the image from creation to release.

This workflow diagram for images includes the following information:

- Each valid status for an image in a rounded box.
- The user role assigned to complete an action at a particular status in parentheses.
- Status transitions in blue arrows labeled with the user role that is authorized to make the transition.
- Starting status in white arrows.




**Tip:** You can enlarge the diagram in one of the following ways:

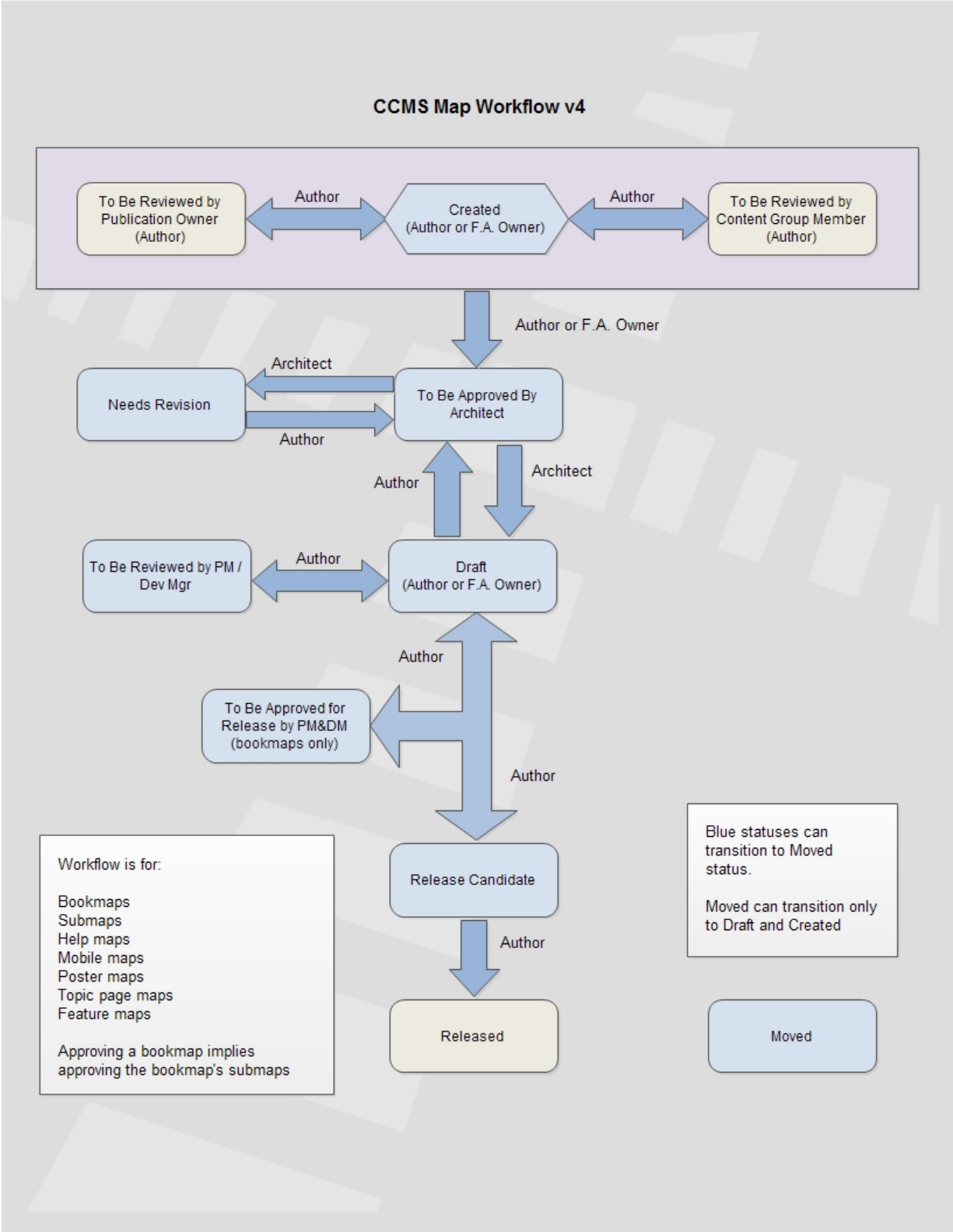
- In the PDF output, zoom in.
- In the HTML output, right-click the diagram and open it in a new tab or window.

## Overview of the Workflow for a Map

The workflow for a map tracks the status of the map from creation to release.

This workflow diagram for maps includes the following information:

- Each valid status for a map.
- The user role assigned to complete an action at a particular status in parentheses.
- Status transitions in blue arrows labeled with the user role that is authorized to make the transition.


CCMS Map Workflow v4
To Be Reviewed by Publication Owner (Author)
Author
Created (Author or F.A. Owner)
Author
To Be Reviewed by Content Group Member (Author)
Author or F.A. Owner
Architect
Needs Revision
To Be Approved By Architect
Author
Author
Architect
To Be Reviewed by PM / Dev Mgr
Author
Draft (Author or F.A. Owner)
Author
To Be Approved for Release by PM&DM (bookmaps only)
Author
Workflow is for:
Bookmaps
Submaps
Help maps
Mobile maps
Poster maps
Topic page maps
Feature maps
Approving a bookmap implies approving the bookmap's submaps
Release Candidate
Author
Released
Blue statuses can transition to Moved status.
Moved can transition only to Draft and Created
Moved

**Tip:** You can enlarge the diagram in one of the following ways:

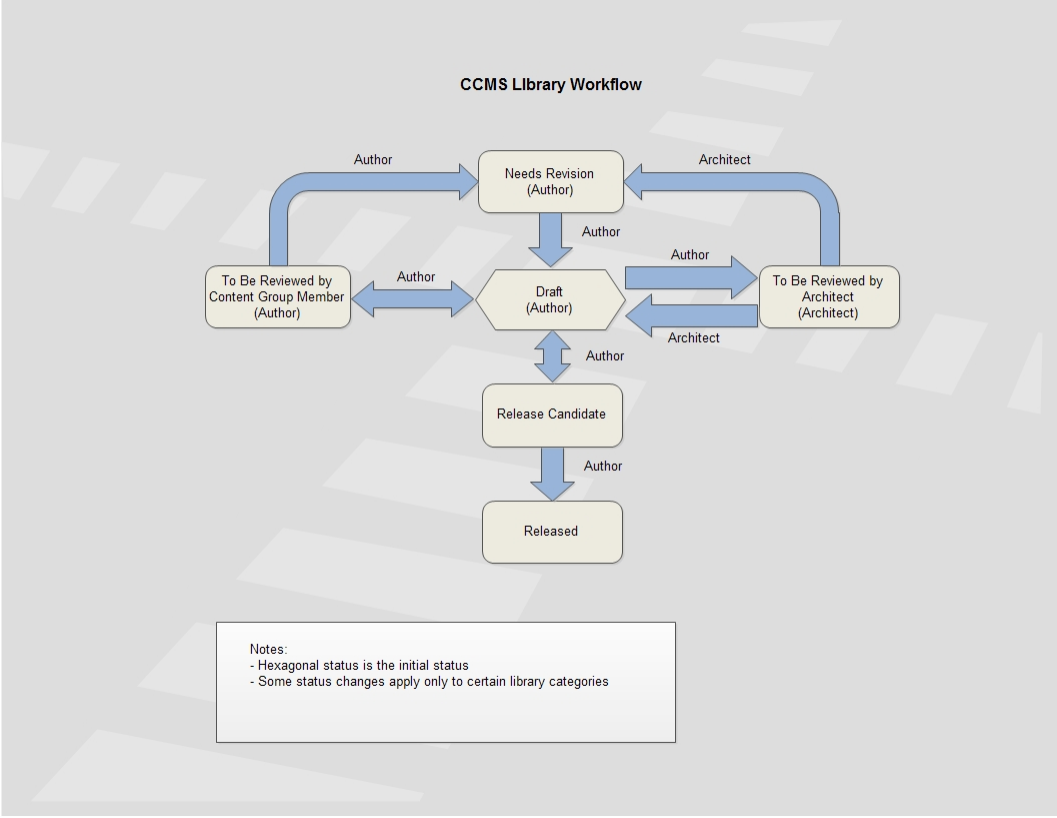
- In the PDF output, zoom in.
- In the HTML output, right-click the diagram and open it in a new tab or window.

## Overview of the Workflow for a Library

The workflow for a library tracks the status of the library from creation to release.

This workflow diagram for libraries includes the following information:

- Each valid status for a library in a rounded box.
- The user role assigned to complete an action at a particular status in parentheses.
- Status transitions in blue arrows labeled with the user role that is authorized to make the transition.




**Tip:** You can enlarge the diagram in one of the following ways:

- In the PDF output, zoom in.
- In the HTML output, right-click the diagram and open it in a new tab or window.

## Overview of the Workflow for a Database Feature Submap

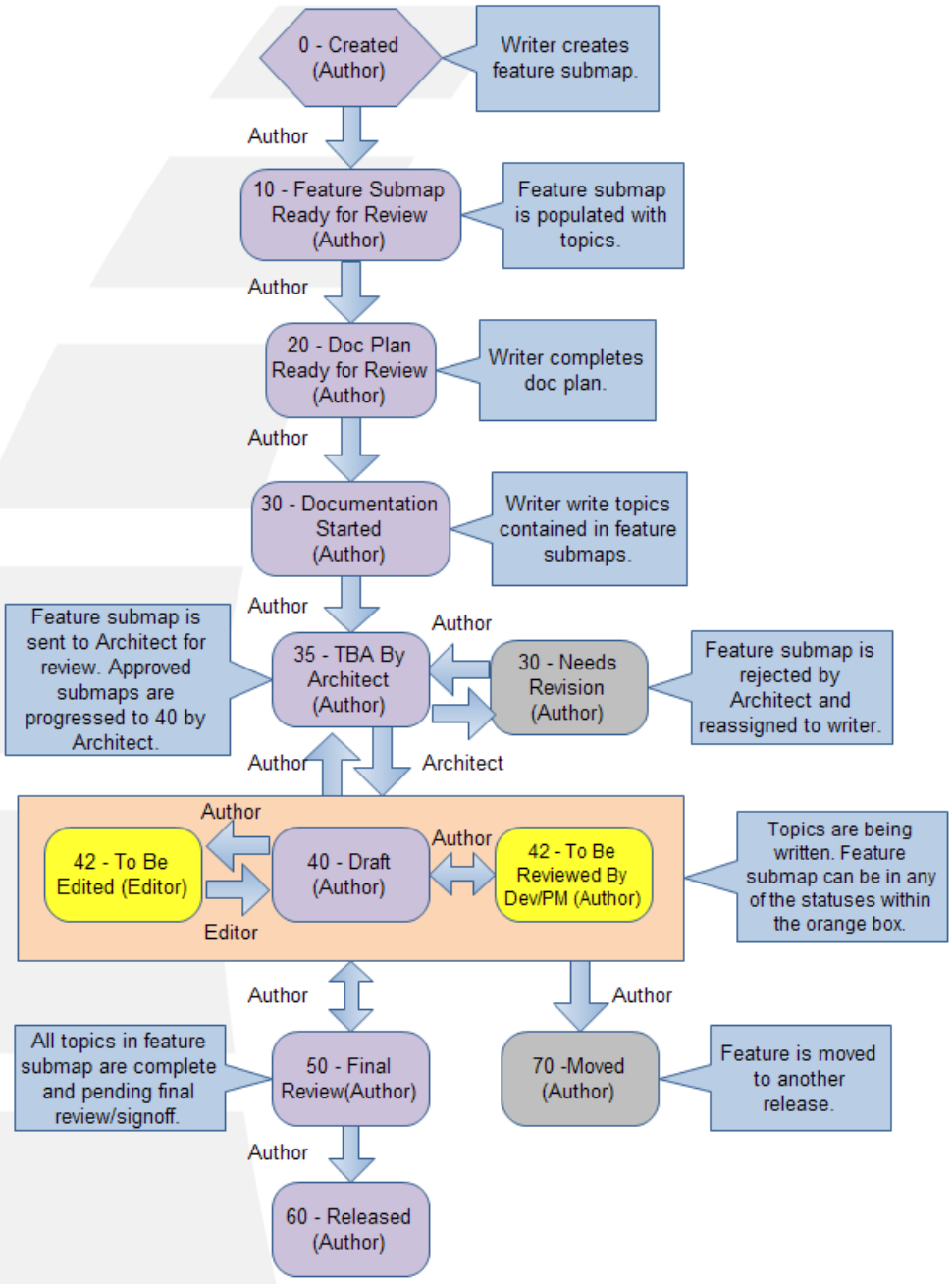
The workflow for a database feature submap tracks the status of the database feature submap from creation to release.

This workflow diagram for database feature submap includes the following information:

- Each valid status for a database feature submap.
- The user role assigned to complete an action at a particular status in parentheses.
- Status transitions in blue arrows labeled with the user role that is authorized to make the transition.
- Brief descriptions of some of the statuses.

The legend in the workflow diagram contains more information about the diagram.

**CCMS Feature Submap Workflow V.1**




**Legend**

- Statuses shown in violet are required statuses.
- Statuses shown in yellow are recommended.
- Statuses shown in gray are not required, but are possible statuses.
- The orange box contains additional statuses numbered 43 thru 48, which are not shown in the diagram. These statuses reflect the review statuses in ST Project and are optional statuses.
- The user role shown in parentheses for each of the statuses indicates the user role to which the topic is assigned for the given status.
- The user role next to the arrows indicates the user role who can initiate the corresponding status transition.
- Numbers in statuses correspond to ST Project doc mapping status values. For statuses that do not have corresponding ST Project status values, an arbitrary number is used to signify progression in the workflow.

**Tip:** You can enlarge the diagram in one of the following ways:

- In the PDF output, zoom in.
- In the HTML output, right-click the diagram and open it in a new tab or window.

## Viewing Your Inboxes

Objects that are related to you, such as objects that you own or objects that require your attention, are organized in the inboxes in the Browse Repository dialog box.

In your inboxes you can view objects organized according to your various roles in the workflow of a publication, such as Author or Publication Owner. For example, if one of your roles in the workflow of the publication is that of author, then objects of which you are the Author are displayed in the **Author-DB** and **Author (All Statuses)** folders in your inbox.

1. Open the Browse Repository dialog box.
2. Click the **Inbox** tab.

   A list of folders displays in the left pane of the window.
3. Select a folder to view that list of objects that are appropriate for the folder.
4. Use the **Objects of type** drop-down to filter objects by a specific type, such as modules.

**See Also:**

About Workflows
: Using workflows, you can send topics and illustrations to others in your group for authoring, reviewing, translating, or any other task required to complete the work.

## Keeping the Workflow Moving

When an object you need to work on is assigned to an architect, you cannot modify the object. To keep the workflow moving, you can contact the architect assigned to the object or a different architect.

1. Contact the architect that is assigned the object currently, and ask the architect either to provide feedback or approve the object.
2. If the architect that is assigned the object is not available, then contact another architect, and ask the architect either to provide feedback or approve the object.

**See Also:**

About Workflows
: Using workflows, you can send topics and illustrations to others in your group for authoring, reviewing, translating, or any other task required to complete the work.

## Writer's Daily Workflow

Each day writers should check their inboxes for objects that require their attention, work on appropriate objects, and change the statuses of objects when necessary.

The objects you own move through a workflow from creation to release. The workflow helps writers, architects, and managers track the progress of individual objects and an entire release. Writers usually perform the following tasks every day to keep the objects they own moving through this workflow:

- Check your "Author - DB" inbox for objects that require your attention.

  For example, a functional area owner might assign new, empty topics to you. Typically, these are topics that must be written to cover a new feature. When you start to add content to a new topic, set the topic's status to "30 - Being Written."

  For another example, an architect might review a topic that you wrote and provide feedback about required changes. The architect sets the status to "30 - Needs Revision" so that you know that there is feedback in the topic. Architects usually provide feedback in the form of comment tags in a topic.

- Check out objects to write new content or revise existing content.

  When you check out a topic, it opens in Arbortext Editor. You can add content to the topic or make changes based on feedback from an architect.

- Change the statuses of objects to move them through the workflow.

  For example, when an object, such as a topic, is ready for review by an architect, change the object's status to "35 - TBA by Architect" so that the topic appears in the appropriate architect's inbox.

**See Also:**

About Workflows
: Using workflows, you can send topics and illustrations to others in your group for authoring, reviewing, translating, or any other task required to complete the work.

Viewing Your Inboxes
: Objects that are related to you, such as objects that you own or objects that require your attention, are organized in the inboxes in the Browse Repository dialog box.

# 5

# SDL LiveContent Architect and ST Project

SDL LiveContent Architect can track the feature numbers from ST Project as part of the metadata for topics.

You can use Publication Manager to enter one or more feature numbers for each topic and search for topics that cover specific features. You can also indicate when features have been deprecated or desupported.

## About Feature Maps

A feature map is similar to an outline that contains all of the topics that are required to document a specific feature.

A feature map provides a comprehensive list of the proposed topics that are required to document a feature. The content group for a feature creates this list of topics.

Feature maps can be reviewed and approved by the product manager for the feature. In this case, the feature map is published, and the output is an outline of the proposed topics that will cover the feature. After the feature map is approved by the product manager, the functional area owner places the individual topics into the appropriate bookmaps of publications, and the writers can start to work on the topics.

**See Also:**

Creating Feature Maps
Create a map in the repository, and add the topics that will cover a specific feature to the map.

## Creating Feature Maps

Create a map in the repository, and add the topics that will cover a specific feature to the map.

The topics that will be included in the feature map must exist in the repository.

1. In Publication Manager, from the **Tools** menu, select **Browse Repository**.
2. In the **Repository** tab, locate and select the folder that will contain the feature map.

   **Note:** Ensure that the content type of the selected folder is **Master Document**.

3. In the right pane, click the **New Object** button.
4. In the Select Template dialog box, under the tab for your group, select the **Feature Map** template. For example, if you are a database writer, then select the template under the **Database Maps** tab.
5. In the Add Object dialog box, fill in the metadata fields.

6. Click the **OK** button.
7. In the right pane of Publication Manager, right-click the new feature map, and select **Check Out**.

   Arbortext Editor opens, showing the new feature map. The repository window also remains open.
8. In the repository window, locate a topic that should be part of the feature map and drag it to a valid location in the feature map in Arbortext Editor.

   **Note:** Repeat this step for each topic that should be added to the feature map.
9. Check in the feature map to save your changes.

**See Also:**

About Feature Maps
: A feature map is similar to an outline that contains all of the topics that are required to document a specific feature.

Metadata for Maps
: Metadata for maps aids in searching for maps and tracking a map's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

Checking In Objects
: When you finish editing an object, you must check it back into the Repository to upload the latest data and to unlock the object, which makes it available for checkout by other users.

## Entering Feature Numbers for Topics

You record feature number metadata at the topic level, and the feature number metadata is searchable in the repository.

When a topic covers one or more features, the feature numbers must be part of the metadata for the topic. This metadata enables writers, managers, and others to identify the topics associated with a specific feature by searching on the feature number in the repository.

1. In Publication Manager, from the **Tools** menu, select **Browse Repository**.
2. In the **Repository** tab, locate and expand the folder that contains the topic.
3. Right-click the topic, and select **Properties**.
4. Click the **Version** tab.
5. Enter the feature number in the **Feature #** field.

   If the topic covers multiple features, then enter a comma-delimited list of feature numbers.
6. Click **OK** to save your changes.

**See Also:**

Searching for Topics that Cover a Feature
: Feature number metadata is recorded at the topic level. You can perform a search to identify the topics associated with a specific feature or features by searching on the feature number in the repository.

## Searching for Topics that Cover a Feature

Feature number metadata is recorded at the topic level. You can perform a search to identify the topics associated with a specific feature or features by searching on the feature number in the repository.

1. In Publication Manager, from the **Tools** menu, select **Browse Repository**.
2. Click the **Search** tab.
3. Open the **Version** section.
4. Enter the feature number in the **Feature #** field.

   To search for topics that cover multiple features, enter a comma-delimited list of feature numbers. When you enter multiple features, the search uses the equivalent of an `AND` operator in a Boolean search. Therefore, only topics that cover all of the features listed are returned by the search.
5. Click **Search**.

   Search results appear in the right pane.

**See Also:**

Entering Feature Numbers for Topics
: You record feature number metadata at the topic level, and the feature number metadata is searchable in the repository.

## Deprecating and Desupporting Features

Topics that document deprecated and desupported features must proceed through a workflow.

Deprecated features might be desupported in a future release of the product. Desupported features are no longer available or supported.

Typically, a feature goes through a process of deprecation and desupport when a different, new feature replicates the feature's functionality. Documentation tracks this process by noting the features that are deprecated and desupported in each release. Each topic that covers these features must proceed through a workflow that deprecates and later desupports the topic.

Before a topic can be deprecated, the topic must be at 60 — Released status.

Before a topic can be desupported, the topic must be at 72— Deprecated status.

1. In Publication Manager, locate and select the topic.
2. Right-click the topic and select **Properties**.
3. In the Properties dialog box, click the **Workflow** tab.

4. In the **Status** list, select **71 — To Be Approved For Deprecation By VP PM**.
5. Click **OK**.
6. Wait for the product manager to change the topic's status to **72 — Deprecated**.

   At this point, the topic is deprecated.
7. To begin the workflow to desupport the topic, locate and select the topic in Publication Manager.
8. Right-click the topic and select **Properties**.
9. In the Properties dialog box, click the **Workflow** tab.
10. In the **Status** list, select **73 — To Be Approved For Desupport By VP PM**.
11. Click **OK**.
12. Wait for the product manager to change the topic's status to **74 — Desupported**.

    At this point, the topic is desupported.

# 6

# Creating and Managing Publications in Architect

Create and manage publication using SDL LiveContent Architect. Management tasks include adding master documents to publications, adding topics to publication, adding output formats to publications, and publishing publications.

## Overview of the Publication Lifecycle

A publication goes through a lifecycle that includes creation, management, publication, and release.

After a publication is created, writers can add, modify, and remove objects such as topics, and the publication owner can manage the publication's metadata and maps. Writers document new features, edit existing content, and correct documentation bugs until the document is final for a specific product release. At that point, the publication owner publishes the final output for the release and releases the publication.

Releasing a publication freezes its baseline automatically. Therefore, before it can be can modified for the next release, the publication owner must create a new version of the publication or create a new publication based on the publication.

A publication owner usually creates a new version of a publication when a new revision of the publication's part number has been ordered (for example, moving from –01 to –02 in the part number).

A publication owner usually creates a new publication based on the released publication when a new publication part number has been ordered for a new product release. A new part number is required when a new release of the product includes new features that must be documented in the publication.

**See Also:**

Creating a Publication
: You create a publication to assemble the content in a document for output.

Creating a New Version of a Publication
: A new version of a publication has its own properties and baseline, which are distinct from the properties and baselines of the other versions of the same publication.

Creating a New Publication from an Existing Publication
: A new publication created from an existing publication gets a new GUID. It also has its own properties and baseline.

## Creating a Publication

You create a publication to assemble the content in a document for output.

Before creating a publication, the metadata for the publication must be defined. For example, you must know its title, doc ID, and part number.

The Publications folder for the new publication must exist in the repository. This folder must be created in the following folder in the repository:`/General/Oracle Database/Database Publications/`*`DOCID — document_name`*`/db_`*`release_number_part_number`*`/`.

Substitute the following information for the variables in the path:

- Enter the five letter document ID for *DOCID* (for example, ADMIN).
- Enter the name of the document for *document_name* (for example, *Oracle Database Administrator's Guide*).
- Enter the release number for *release_number* (for example, 12.1.0.2)
- Enter the part number of the document for *part_number* (for example, e41484_06).

> **Note:** This folder structure is for the Database documentation. Other documentation might have a different folder structure.

Here is a full sample path for a Publications folder: `/General/Oracle Database/Database Publications/ADMIN — Oracle Database Administrator's Guide/db_12.1.0.2_e41484_06/Publications/`.

1. Open Publication Manager.
2. Click the **New Publication** button, or select **Publication** > **New**.
3. In the Select Template window, ensure that **Empty Publication** is selected, and click **Next**.
4. In the Select Target Folder window, locate and select the folder that will contain the publication.
5. Click **Next**.
6. In the Add Publication dialog box, enter the metadata for the new publication.

7. Save the publication by clicking the **Save** button, or by selecting **Publication** > **Save**.

To define the table of contents for the new publication, add a master document to it.

**See Also:**

Overview of the Publication Lifecycle
: A publication goes through a lifecycle that includes creation, management, publication, and release.

Metadata for Publications
: Metadata for publications aids in searching for publications and tracking a publication's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

Adding a Master Document to a Publication
: A master document is an XML file that defines the relationships between the topics in a publication. You add a master document to a publication to define the equivalent of the publication's table of contents.

Creating a New Publication from an Existing Publication
: A new publication created from an existing publication gets a new GUID. It also has its own properties and baseline.

Adding Output Formats to a Publication
: To produce output for a publication, add output formats, such as PDF and XHTML.

## Creating a New Version of a Publication

A new version of a publication has its own properties and baseline, which are distinct from the properties and baselines of the other versions of the same publication.

A publication owner usually creates a new version of a publication only when the current version of the publication is released and a new revision of the publication's part number has been ordered (for example, moving from –01 to –02 in the part number).

---

**Note:** When the publication owner orders a new part number instead of a revision, the publication owner creates a new publication based on the existing publication. In this case, the publication owner does not create a new version of the publication.

---

1. In Publication Manager, open the publication by checking it out.
2. In the tree view of Publication Manager, right-click the publication and select **New Version**.

   The publication is at the top level in the tree view.
3. In the Add Publication Output dialog box, enter the publication output metadata in the **Publishing Options** tab.

   Ensure that you enter the correct information in the **Part #** and **Print Date** fields.
4. Click **OK**.

Publication Manager refreshes with the newly created version of the publication selected in the right pane.

**See Also:**

Metadata for Publications
: Metadata for publications aids in searching for publications and tracking a publication's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

Overview of the Publication Lifecycle
: A publication goes through a lifecycle that includes creation, management, publication, and release.

Creating a New Publication from an Existing Publication
: A new publication created from an existing publication gets a new GUID. It also has its own properties and baseline.

## Creating a New Publication from an Existing Publication

A new publication created from an existing publication gets a new GUID. It also has its own properties and baseline.

When you save a publication from an existing publication, the new publication's structure and contents are the same as those in the publication from which it was saved. A publication owner usually creates a new publication using this method when the current version of the publication is released and a new part number is required for the publication. A new part number is required when a new release of the product includes new features that must be documented in the publication.

---

**Note:** When the publication owner orders a revision of an existing part number instead of a new part number, the publication owner creates a new version of the publication. In this case, the publication owner does not create a new publication from an existing publication.

---

The Publications folder for the new publication must exist in the repository. This folder must be created in the following folder in the repository:`/General/Oracle Database/Database Publications/`*`DOCID — document_name/`*`db_`*`release_number_part_number/`*.

Substitute the following information for the placeholders in the path:

- Enter the five letter document ID for *DOCID* (for example, ADMIN).
- Enter the name of the document for *document_name* (for example, *Oracle Database Administrator's Guide*).
- Enter the release number for *release_number* (for example, 12.1.0.2)
- Enter the part number of the document for *part_number* (for example, e41484_06).

Here is a full sample path for a Publications folder: `/General/Oracle Database/Database Publications/ADMIN — Oracle Database Administrator's Guide/db_12.1.0.2_e41484_06/Publications/`.

1. In Publication Manager, open the existing publication.
2. Select **Publication** > **Save As**.

If you made changes, then you are prompted to save the changes. Click **Yes** to do so.

The Select Target Folder dialog box opens, showing the repository.

3. Locate and select the Publications folder that will contain the publication.
4. Click **Next**.
5. In the Add Publication dialog box, enter the metadata for the new publication.
6. Click **OK**.
7. In the Add Publication Output dialog box, enter the publication output metadata in the **Publishing Options** tab.

   Ensure that you enter the correct information in the **Part #** and **Print Date** fields.
8. Click **OK**.

Publication Manager refreshes and shows the new publication. The new publication shares the existing publication's master document.

If the new publication should not share a master document with the existing publication, then add a different master document to the new publication.

**See Also:**

Creating a Publication
: You create a publication to assemble the content in a document for output.

Overview of the Publication Lifecycle
: A publication goes through a lifecycle that includes creation, management, publication, and release.

Opening a Publication
: You open a publication to view its structure, modify it, view or modify its content, or publish it.

Adding a Master Document to a Publication
: A master document is an XML file that defines the relationships between the topics in a publication. You add a master document to a publication to define the equivalent of the publication's table of contents.

Creating a New Version of a Publication
: A new version of a publication has its own properties and baseline, which are distinct from the properties and baselines of the other versions of the same publication.

Metadata for Publications
: Metadata for publications aids in searching for publications and tracking a publication's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

# Creating Maps

Use maps to logically organize objects, including other maps and topics, in a publication.

Maps enable you to logically organize your publication. Bookmaps are the highest level map to which you can add submaps, and submaps enable finer grained organization within the publication. In this context, submaps are like chapters, and the sections that make up a typical chapter are the topics organized by the submap. Submaps can make managing the organization of documents easier. For example, you can easily move a submap to a different location in a master document or to a different master document. However, each map and submap must go through the workflow from creation to release.

A feature map is another type of map that you can use to organize topics that are created by different authors for a specific feature for a release.

1. Open the Browse Repository dialog box.
2. Navigate to the **Maps** folder of your publication.
3. Click the New Object button  , or right click in the right pane of the Browse Repository dialog box.
4. Click **New Object...**.
5. In the Select Template dialog box, under the tab for your group, select the appropriate map template for your map. For example, if you are a database writer, then select a template under the **Database Maps** tab.

6. Click the **Next** button.
7. Edit the properties for the map and click **OK**.

**See Also:**

Metadata for Maps
: Metadata for maps aids in searching for maps and tracking a map's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

Adding a Chapter to a Bookmap
: You can add a chapter to a book map directly, or you can add a submap that contains (or will contain) the topics in the chapter to the bookmap.

## Adding a Master Document to a Publication

A master document is an XML file that defines the relationships between the topics in a publication. You add a master document to a publication to define the equivalent of the publication's table of contents.

When you create a new publication, you add a master document to the publication to define its table of contents. This type of master document is also commonly referred to as a bookmap.

When you create a new publication from an existing publication, the new publication shares the existing publication's master document. If you do not plan to change the master document in the new publication, then it can share the existing publication's master document. However, in most cases, changes to the master document will be required in the new publication. For example, changes to the master document are required when you add, remove, or reorder chapters. In this case, you must add a new master document to the new publication.

Before you can add a master document to a publication, both the publication and the master document must exist.

1. In Publication Manager, open the publication.
2. If the publication has a master document that it should no longer use, then delete the master document:
   a. Right-click the master document and select **Delete**.

      The master document is directly below the publication in the tree view.

   b. In the Confirm Object Delete dialog box, click **Yes**.
3. In the tree view, right-click the publication's title at the top level, and select **Add** > **Master Document**.
4. In the Add Master Document window, locate and select the master document for the publication, and click **Add**.
5. Save the publication by clicking the **Save** button, or by selecting **Publication** > **Save**.

**See Also:**

Creating a Publication
: You create a publication to assemble the content in a document for output.

Creating a New Publication from an Existing Publication
: A new publication created from an existing publication gets a new GUID. It also has its own properties and baseline.

Opening a Publication
: You open a publication to view its structure, modify it, view or modify its content, or publish it.

## Adding Output Formats to a Publication

To produce output for a publication, add output formats, such as PDF and XHTML.

1. In Publication Manager, open the publication.
2. Click the **Output** tab.
3. Click the **Add** button at the upper left.
4. In the Select Output Format dialog box, select **InfoDev PDF**, and click **Next**.
5. In the Add Publication Output dialog box, specify the properties on the **Publishing Options** tab for the output.
6. Click the **OK** button.

   "InfoDev PDF" is listed as an available output format on the **Output** tab.
7. Click the **Add** button at the upper left.
8. In the Select Output Format dialog box, select **InfoDev XHTML**, and click **Next**.
9. In the Add Publication Output dialog box, specify the properties on the **Publishing Options** tab for the output.
10. Click the **OK** button.

    "InfoDev XHTML" is listed as an available output format on the **Output** tab.
11. Save the publication by clicking the **Save** button, or by selecting s**Publication** > **Save**.

**See Also:**

Publishing a Publication
: When you publish a publication, you generate output for the publication in either PDF or XHTML format.

Oracle Properties for Publishing Options
: When you add output formats for a publication, such as PDF and XHTML, you must specify publishing options for the output. The

publishing options include properties that are specific to Oracle publications.

Creating a Publication
: You create a publication to assemble the content in a document for output.

## Oracle Properties for Publishing Options

When you add output formats for a publication, such as PDF and XHTML, you must specify publishing options for the output. The publishing options include properties that are specific to Oracle publications.

***Table 6-1 Oracle Properties for Publishing Options***

| Oracle Property | Description |
|---|---|
| Part # | The part number of the publication, including the revision number. For example, A12345–01. |
| Print Date | The print date of the publication, which is usually the current month and year. For example, July 2014. |
| Beta draft | Select `none` if the output is for a production release.<br>Select `external` if the output is for a public beta release.<br>Select `internal` if the output is for a private beta release. |
| Auto-gen links | Select `all` to automatically generate links from parent topics to their child topics. Each link gets a link preview, which is text just below the link that is the short description from the target topic. Links are generated for parent topics at all levels (chapter, H1, H2, and so on).<br>Select `nofamily` to specify that no automatic links to child topics are generated. Use this value for most converted books, because they already have hand-coded internal TOCs in parent sections.<br>Select `none` if you do not want to use automatically generated links. |
| Brand | |
| Include draft-comments | Select the check box to include the content in `draft-comment` tags in the output. |

**See Also:**

Adding Output Formats to a Publication
: To produce output for a publication, add output formats, such as PDF and XHTML.

## Opening a Publication

You open a publication to view its structure, modify it, view or modify its content, or publish it.

You can open a publication in the following ways:

- Opening a publication from Publication Manager:
  1. In Publication Manager, from the **Publication** menu, select **Open**.
  2. In the **Open Publication** dialog box, locate and select the folder that contains the publication.
  3. In the right pane, select the publication.
  4. Click the **Open** button.
- Opening a publication from the Browse Repository dialog box:
  1. In Publication Manager, from the **Tools** menu, select **Browse Repository**.
  2. In the **Repository** tab, locate and select the folder that contains the publication.
  3. In the right pane, right-click the publication, and select the **Check Out** option.

  **Note:** Although you select the **Check Out** option, the publication is opened, not checked out.
- Opening a publication from the recent publications list:
  1. Open Publication Manager.
  2. From the Publication menu, select **Recent Publications** > ***publication_title***.

  **Note:** The publication must have been opened recently to appear in the recent publications list.

The publication opens in Publication Manager.

## Editing the Properties of a Publication

A publication's properties include information that is important for internal tracking, such as the publication's doc ID, and information that is part of the output, such as the copyright years.

A publication's properties include the following information:

- Title, type, and description
- Product name, doc ID, release number, and platform

- Owner and peer reviewer
- Copyright years

> **Note:** To modify a publication's part number, edit the publication's output properties.

1. In Publication Manager, from the **Tools** menu, select **Browse Repository**.
2. In the **Repository** tab, locate and expand the folder that contains the publication.
3. Right-click the publication, and select **Properties**.
4. Select the appropriate tab and edit the properties.
5. Click the**OK** button to save your changes.

## Adding a Chapter to a Bookmap

You can add a chapter to a book map directly, or you can add a submap that contains (or will contain) the topics in the chapter to the bookmap.

You have the following options when you add a chapter to a bookmap:

- Add the chapter to the bookmap directly.

  After adding the chapter, you can add topics to the chapter with topic refs. The topic refs are included directly in the bookmap. It is best to use this method if you are sure the chapter will not need to be moved or copied to a different publication. The advantage adding the chapter directly to the publication is that there are fewer submaps to maintain.

- Add a submap to the bookmap.

  After adding the submap, the chapter contains the topics in the submap. You can add topics to the submap with topic refs, and the topics are included in the chapter through the submap. The advantage of using a submap is that the submap can be moved or copied easily from one publication to another.

Before completing this task, the publication using the bookmap to which you want to add the chapter must exist.

1. With the publication open in Publication Manager, right-click the chapter that will immediately follow the chapter you are adding and select **Insert Before** > **Chapter**.

   The added chapter appears in the bookmap in Publication Manager before the chapter that was selected.

2. Right-click the added chapter and select **XML Attributes**.
3. In the Insert Values for "Chapter" dialog box, complete one of the following actions:

   - If you are adding the chapter to the bookmap directly, then enter the title of the chapter in the **Navigation title** field.
   - If you are adding a submap to the bookmap, then click the **Browse** button and insert the submap in the **HRef** field. The title of the submap appears in the **Navigation title** field.

The text in the **Navigation title** field is for metadata purposes only. Each chapter starts with an orientation topic, and the title of this orientation topic is the chapter title in the output.

4. Click the **OK** button.

You can add topics to the chapter with topic refs if necessary.

**See Also:**

Adding a Master Document to a Publication
: A master document is an XML file that defines the relationships between the topics in a publication. You add a master document to a publication to define the equivalent of the publication's table of contents.

Creating Maps
: Use maps to logically organize objects, including other maps and topics, in a publication.

Adding Topics to a Publication
: You add an existing topic to a publication by inserting a topic reference (topicref) to the topic in the publication's bookmap. The topicref points to the topic in the repository using its GUID.

# Managing Topics in a Publication

You manage the topics in a publication by adding topics to a bookmap or submap, selecting the version of a topic in a publication, and removing topics from a bookmap or submap. You accomplish these tasks in either Arbortext Editor or the tree view of the Publication Manager.

A bookmap assembles DITA topics into a publication. A bookmap contains pointers (topicrefs) to topics, and these topicrefs are organized hierarchically like a table of contents. A bookmap can contain submaps, and each publication references a single bookmap.

**Note:** A bookmap differs from a regular DITA map because it includes elements for front matter and back matter, and it includes chapter, part, and appendix elements.

## Adding Topics to a Publication

You add an existing topic to a publication by inserting a topic reference (topicref) to the topic in the publication's bookmap. The topicref points to the topic in the repository using its GUID.

The same topic can be added to multiple publications. In this case, each publication points to the same topic in the repository with a topicref. If the topic is modified, then the changes are reflected in all of the publications that include the topic.

**Note:** When a topic is added to a publication, you can select which version of the topic to use in the publication, and different versions of a topic can contain different content.

1. With the publication open in Publication Manager, complete one of the following actions in the tree view:
   - Locate and select the topic or submap that will contain the topic you are adding.
   - Locate and select the topic that will immediately follow the topic you are adding.
2. Complete one of the following actions:
   - If the currently selected object will contain the topic you are adding, then right-click the object and select **Add Within** > **Topic Ref**.
   - If the currently selected topic will immediately follow the topic you are adding, then right-click the topic and select **Insert Before** > **Topic Ref**.
3. In the Insert Values for 'Topic Ref' dialog box, click **Browse**.
4. In the Insert Link window, locate and select the topic you want to add.
5. Click **Insert**.
6. In the Insert Values for 'Topic Ref' dialog box, click **OK**.
7. If necessary, drag the topic to the correct location in the table of contents.

**See Also:**

Creating Topics from Publication Manager
: Create a topic from the Browse Repository dialog box in Publication Manager.

Creating Topics from Arbortext Editor
: Create a topic from the **SDL LiveContent** menu in Arbortext Editor as an alternative to the Browse Repository dialog box in Publication Manager.

Opening a Publication
: You open a publication to view its structure, modify it, view or modify its content, or publish it.

Selecting an Object Version
: Several versions of an object can exist. Select the correct version of each object in a publication. The selected version is included in the publication's baseline.

## Removing Topics from a Publication

You remove a topic from a publication by deleting its topic reference (topicref) in the publication's bookmap. Removing a topic from a publication does not delete the topic in the repository.

1. In Publication Manager, open the publication.
2. In the tree view of Publication Manager, locate the topic you are removing.
3. Right-click the topic, and select **Delete**.
4. In the Confirm Object Delete window, click **Yes** to confirm deletion or **No** to cancel the deletion.

# Saving and Closing a Publication

You save the publication that is opened in Publication Manager currently to save modifications to the publication in the repository. You close the publication when you finish making changes.

Before closing a publication, ensure that no objects in the publication are checked out in Arbortext Editor.

1. In Publication Manager, click the **Save** button, or select **Publication** > **Save**.
2. If no additional changes are required, then close the publication by selecting **Publication** > **Close**.

---

**Note:** If an object in the publication is checked out in Arbortext Editor, then you will see an error when you try to check in the object in Publication Manager.

---

# Managing Publication Baselines

A baseline lists the versions of the objects that are used in a publication. Each publication has exactly one baseline. You manage baselines with the **Baseline** tab in Publication Manager.

**See Also:**

Selecting an Object Version
: Several versions of an object can exist. Select the correct version of each object in a publication. The selected version is included in the publication's baseline.

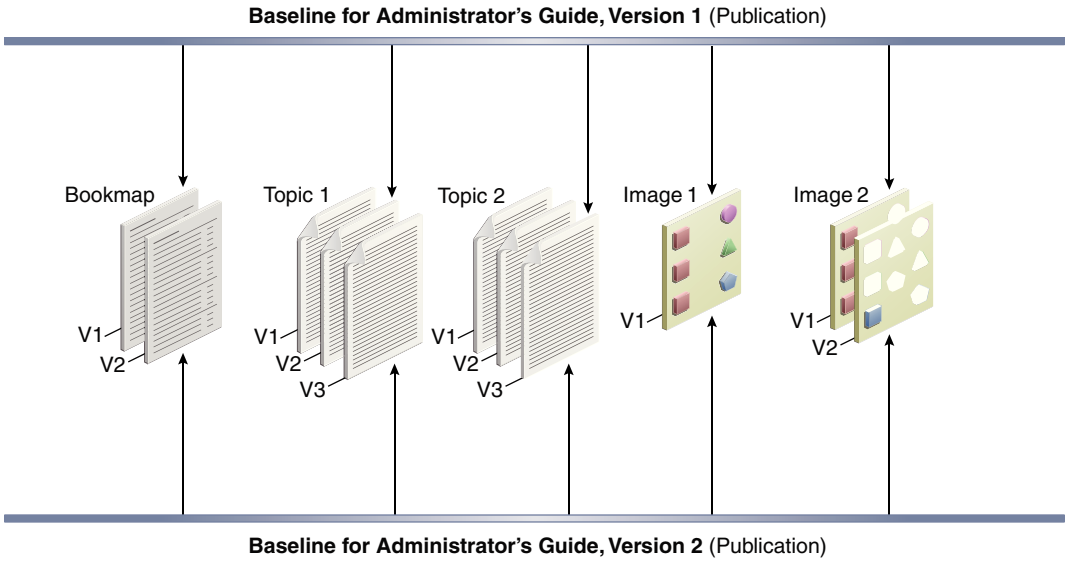
## About Baselines

A baseline lists the version of each object used in a publication. When a publication is open in Publication Manager, you can view and manage its baseline by clicking the **Baseline** tab.

The objects in a baseline include topics, graphics, maps, and variable libraries. Bookmaps reference objects by their GUIDs, and all versions of an object have the same GUID.

A baseline is a property of its publication. Because each publication has its own baseline, baselines enable each publication to reference different versions of the same object or the same version of an object. For example, two publications that use the same bookmap can use different versions of topics. In Publication Manager, you can select which version of an object to use in a publication. This selection is recorded in the baseline.

The following illustration shows how the baselines for different versions of the same publication can reference different versions of objects, such as bookmaps, topics, and images.




When a publication is released, its baseline is frozen. You cannot add objects to a publication with a frozen baseline, and you cannot modify version selections of objects. However, you can modify conditional text settings, change the output formats, and publish output. When you create a new version of a publication, the publication gets a new baseline, and you can modify this new baseline until it is frozen.

## Autocompleting a Baseline

When a publication has incorrect versions for any of its objects, autocompleting the publication's baseline enables you to select the version of each of these objects.

When you autocomplete the baseline of a publication, you can select the latest available version, the released version, or the first version of objects with incorrect versions. You can also autocomplete a baseline using a baseline from another publication.

Autocompleting a baseline is useful for specifying the versions of multiple objects in one operation. You can autocomplete a baseline multiple times if necessary. After autocompleting a baseline, you can select specific versions for objects if necessary.

1. In Publication Manager, open the publication if it is not open by checking it out.
2. Click the **Baseline** tab.
3. Click **Autocomplete**.

   The Autocomplete dialog box appears.
4. Select an option:

| Option | Description |
|---|---|
| **Baseline** | Click **Select Baseline** and choose a baseline. For any object with a version selection that is different from the one used in the selected baseline, the version from the selected baseline is used for the object. |
| **Candidate for baseline** | Click **Select Baseline** and choose a baseline. For any object with a version selection that is different from baseline name in its |

| Option | Description |
|---|---|
| | **Candidate for baseline** property, the version from the baseline in the **Candidate for baseline** property is used for the object. |
| **Latest released versions** | For any object without the released version selected, the latest released version is selected for the object if one is available. |
| **Latest available versions** | For any object without the latest version selected, the latest version is selected for the object if one is available. |
| **First versions** | For any object without the first version selected, the first version is selected for the object. |

5. Click **OK**

### Verifying If a Newer Version of an Object Exists in a Baseline

You can verify if a newer version of an object exists in a baseline using the **Baseline** tab in Publication Manager.

If a newer version of an object exists, then you can examine the newer version to see if the baseline should use the newer version of the object.

1. In Publication Manager, open the publication if it is not open.
2. Click the **Baseline** tab.
3. Inspect the **Newer version** column.

> **Note:** The **Newer version** column shows "Yes" if a newer version of an object exists.

## Publishing a Publication

When you publish a publication, you generate output for the publication in either PDF or XHTML format.

One or more output formats must exist for the publication. You add output formats during publication creation.

1. In Publication Manager, open the publication.
2. Click the **Output** tab.
3. Select the row for the output format that you want to publish. For example, **Infodev PDF**.
4. Check the build properties, and update them if necessary. For example, you might need to modify one or more of the following properties: `Part #`, `Print Date`, `Beta draft`, or `Auto-gen links`.
   a. Right-click the selected row, and select **Properties**.
   b. In the Properties dialog box, update the properties as needed.
   c. Click the **OK** button.
5. Click the **Publish** button.

If you are prompted to save the publication first, then save it.

6. Check the **Status** column.

   The status should change to `Pending` and then to `Publishing`. If there is an error, then attempt to correct it, and publish again. If you cannot correct the error, then contact an administrator.

7. To download the output after the **Status** column changes to `Draft`, select the row for the output format (such as **InfoDev PDF**), and click the **Download** button.

8. Click **Open** to view the published output.

**See Also:**

Adding Output Formats to a Publication
: To produce output for a publication, add output formats, such as PDF and XHTML.

## Running Workflow Reports with the Web Client

Workflow reports enable you to quickly understand the current status of all the objects that conform a publication at a given moment. You can generate them from the web interface.

1. In the Web Client, click the **Repository** tab.
2. In the Repository pane at left, expand the folder structure as necessary and click the folder where the publication is stored.
3. In the Objects pane, click the title of the publication from which you want to run a report.
4. In the Variables pane, click the **Workflow** button.
5. In the Workflow Management for Publication dialog box, select one of the following object version options:

| Version options | Description |
|---|---|
| Use baseline information only | All version information is taken from the baseline that is associated with this version of the publication. |
| Use the latest available versions | If there is no baseline, or the baseline does not contain version information for all objects, include the latest available version of each object. This action does not modify your baseline. |
| Use the latest released versions | If there is no baseline, or the baseline does not contain version information for all objects, include the latest released version of each object. This action does not modify your baseline. |

6. Click **OK**.

   The Workflow Report is generated in a new browser tab. It consists of three sections: the publication information, statistics, and details.

**Note:** You can download all the object information contained in the details section in a spreadsheet. Click the **Download Statistics Report** to download it in a comma-separated values (.csv) file.

**See Also:**

Managing Publication Baselines
: A baseline lists the versions of the objects that are used in a publication. Each publication has exactly one baseline. You manage baselines with the **Baseline** tab in Publication Manager.

# Releasing a Publication

You release a publication when it is final for a product release. The product release can be a beta release or a production release.

You release a publication by releasing one of its output objects.

Releasing a publication freezes its baseline automatically. Therefore, before you can modify the publication for the next release, you must create a new version of the publication or a new publication based on the publication.

Before you can release a publication, all of the publication's objects must be present and must be at "60 --Released" status.

The publication must have no invalid links, hyperlinks, or conref links.

At least one of the publication's output formats must be published and have "Release Candidate" status.

1. In Publication Manager, open the publication by checking it out.
2. Click the **Output** tab.
3. Select the output format with "Release Candidate" status.
4. Click the **Release** button.

**Note:** The publication is released when the status of the selected output formats is changed to "Released." Release both the PDF and XHTML output formats when the publication is final for a product release.

**See Also:**

Reusing Small Chunks of Content with Conrefs
: A content reference (conref) is a reference to a reusable element in a library or topic. Conrefs enable writers to share small chunks of content in multiple topics.

Creating Links
: You can create links to topics in the repository, to external documents, or within a topic.

## Deleting a Publication

You delete a publication when it is no longer needed. Deleting a publication does not delete the objects that make up the publication, such as maps, topics, and images.

1. In Publication Manager, from the **Tools** menu, select **Browse Repository**.
2. In the **Repository** tab, locate and expand the folder that contains the publication.
3. In the right pane, select the publication.
4. Right-click the publication, and select **Delete**, or click the **Delete** button.
5. In the Confirm Object Delete dialog box, click **Yes** to confirm the deletion.

# Part II

## The Arbortext Editor

# 7

# Getting Started with Arbortext Editor

Installing Arbortext Editor is the first step to set up your authoring environment. After you install Arbortext Editor, configure it to access the Information Development custom document types to use the InfoDev DITA element specializations.

---

**Note:** If you find any issue while setting up your environment, please send an email to the following help group: arborhelp_ww_grp@oracle.com

---

**See Also:**

About Arbortext Editor
: Information Development (InfoDev) has selected Arbortext Editor as the authoring tool to create and edit documents in DocBook and other XML languages. Arbortext Editor has an interface that facilitates creating structured documents, and fully supports topic-based writing.

## Installing the Arbortext Editor

Arbortext Editor Release 6.0 is required as your authoring software to achieve compatibility with other tools used in InfoDev, such as our Component Content Management System. Download the installer from the Development Systems Software Management (DSSM) site and, after installing the editor, obtain a floating license from our server.

---

**Tip:** Before downloading the Arbortext Editor installer:

- Make sure that an Arbortext Editor License has been allocated for your account. You should receive an email from pdit-licensing_ww@oracle.com confirming the license allocation.
- Ensure that you are using the latest JRE version. Go to My Desktop to automatically verify the version installed in your computer.

---

1. Download the Arbortext Editor Installer from the DSSM site:
   a. Access the Oracle DSSM by entering the following URL in your browser's address bar: `dssm.us.oracle.com`.
   b. Log in using your single sign-on credentials.
   c. On the main DSSM page, select "Other Software and Tools".
   d. Enter `PTC` in the Vendor field and click the **Search** button.

e. In the Results pane, locate the `Arbortext_Editor_US` product, version `6.0.0070`. Disregard any other entries that may be listed.

   If you are not based within the US then the product is called `Arbortext_Editor_Global`

| Vendor | Product | Version | Rights |
|---|---|---|---|
| PTC | Arbortext_Editor_Global | 6.0.0070 | |

> **Note:** The DSSM site is known for being inconsistent. If your search returned no results then click the **Reset** button. If a subsequent search (again, specifying PTC in the Vendor field) returns no results, verify that an Arbortext license has been allocated for your account.

f. Click the green arrow next to the previous entry.

> **Note:** Only download version 6.0.0070, which is a 32-bit Arbortext installation. You can install the 32–bit version on a 64–bit OS

g. After clicking the green arrow, DSSM displays the "Nearest Location" selection. Select the appropriate location using the drop-down menu.

h. Copy and save the license server name that appears in the message at the bottom of the page.

   The server information should be similar to the following:
   `7788@adc61101.us.oracle.com`

> **Note:** Use only the license server information displayed by DSSM or you may be violating our software license agreement.

i. Click **Download Product**.

   DSSM downloads an executable file. You need this file to start downloading the actual editor installer.

j. Locate the .exe file that you downloaded and run it.

   DSSM begins downloading the Arbortext Editor installer.

k. In the Save As dialog box, select the location in your system where you want to save the Arbortext Editor installer.

> **Note:** Verify that the name of the file corresponds to `MED-71692-CD-060_M070_Win32.zip`

l. Click **Save**.

2. Install the Arbortext Editor:

a. Locate the downloaded Zip folder in your system and extract it to a desired location.

b. Run the `Windows\setup.exe` file to launch the Arbortext Editor installer.

c. In the product selection step, select only the **Arbortext Editor 6.0** option.

> **Note:** Make sure to deselect all other check boxes. Some of the components require an additional license.

d. Click **Next**.

e. Continue the installation leaving the default values. A typical installation meets our authoring needs.

3. Obtain a floating license from our license server:

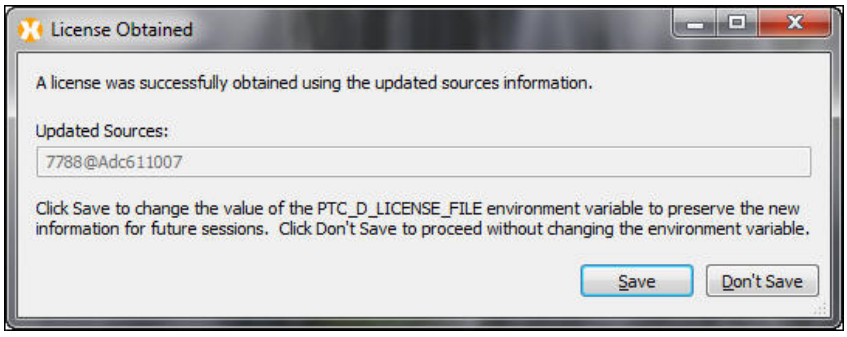
a. When the installation is complete, open Arbortext Editor.

A Licensing Error dialog box appears, it requires information about the license sources.

b. In the Update Sources field, enter the server information that you copied and saved in step 1.h.

c. Click the **Retry** button.

A confirmation dialog box indicates that a license has been successfully retrieved from the server:




d. Click **Save** to set the indicated environment variable.

Arbortext Editor is now installed in your computer.

## Installing the InfoDev Arbortext Extensions

After installing Arbortext Editor, add the InfoDev Arbortext Extensions developed for the Information Development organization. They include new features, DocGuru and InfoDev Preview, and our own document specializations.

Close any instances of Arbortext Editor before following the next procedure.

1. Download the InfoDev Arbortext Extensions installer:

a. Enter the following URL in your browser's address bar:

http://slc04lzk.us.oracle.com/arbortext-updater/InfoDevArbortextExtensionsInstaller.exe

**Note:** If prompted, login using your SSO credentials.

b. In the Opening InfoDevArbortextExtensionsInstaller.exe dialog box, click **Save File**.

2. If open, close the Arbortext Editor.
3. If you were a user of the InfoDev Customizations for Arbortext (now obsolete) then remove the APTCUSTOM environment variable:
   a. Click **Start**, and select **Run** from the menu.
   b. In the Run dialog box, enter `sysdm.cpl` and click **OK**.
   c. In the System Properties dialog box, select the **Advanced** tab, and click **Environment Variables**.
   d. In the Environment Variables dialog box, search for a variable named `APTCUSTOM` and select it.
   e. Click the corresponding **Delete** button.
   f. Click **OK**.
4. Install the InfoDev Arbortext Extensions:
   a. Locate the installer in your file system and run it.
   b. Accept the default path value and click **Install**.
   c. Click **Close** when the installation process is finished.
5. Verify that the extensions are now available within the Arbortext Editor:
   a. Start the Arbortext Editor.

      The InfoDev Arbortext Extensions installer automatically starts checking for updates.

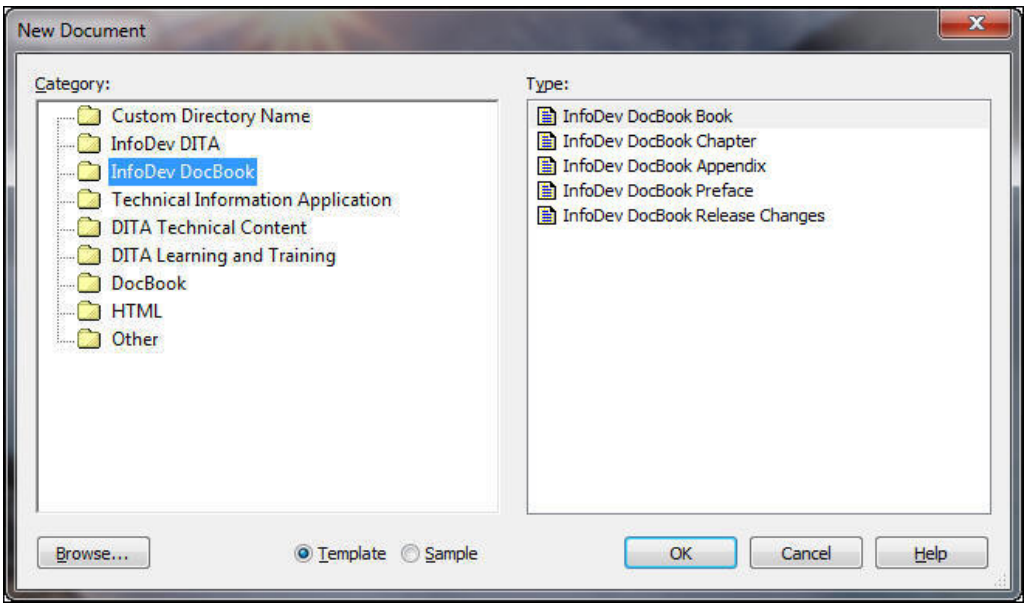
   b. Enter your SSO credentials and click **Log In.**

      **Note:** You can select the **Remember my credentials** option to avoid entering this information every time you start Arbortext Editor.

      The installer installs the latest version of the extensions and restarts the Arbortext Editor.

When Arbortext Editor opens again, two new menus are available, one DocGuru and one for InfoDev Preview. Additionally, InfoDev DocBook document type definitions are available when you create a new document. Figure 7-1 shows the New Document dialog box, it lists the customized InfoDev document types under the InfoDev DITA and InfoDev DocBook categories.

***Figure 7-1 The InfoDev DocBook Document Types Available in Arbortext Editor***




## Setting Arbortext as Your Default XML Editor

After installing the Arbortext Editor, along with the InfoDev Extensions, make sure to set Arbortext as your default editor for DITA documents.

1. In Windows Explorer, right-click the DITA file that you want to open.
2. In the context menu, click **Open With**.
3. In the Open With dialog box, do one of the following:
   - In the central pane, select the **Arbortext Editor for Windows** option.
   - If Arbortext Editor is not listed in the pane then browse for its executable file in your system:
     1. Click **Examine**.
     2. In the Open With dialog box, browse for the executable file and select it.

        **Note:** If you followed the typical installation then you can find the file in `C:\Program Files (x86)\PTC\Arbortext Editor\bin\x86`.

     3. Select **editor.exe** and click **Open**.

        The **Arbortext Editor for Windows** option is now included in the central pane of the dialog box.
4. Click **OK**.

Arbortext Editor opens the document. It is now set as the default editor for DITA files.

# Overview of Arbortext Editor Interface

The Arbortext Editor window provides the interface that you use to edit and modify your document. You can customize how the menus, toolbars, Edit pane area, and other components are sorted and displayed in the window according to your preferences.

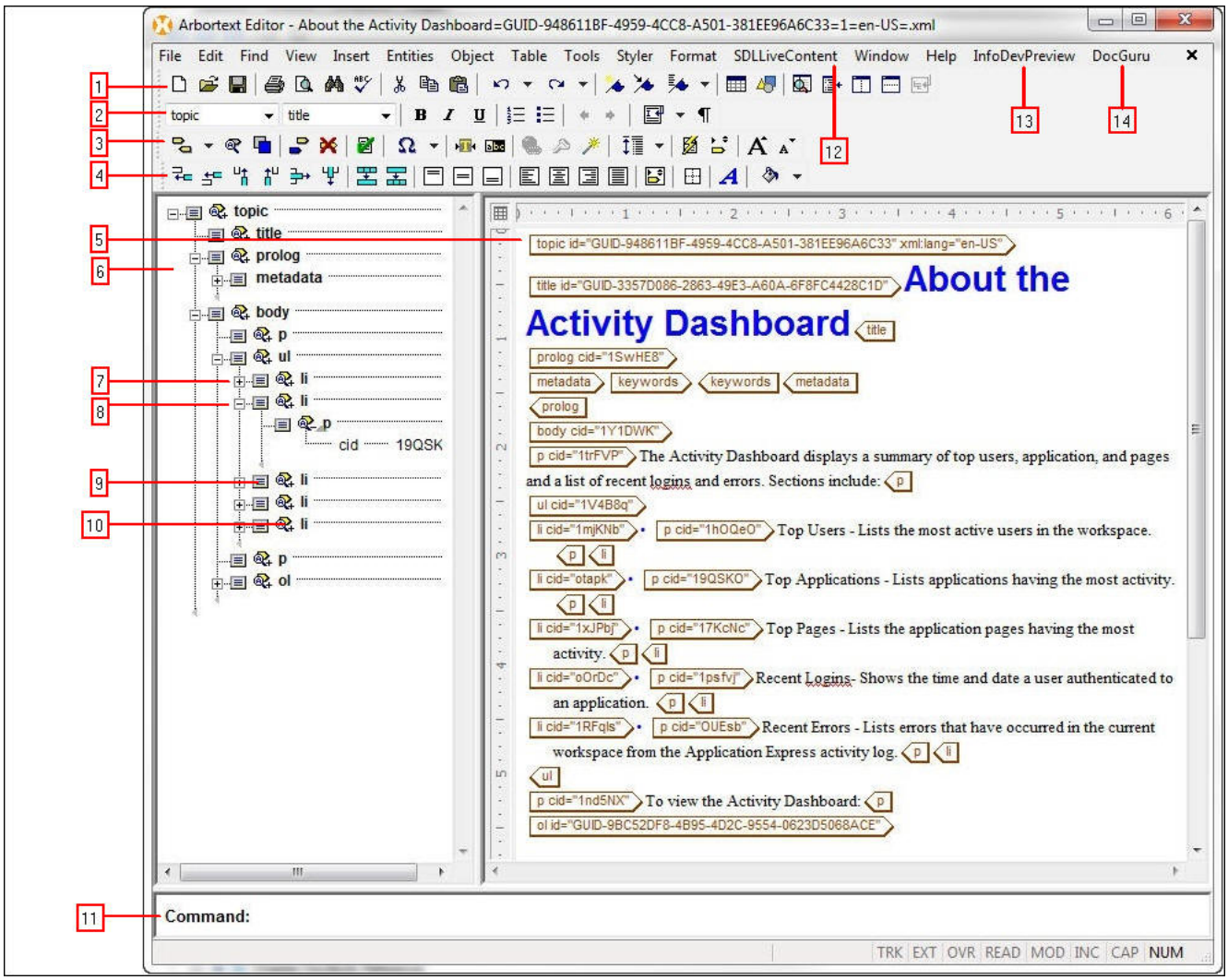
The Arbortext Editor window in Figure 7-2 shows the most common elements of the Arbortext Editor interface. Each element is labeled with a callout that is explained in the following list.

---

**tip:** To see all the interface elements in your editor, open a sample file by selecting **File** > **New** > **InfoDev DITA** > **InfoDev DITA Concept**. Choose the **Sample** option at the bottom of the dialog, select a **Type** from the list on the right, and click **OK**.

---

***Figure 7-2 The Arbortext Editor Window***




1. Edit toolbar. Groups buttons for recurring editing operations such as Open, Print, and Copy.

2. Application toolbar. Contains functions to format your documents: bold, italic, and underlining; promote and demote elements; and convert tags to lists.
3. Markup toolbar. Includes options useful when you work with markup such as inserting markup, modifying attributes, or applying profiles.
4. Table toolbar. Contains buttons to edit and format a table.
5. Edit view. A part of the Edit pane, this view shows the document with emphasis on its content. It is useful for most writing tasks.
6. Document map. A part of the Edit pane, this view displays a hierarchical outline of the document. It is useful for navigating the document and selecting elements.
7. Expand (shown in document map). Click the plus sign to display the contents and child elements of an element.
8. Collapse (shown in document map). Click the minus sign to collapse all contents of a tag, leaving a cleaner view.
9. Tag icon (shown in document map). Represents a tag or element in your document.
10. Modify attributes icon (shown in document map). Appears when one or more attributes of an element have edited values.
11. Command-line bar. Used to enter Arbortext Command Line (ACL) commands.

    **tip:** Display the command line by selecting the **Preferences** option from the **Tools** menu and then checking the **Command Line** box in the Window category.

12. SDLLiveContent menu: Includes all the Authoring Bridge features necessary to interact with the SDL server from Arbortext Editor.

    **Note:** The SDLLiveContent menu doesn't display in the interface until the InfoDev Arbortext Extensions.

13. InfoDevPreview menu: An InfoDev custom menu, it includes an option to visualize how the editor's buffer content (the document being edited in Arbortext editor) would appear in a document that follows the InfoDev style.
14. DocGuru menu: An InfoDev custom menu, it includes options that help writers converting content from DocBook to DITA.

# 8

# Customizing the Arbortext Editor Interface

Arbortext Editor provides a flexible authoring environment. The interface can be adapted in several ways to your writing preferences. For example, you can specify the view of the document or choose to display markup as full, partial, or no tags.

## Displaying Full Menus in Arbortext

Postinstallation tasks include displaying full menus and menu options in Arbortext Editor. The new options enable you to take full advantage of the product's capabilities.

Complete the following procedure to enable full menus:

1. From the **Tools** menu, select the **Preferences** option.
2. In the Preferences dialog box, select the **Window** category at the end of the list.
3. Make sure that the **Full Menus** check box is selected.
4. Click **OK**.

## Showing or Hiding Toolbars

The Edit, Markup, Table, and Application toolbars are included in Arbortext Editor to quickly invoke common operations. You can customize whether to display them or not.

1. In the **View** menu, click the **Toolbars** option.
2. Click the toolbar that you want to display or hide.

> **Note:** Alternatively, you can right-click beside any toolbar to quickly display the **Toolbars** submenu.

## Displaying the Document Map and Edit View

The document map and the edit view are included in Arbortext Editor as ways of visualizing the current document. The document map emphasizes the document structure and its elements in a graphical, hierarchical manner. In the edit view, the content of the elements is more evident; you may prefer to use this view to add to or edit the document.

1. Click the Edit pane.
2. Do one of the following:
   - From the **View** menu, select either **Edit View** or **Document Map**.

- In the **Edit** toolbar, click the **Document Map** button.

**Note:** Toggle between views by clicking the **Document Map** button more than once.

## Displaying Split and No Split View

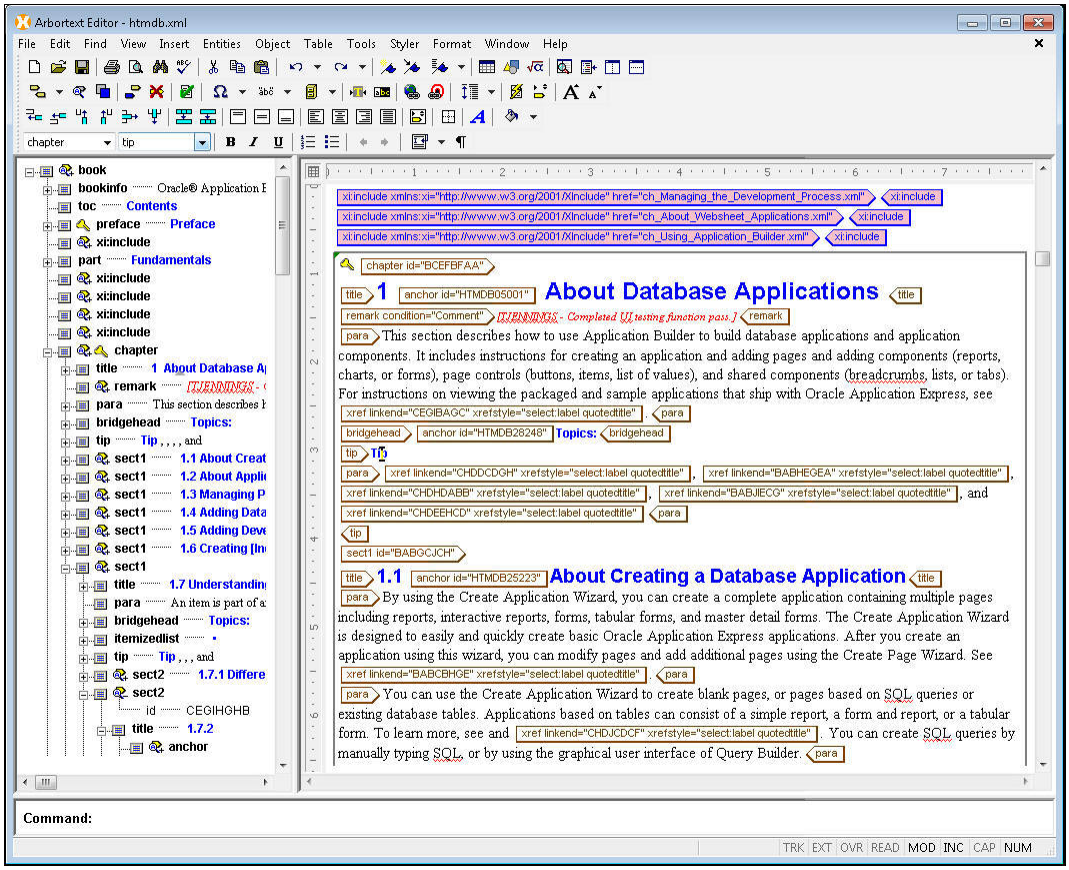
Split the Arbortext Edit pane vertically or horizontally to see more than one view of a given document concurrently. For example, splitting the pane vertically to show both the document map and edit view helps visualize both document content and structure; similarly, splitting the pane horizontally to show two edit views enables you to simultaneously see two parts of the same document.

To split the Edit pane, do one of the following:

- From the **Window** menu, select **No Split**, **Left-Right Split**, or **Top-Bottom Split**.
- In the **Edit** toolbar, click the **Left-Right Split** or **Top-Bottom Split** button.

**Note:** Clicking the button corresponding to the current split option displays only one view (no split).

***Figure 8-1 Edit Pane Split into Document Map and Edit View***




# Showing or Hiding Tags

In Arbortext Editor, markup tags are the building blocks of a structured document. Markup tags represent XML elements and attributes. You can choose how markup tags are displayed in the Edit view.

Arbortext Editor supports three options to show tags in a document:

- Full tags: Displays each element as an icon that includes its name along with its attributes and values.
- Partial tags: Shows the location of all starting and ending tags, but without the element name and attributes.
- No tags: Hides all tags.

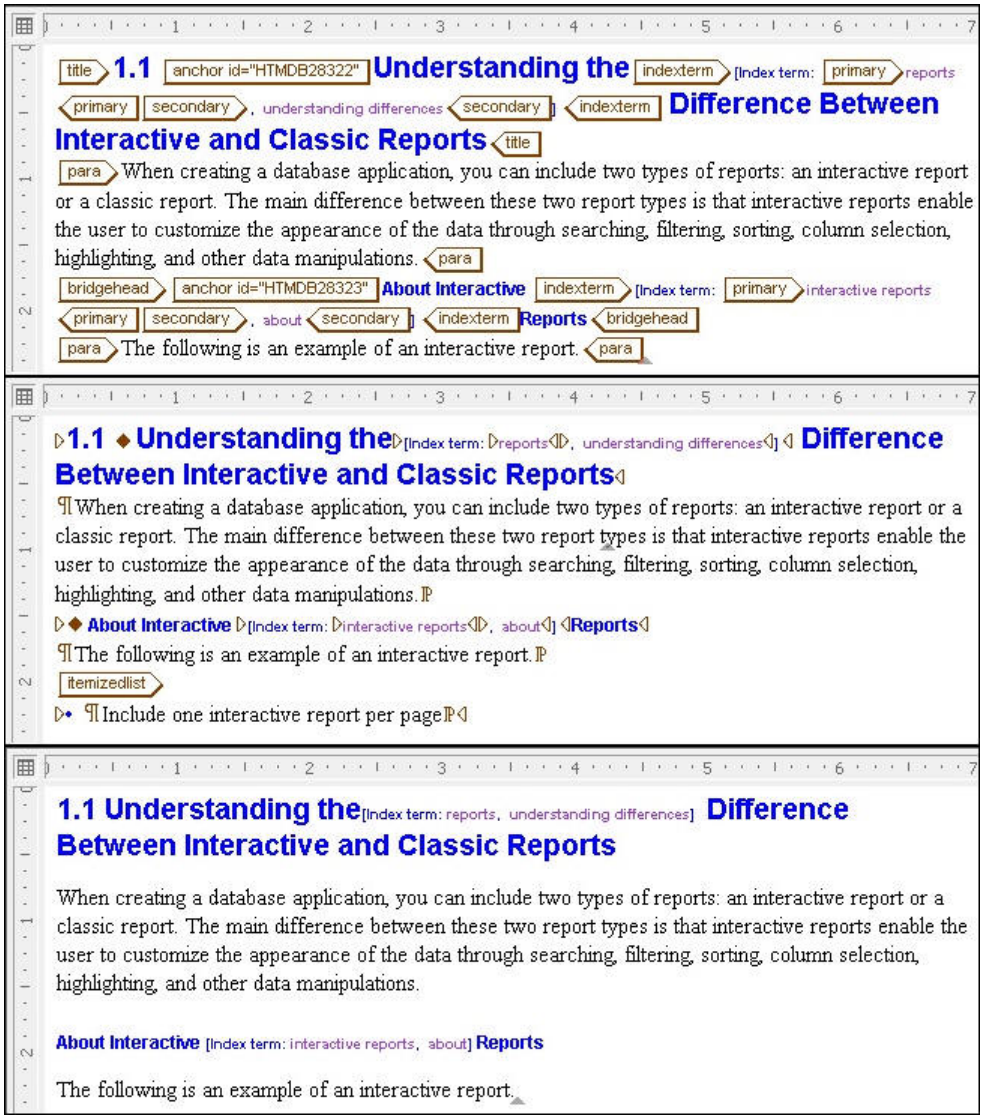
> **Note:** When you are writing a structured document, it can be easier to place the cursor if full or partial tags are displayed. Hiding all tags is useful when you are reviewing or editing documents. Figure 8-2 shows a piece of sample content displayed in different tag modes.

To change the way tags are displayed, do one of the following:

- From the **View** menu, select **Full Tags**, **Partial Tags**, or **No Tags**.
- From the **Markup** toolbar, click the **Tag Display** button.

Review and select the display mode that you want by repeatedly pressing this button.

***Figure 8-2 A Document Displayed in Full Tags, Partial Tags, and No Tags***



# 9

# Arbortext Editor Fundamentals

Fundamental skills for using Arbortext Editor include understanding the various cursors and working with elements and its attributes.

**See Also:**

About Arbortext Editor
: Information Development (InfoDev) has selected Arbortext Editor as the authoring tool to create and edit documents in DocBook and other XML languages. Arbortext Editor has an interface that facilitates creating structured documents, and fully supports topic-based writing.

## About the Arbortext Editor Cursor

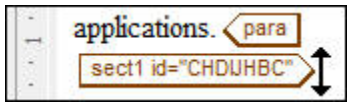
The Arbortext Editor cursor changes shape to indicate the operations permitted where the cursor is placed.

### The I-Beam Cursor

The I-beam cursor indicates that entering text in the current location is permitted.


### The Invalid Cursor

The invalid cursor is a vertical bar with arrow heads at each end. The invalid cursor indicates that no text insertion is allowed in the current location. Before you add content, you must insert a valid element at the cursor location that can contain text.




## Inserting Elements and Content

Arbortext Editor provides several ways to insert elements in a document: the Quick Tags menu, the Markup toolbar, and the Insert Markup dialog box. Of these options, the Quick Tags menu is the most practical way.

To insert a tag in your document, position the cursor at the desired location and do one of the following:

- Use the Quick Tags menu:

1. Locate the editor cursor at the document position where you want to insert the element and press Enter.
2. In the Quick Tags menu, select the desired option.

> **Note:** The `split` option is used to quickly insert a similar tag after the current one. While choosing among options in the Quick Tags menu, press a letter repeatedly to highlight those options starting with it. For example, pressing the letter *I* once would highlight the `image` option, whereas pressing it twice would highlight `imagemap`.

- Use the Insert Markup icon:

1. In the Markup toolbar, click the arrow next to the **Insert Markup** icon to show the elements permitted at the current location.

2. From the drop-down list, select the desired option.

- Use the Insert Markup dialog box:

1. Open the Insert Markup dialog box by doing one of the following:
   - From the **Insert** menu, select the **Markup** option.
   - Press Ctrl+Shift+M.
   - From the context menu, select the **Insert Markup** option.
   - In the Markup toolbar, select the **Markup** button.
2. In the Insert Markup dialog box, select an element from the list.
3. Click **Insert**.

> **Note:** You can leave the dialog box open while working on your document. The list updates its elements according to the cursor position.

After an element is inserted, the cursor is automatically positioned next to its opening tag for you to add content.

## Hiding and Showing Element Contents

The expand and collapse functions let you show or hide a tag's child elements. This enables you to better understand a document and facilitates navigation.

To expand or collapse child elements under a tag, do one of the following:

- Use the View menu options.

1. Select the desired element by double-clicking its tag icon.
2. From the **View** menu, select either **Expand Element Content** or **Collapse Element Content**.

- In the document map, click the expand or collapse icon at the left side of a tag.

| | |
|---|---|
| Expand icon |  |
| Collapse icon |  |

- Use the context menu options:

  1. Right-click the target element.
  2. In the context menu, select either **Expand Element Content** or **Collapse Element Content**.

## Hiding and Showing Tag Attributes in Edit View

Arbortext Editor's Edit view displays element attributes inside the corresponding element tag in the form of *element_name="element_value"* however, as the amount of attributes grows, the view can become cluttered.

Complete the following steps to hid or show the element attributes in Edit view:

1. From the **Tools** menu, select the **Preferences** option.
2. In the Preferences dialog box, select the **View** category.
3. In the Edit View pane, select or deselect the **Attributes** option according to the case.

## Selecting an Element or its Content

In the Arbortext Editor, you can select content in two ways: you can either select an entire element within the document structure or limit the selection to its child elements and contents.

Do one of the following to select an entire element:

- Double-click a tag icon.
- Right-click a tag icon or its content and select the **Select Element** option.

> **Note:** In the Edit view, it is easier to make a selection by viewing full tags or partial tags.

Do one of the following to select an element's content:

- Select the content by dragging the mouse cursor across it.
- Right-click a tag icon or its contents and select the **Select Element Content** option.
- With the cursor within the target element:
  - Choose the **Select Element Content** under the Edit menu.
  - In the Markup toolbar, click on the **Select Element Content** button.

- Press Control+E.

## Changing an Element Markup

Arbortext Editor enables you to transform one element to another element, as long as the new element you choose is a valid element at the current location. This feature enables you to make changes quickly. For example, you can modify the type of list you use, or modify a semantic tag.

To change an element markup type, select the target element and do one of the following:

- From the Application toolbar, expand the **Current Markup** list and select among the valid elements.


- Use the Change Markup dialog box:

  1. In the Markup toolbar, click the **Change Markup** button.

  2. In the Change Markup dialog box, ensure that Tags is selected in the **Mode** list, and select among the items in the Elements pane.
  3. Click **OK**.

The target element is changed to your selection.

## Deleting an Element

You can delete an element by using either the **Delete** key, or using the **Edit** menu.

Be careful when you delete an element, so that you are sure you are deleting only the content you intend to delete. . Deleting an element with a high level in the document hierarchy eliminates its child elements.

1. Double-click the element you want to delete to select that element.
2. Do one of the following:
   - Press **Delete**.
   - In the **Edit** menu, select the **Delete** option.

The selected element is deleted from the document.

## Moving Content with the Document Map

Moving content within or across documents is done by dragging and dropping. The Arbortext Editor enables to move a complete element or a selection. Selections can include text or other element; they can be moved as long as each selection is deposited in a valid document location.

1. If the document map view is not already displayed, then select the **Document map** option from the **View** menu.

2. In the document map, select the target element, content, or a mix of the two.
3. Click the selection and drag it to the desired document position.

   The cursor changes depending on its location in the hierarchy and the resulting action of dropping the selection there.

| | |
|---|---|
|  | This is a valid location for dropping the content. |
|  | Ending document location is invalid for the attempted element movement. |
|  | Ending document location is valid for the attempted element movement, but will result in adding a new element. |

4. Release the mouse button to complete the element movement.

# About Expanding or Collapsing the Document Structure

As books grow and the number of element increases, the Edit pane can become cluttered, making navigation through the document difficult. Use the Arbortext Collapse and Expand View options to simplify the view of the file you are editing.

## Expanding or Collapsing Single Element Contents

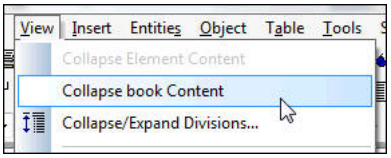
You can manage the way element contents are displayed in Edit View or Document Map by selecting an Expand or Collapse View option. The position of the cursor determines where the command you select takes effect. You can expand or collapse contents, ranging from an individual element that you select, up to *high-level container* elements in your book file.

Do one of the following to simplify the view of a book or high-level container:

- In Document Map, click the **AttributePlus** or **AttributeMinus** sign next to an element to collapse or expand its contents.

- Select an element and, from the**View** menu, select one of the **Expand or Collapse** options.

  The text displayed in the option label varies according to the case. The following graphic shows where to find the option under the View menu:



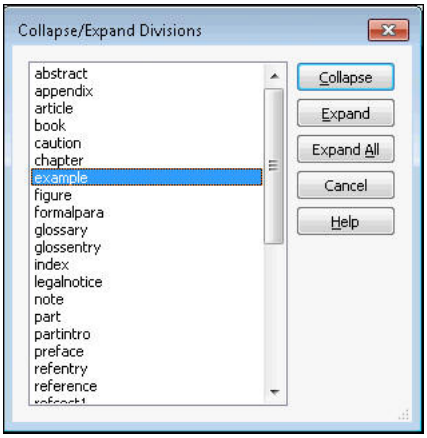
**Note:** This option enables you to easily manage how **high-level containers** are shown. They include the following kinds of elements:

- Books
- Parts or divisions
- **Component-level elements** such as chapters, appendixes, preface, glossary, and bibliography
- Sections, from level 1 to 4

## Expanding or Collapsing Divisions

The Collapse/Expand Divisions dialog box enables you to collapse or expand all occurrences of a particular kind of element in the document.

1. From the **View** menu, select the **Collapse/Expand Divisions** option.
2. In the Collapse/Expand Divisions dialog box, select the kind of elements to collapse or expand from the list at the left.




3. Click **Collapse** or **Expand** depending on the action you want to perform.

   The action is applied to the selected elements.

4. Click **Close** to dismiss the dialog box.

## Using Keyboard Shortcuts to Expand or Collapse the Document Structure

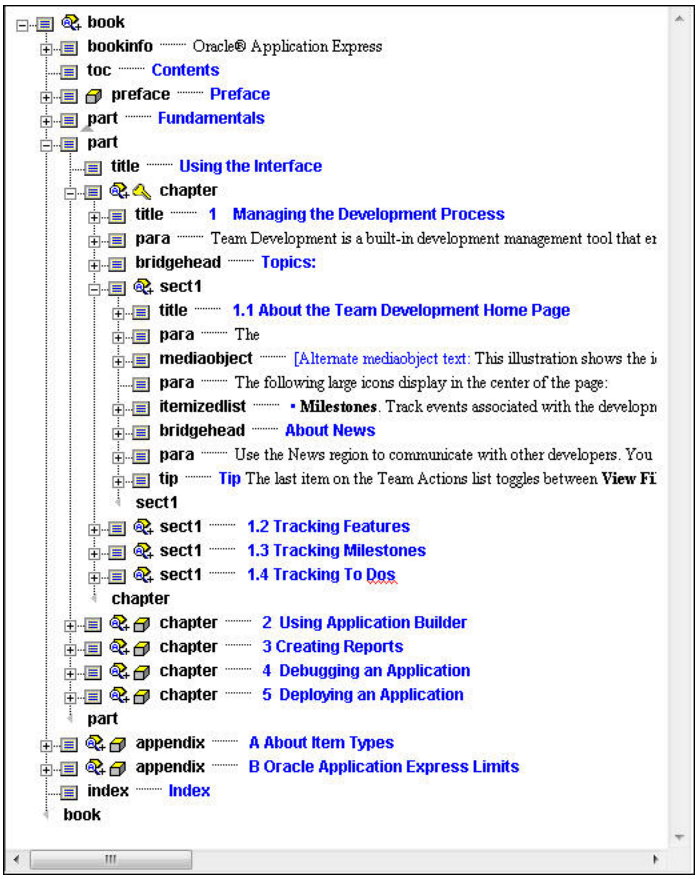
Press Control+Shift+[0|1|2|3|4|5|6|7|8|9] to expand the document hierarchy out to the level indicated by the number that you select. Zero corresponds to the highest level, 1 to both the highest element and to its first child element, and so on.

The results of using the keyboard shortcut option to collapse a level are easier to anticipate in the Document Map, but you can use it in the Edit view as well.

**Note:** Select one element in the hierarchy to restrict the collapse or expand action to its child elements.

Figure 9-1 shows an example of a book hierarchy expanded by a combination of selecting elements and using keyboard shortcuts.

***Figure 9-1 Book Hierarchy Expanded with Keyboard Shortcuts***




## Viewing Element Attributes

You can view element attributes in the Document Map, or in Edit view. The Document Map includes icons that indicate attributes whose default values have been modified; the Edit view shows attributes inside the element tag. The Modify Attributes dialog box shows all attributes, whether modified or not.

To view an element attributes, do one of the following:

- In the Document Map view, click the **AttributePlus** icon next to an element tag.

  The element attributes whose default value has been changed are displayed.

- Display the Modify Attributes dialog box by placing the cursor within an element and doing one of the following:
  - Press Ctrl+D.
  - In the Markup toolbar, click the **Modify Attributes** button.

- In the **Edit** menu, select the **Modify Attributes** option.

The dialog box shows the element being modified and its attributes in the pane below.

## Assigning Element Attributes

Use the Modify Attributes dialog box to assign or change attribute values. Consider the valid values for the attribute before changing its values.

1. Position the Arbortext Editor cursor inside the element whose attributes you want to assign or change.
2. Display the Modify Attributes dialog box by doing one of the following:
   - Press Ctrl+D.
   - In the Markup toolbar, click the **Modify Attributes** button.

   - In the **Edit** menu, select the **Modify Attributes** option.

   The Modify Attributes dialog box shows the kind of element being modified at its top and may group attributes within tabs when an element has a large number of them.
3. Enter or select a value in the field next to the desired attribute.
4. Click **OK**.

The attribute and its new value are now displayed in the element tag icon.

**Note:** If attributes are hidden, then set the editor preferences to display them.

## Assigning Element IDs

Generate an unique ID value to identify an element within and across documents. You must assign an ID to an element whenever there is a crossreference to that element (for example, a formal table, figure, or formal example).

1. Select the element where you want to assign an ID.
2. Automatically assign an ID by doing one of the following:
   - In the **Markup** toolbar, click the **Generate ID** button.
   - From the **Insert** menu, select the **Generate ID** option.
   - Use the Modify attributes dialog box:
     1. Press Ctrl+D to open the Modify Attributes dialog box.

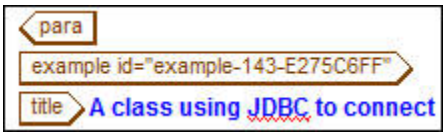
2. Under the **Reference** tab, click the magic wand icon next to the id field.

   Arbortext generates an ID.

3. Click **OK**.

The new ID value is displayed with the element tag as shown in Figure 9-2 .

***Figure 9-2 An example Tag Showing its ID Attribute Value***




## Inserting Symbols and Special Characters

Insert symbols unavailable in your keyboard as well as *character entities* by using the Insert Symbol dialog box provided by Arbortext Editor. Undesired output is occasionally generated from a document when characters are misinterpreted, such as those used in XML markup or to delimit text strings. Use *character entities* to avoid misinterpretation.

In the XML context, the list of characters included in Table 9-1 could be misinterpreted if you entered them directly from the keyboard.

***Table 9-1 Character Entities Available in the Arbortext Insert Symbol Dialog Box***

| Character | Name | Description |
|---|---|---|
| “ | quot | Double quotation mark |
| & | amp | Ampersand |
| ‘ | apos | apostrophe |
| < | lt | Less-than sign |
| > | gt | Greater-than sign |
| | nbsp | Nonbreaking space |

1. Position the Arbortext Editor cursor at the location where you want to insert a symbol.
2. Open the Insert Symbol dialog box by doing one of the following:
   - Press Control+Shift+R.
   - In the **Markup** toolbar, click the **Insert Symbol** button.

   - From the **Insert** menu, select the **Symbol** option.
3. In the Insert Symbol dialog box, choose the **Symbols** or **Character Entities** tab depending on which kind of symbol you want to insert.
4. Scroll through the symbol pane if necessary and select the desired symbol.

5. Click the **Insert** button.

   The symbol is inserted in your document.

6. Click **Close**.

> **Note:** Quickly add a hyphen, en dash, or em dash by pressing the Hyphen key repeatedly to cycle between these options.

# 10

# Advanced Topics

Arbortext Command Language (ACL) is a powerful scripting tool included in Arbortext Editor. Advanced users benefit from ACL because it allows them to customize the editor interface and gain control over the authoring process.

## About Editing XML Markup

Editing content in its plain XML source instead of using the Arbortext Editor tag representations can help advanced users to better understand a document and have more control over it. Arbortext Editor enables users to directly view or edit a file's source code for these purposes.

Arbortext Editor provides the following features to edit content in its raw XML form:

- Editing the whole document: Using this option will replace the current view of the Edit pane (Edit view or Document Map) with a text editor showing XML markup and text.
- Editing a piece of content in a document: A new dialog box is used to make simple changes to a selection in a document. The selection is in a text editor but isolated from the rest of the document, making it easier to understand.

## Showing the Command Line

Use the command line to enter Arbortext Command Language (ACL) commands. ACL commands are used to customize the editor window, write simple scripts, and change the behavior of keyboard keys.

Complete the following steps to show or hide the command line in Arbortext Editor:

1. Under the **Tools** menu, select the **Preferences** option.
2. In the Preferences dialog box , select the **Window** category from the left pane.
3. In the Show group, select the **Command Line** option.
4. Click **OK**.

The Command Line bar shows in the Arbortext Editor window, below the Edit pane.

> **Note:** If anything other than *Command* is shown (for example *JavaScript* or *VBScript*) then press F5 while the cursor is in the command line until *Command* is shown.

# Editing a Whole Document in XML

The ACL `edit` command shows a document's XML source code. The `edit` command converts the Edit pane into a text editor that displays the current file's contents without its tag representations.

> **Note:** Save your document and any recent modifications before you attempt to view or edit the document in XML mode.

1. Position the cursor in the command line.

   > **Note:** If the command line does not show the *Command* label, then press F5 (while the cursor is in the command line) to iterate through the options.

2. Enter the following text in the Command Line:

   ```
   edit -current -untagged
   ```

3. Press Enter.

   The Edit pane shows the document XML code. The pane behaves similar to a text editor.

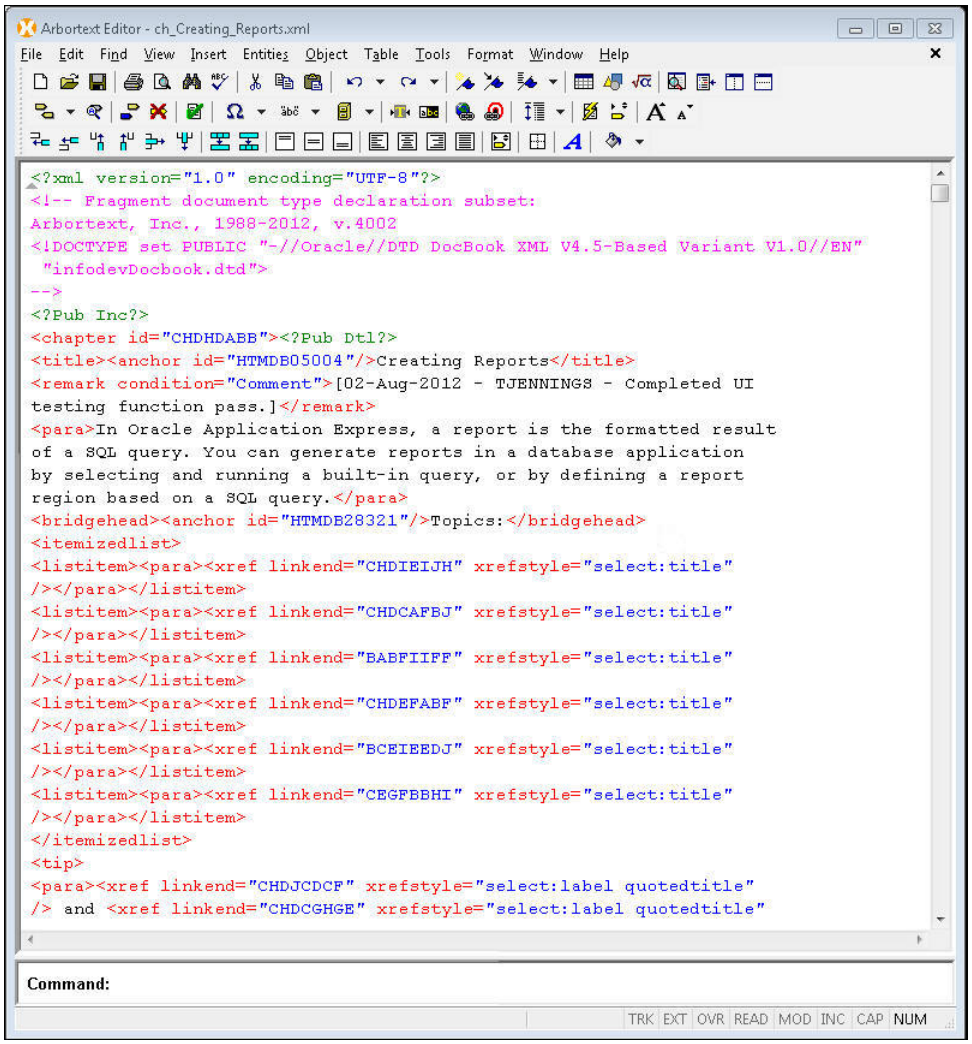
4. Save the document when the necessary changes have been made.

5. Optional: Enter the following text to return the Edit pane to its previous form:

   ```
   edit -current -xml
   ```

Figure 10-1 shows Arbortext Editor displaying a document's XML source code after using the `edit` command.

***Figure 10-1 The Arbortext Editor Edit Pane Showing a Document's XML Source Code***




# Editing a Portion of a Document in XML

Use the Edit Selection as XML Source option to display part of a document's content in its raw XML form. This option is useful when you need to change the source code from one or more elements but the document has become to complicated to view it as a whole.

> **Note:** Configure Arbortext Editor so that it shows full menus before you proceed with the current task.

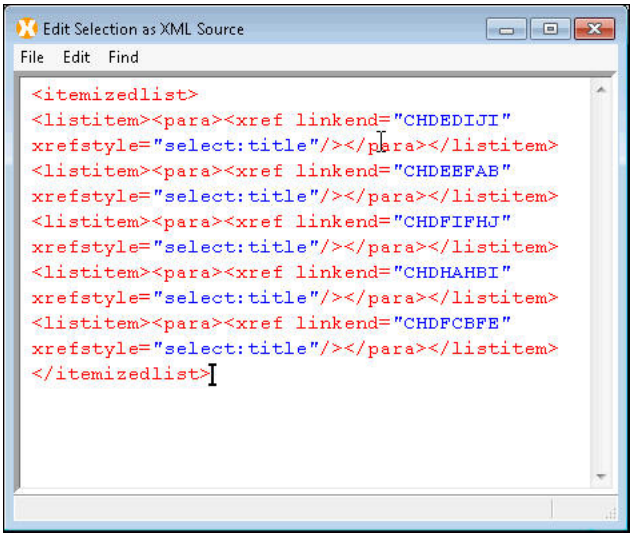
1. In the Edit view, select the desired content to be modified.
2. From the **Edit** menu, select **Edit Selection as XML Source**.

   A new dialog box appears showing the XML markup corresponding to the selected content.
3. Modify the content as needed.
4. From the **File** menu, select **Update Selection**.

The changes are reflected in the main document.

Figure 10-2 shows the Edit Selection as XML Source dialog box, which displays the source code of an `itemizedlist` element and its child elements.

***Figure 10-2 The Arbortext Editor Edit Selection as XML Source Dialog Box***




**See Also:**

Displaying Full Menus in Arbortext
: Postinstallation tasks include displaying full menus and menu options in Arbortext Editor. The new options enable you to take full advantage of the product's capabilities.

# Part III

## Adding Content

# 11

# Formatting Text

Format inline text in DITA either by wrapping text with semantic tags (preferred), or by wrapping text with typographic tags, such as `b` (for bold).

Semantic tags add both meaning and implied formatting to content. Stylesheets determine the formatting of content enclosed in semantic tags. Typographic tags apply styles to content.

## About Text Formatting in DITA

To format inline text, you enclose the text in one of three kinds of tags: semantic tags, typographic tags, and the `codeph` tag. Inline text is a word or phrase within a sentence. Examples include function names, database view names, or UI elements such as menu names or button names.

### Semantic Tags

Use semantic tags when possible for inline content. Semantic tags assign meaning to the marked up content. They make it easier for other writers reading your content to understand its meaning. The XSL stylesheets determine how content enclosed in semantic tags is formatted in the output. This scheme is preferable to formatting inline content with typographic tags like <b> (bold) or <i> (italic).

Semantic tags provide the following benefits:

- You do not have to remember how to format content; you just have to know the meaning of the content.
- Content formatting is standardized in stylesheets across the library.
- Output can be customized for semantic tags using different stylesheets for different display devices and output types.
- XML processing is made possible. For example, you could write an XSL stylesheet to extract all function names from a publication. You could not do this if function names were marked up with typographic tags.

The following semantic tag marks up a file path:

```
<filepath>...</filepath>
```

Consider the sentence: “The script is located in the `/usr/local` directory.”

In the XML, the path is enclosed in the semantic tags for a file path:

```
<filepath>/usr/local</filepath>
```

### Typographic Tags

Use typographic tags to add styles to text. Use typographic tags only when no semantic tag is applicable.

The following typographic tag results in italic text:

```
<i>...</i>
```

Consider the sentence: "Do *not* start the database at this point."

In the XML, the word "not" is enclosed in the tags for italic font:

```
<i>not</i>
```

### The codeph Tag

The `codeph` tag is a special tag that does not fit in the semantic category or the typographic category. This tag is used as a fallback semantic tag for any code or object name that should be rendered in monospace font but for which no other semantic tag exists.

For example, there is no semantic tag for database views. When you include a view name in a sentence, you use the `codeph` tag both for formatting and to denote a reference to an object. In the following sentence, the `USER_TAB_COLS` view is enclosed in `codeph` tags:

```
Query the <codeph>USER_TAB_COLS</codeph> view to get the columns of
a table.
```

**See Also:**

About DITA
: Darwin Information Typing Architecture (DITA) is an XML-based architecture designed for writing, organizing, and linking topic-based

content. DITA maps organize topics according to their content model to create different kinds of documents.

Formatting Content with Semantic Tags
: You use semantic tags to mark up inline text (phrases, names, file paths, and so on) and impart both meaning and implied formatting to the marked up text. Semantic tags are preferred over typographic tags.

Adding Style to Text with Typographic Tags
: You use typographic tags to add style to text or enclose text in quotes.

Applying Monospace Font to Text with the codeph Tag
: You use the `codeph` tag to apply monospace font to inline text (database table names, view names, and so on) when there is no appropriate semantic tag available.

Semantic Tags Reference
: Use these semantic tags to add meaning and formatting to inline content.

Typographic Tags Reference
: Use these typographic tags to add formatting to your content. Semantic tags are preferred over typographic tags.

# Formatting Content with Semantic Tags

You use semantic tags to mark up inline text (phrases, names, file paths, and so on) and impart both meaning and implied formatting to the marked up text. Semantic tags are preferred over typographic tags.

Mark content with semantic tags to format text correctly. Only format content using typographic tags when there is no semantic tag available.

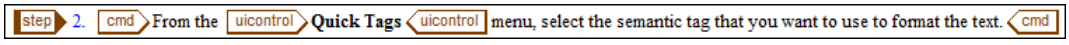
1. In Edit view in Arbortext Editor, select the word or group of words that you want to format, and press Enter on your keyboard.
2. From the **Quick Tags** menu, select the semantic tag that you want to use to format the text.

   The selected text is wrapped with the semantic tag that you selected.

> **Note:** Depending on the element used, the Edit view may or may not show the tagged content in its final output format. However, format is always applied to content within semantic tags when the document output is generated.

Figure 11-1 shows a `uicontrol` tag in a step as seen in Edit view. Use the `uicontrol` semantic tag to format a user-interface control inline.

***Figure 11-1 Using a Semantic Tag to Mark a User-Interface Control Inline***




**See Also:**

About Text Formatting in DITA
: To format inline text, you enclose the text in one of three kinds of tags: semantic tags, typographic tags, and the `codeph` tag. Inline text is a word or phrase within a sentence. Examples include function names,

database view names, or UI elements such as menu names or button names.

Semantic Tags Reference
: Use these semantic tags to add meaning and formatting to inline content.

## Adding Style to Text with Typographic Tags

You use typographic tags to add style to text or enclose text in quotes.

Format content using typographic tags only when there is no semantic tag available. Examples of typographic tags include bold (b), italics (i), and quotes (q).

> **Note:** The advantage of using `<q>` over typing quotes is that the publishing engine inserts locale-specific quotation marks.

1. In Edit view in Arbortext Editor, select the word or group of words that you want to format, and press Enter.
2. From the **Quick Tags** menu, select the typographic tag that you want to use to format the text.

   The selected text is wrapped with the typographic tag that you selected.

**See Also:**

About Text Formatting in DITA
: To format inline text, you enclose the text in one of three kinds of tags: semantic tags, typographic tags, and the `codeph` tag. Inline text is a word or phrase within a sentence. Examples include function names, database view names, or UI elements such as menu names or button names.

Typographic Tags Reference
: Use these typographic tags to add formatting to your content. Semantic tags are preferred over typographic tags.

## Applying Monospace Font to Text with the codeph Tag

You use the `codeph` tag to apply monospace font to inline text (database table names, view names, and so on) when there is no appropriate semantic tag available.

1. In Edit view in Arbortext Editor, select the word or group of words to which you want to apply monospace font, and press Enter on your keyboard.
2. From the **Quick Tags** menu, select `codeph`.

   The selected text is wrapped with the `codeph` tag.

**See Also:**

About Text Formatting in DITA
: To format inline text, you enclose the text in one of three kinds of tags: semantic tags, typographic tags, and the `codeph` tag. Inline text is a word or phrase within a sentence. Examples include function names,

database view names, or UI elements such as menu names or button names.

# Semantic Tags Reference

Use these semantic tags to add meaning and formatting to inline content.

### Semantic Tags

Only use the semantic tags in the following table to reference an object name within a sentence, not to mark up definitions of objects in a reference topic. For example, if you are writing about a function in a reference topic and have a section or table that defines each function parameter, then do not use `parmname` to mark up the parameter name. Instead, use `codeph`. For example, use `cmdname` only in a situation like this: "The `STARTUP` SQL*Plus command starts an Oracle Database." Here, the `cmdname` tag is applied to the word "STARTUP" in the sentence.

***Table 11-1 Semantic Tags***

| DITA Tag | Used to Indicate | Output Format |
|---|---|---|
| `apiname` | The name of an application programming interface (API), such as a function or procedure name, excluding PL/SQL and Java database program units, which are schema objects | `Monospace` |
| `cmdname` | The name of a command, including utility names, script names, and SQL statements | `Monospace` |
| `filepath` | A directory path or file path | `Monospace` |
| `menucascade` | A series of hierarchical menu choices, rendered in the output as **Menu** > **Menu** > **Menu** | **Bold** |
| `msgnum` | A system message or error message number | No formatting |
| `msgph` | A system message or error message | `Monospace` |
| `option` | An option that can be used to modify a command, SQL statement, or subprogram, such as `-lrt` in the command `ls -lrt` | `Monospace` |
| `parmname` | A parameter name, when describing parameters of an API, command, or script, including parameters of PL/SQL and Java database program units | `Monospace` |

| DITA Tag | Used to Indicate | Output Format |
|---|---|---|
| `systemoutput` | Output generated by software or responses to a command | `Monospace` |
| `term` | A new word or phrase that requires a definition | **Bold** |
| `uicontrol` | A graphical user interface control, such as a button, menu item, text field, and so on | **Bold** |
| `userinput` | Text that a user should input | `Monospace` |
| `varname` | The name of a variable that must be supplied to a software application, or a placeholder for a user-supplied value | *Italic* |
| `wintitle` | The title of a window or dialog box in a graphical user interface | No formatting |

**See Also:**

About Text Formatting in DITA
: To format inline text, you enclose the text in one of three kinds of tags: semantic tags, typographic tags, and the `codeph` tag. Inline text is a word or phrase within a sentence. Examples include function names, database view names, or UI elements such as menu names or button names.

Formatting Content with Semantic Tags
: You use semantic tags to mark up inline text (phrases, names, file paths, and so on) and impart both meaning and implied formatting to the marked up text. Semantic tags are preferred over typographic tags.

# Typographic Tags Reference

Use these typographic tags to add formatting to your content. Semantic tags are preferred over typographic tags.

***Table 11-2 Typographic Tags***

| Tag | Markup | Output Format |
|---|---|---|
| `b` | `<b>...</b>` | **Bold** |
| `i` | `<i>...</i>` | *Italic* |
| `q` | `<q>...</q>` | “Quotation marks” |
| `u` | `<u>...</u>` | Underline |

**See Also:**

About Text Formatting in DITA
: To format inline text, you enclose the text in one of three kinds of tags: semantic tags, typographic tags, and the `codeph` tag. Inline text is a word or phrase within a sentence. Examples include function names, database view names, or UI elements such as menu names or button names.

Adding Style to Text with Typographic Tags
: You use typographic tags to add style to text or enclose text in quotes.

# 12

# Inserting Tables

DITA supports both formal and informal tables, and includes elements that you can use to make tables accessible.

## About Tables in DITA

Tables are either formal or informal. They must comply with accessibility requirements by adding descriptive information about the table, and its organization, in the `desc` element which is available to assistive technology.

### Formal Tables

Formal tables are tables that are numbered, that include a title, and that have generated values for their `id` attributes. Use a formal table:

- When users require titles to be able to scan for table content information, such as in a series of reference tables
- When the table is referred to from other parts of the document
- When the table should be included in the table of contents or the list of tables
- When the table exceeds the length of a page in a printed document

In DITA, insert a formal table by adding a `table` element to your document. Formal tables must contain a `title` element, followed by the `desc`, and the body of the table at the end.

### Informal Tables

Informal tables do not have titles. They are commonly embedded in a procedure or within a reference element. Users understand the type of content contained in the table by the topic in which the table is embedded.

Informal tables are not included in the list of tables and cannot be referenced from other parts of the document. When you author in DITA, use the `table` element to insert an informal table.

**See Also:**

Accessibility in DITA
: Tables and figures, either formal or informal, must comply with accessibility requirements. Always use the correct DITA elements to

make descriptive information available to all readers, including those with disabilities.

## Inserting a Formal Table

A formal table is numbered and has a title. You can refer to a formal table from other parts of your document. Formal tables are included in the list of tables that appears after the table of contents.

1. In Edit view, position the cursor where you want to include a formal table, and press Enter.
2. In the Quick Tags menu, select the **table** option.
3. In the Insert Table dialog box, set the table dimensions by entering values in the **Rows** and **Columns** fields.
4. Click **OK**.

   A `table` element is inserted.
5. Generate an ID for the `table` element:
   a. Select the table by double-clicking its tag.
   b. In the Markup toolbar, click the **Generate ID** button  .

      Arbortext Editor generates an ID.

      **Note:** SDL LiveContent Architect automatically assigns a GUID to every table, figure, and example when the containing topic is checked in. However, if you want to create a cross-reference to any of these elements before checking the topic in then you must manually assign an ID. When you manually assign an ID, this ID is not overwritten with a GUID upon check-in.
6. Create a table title:
   a. Position the cursor just inside the opening `table` tag and press Enter.
   b. Using the Quick Tags menu, insert a **title** element.
   c. Enter the table title between the `title` tags.
7. Convert the first row to a header row.
   a. Position the cursor in any cell of the first row.
   b. From the **Table** menu, select the **Convert to Header Row** option.

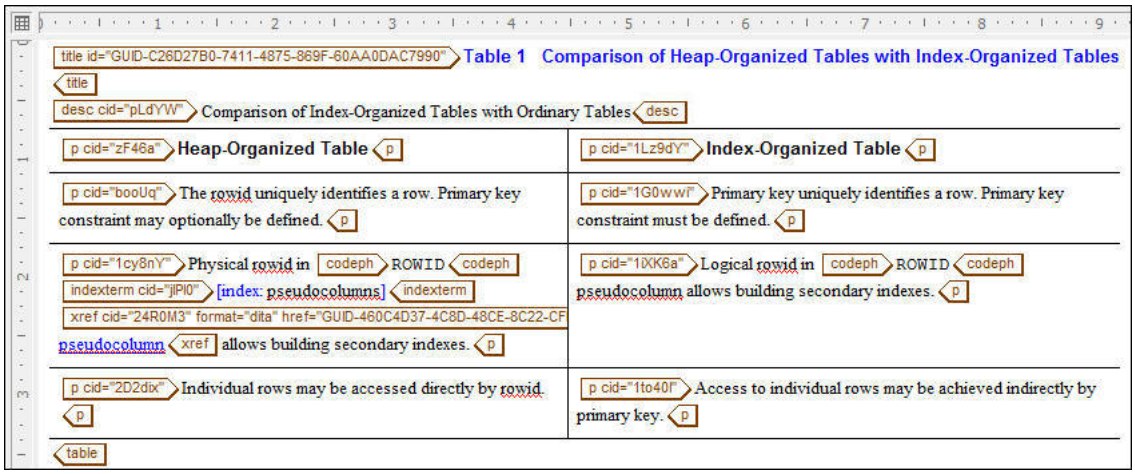
      The header row icon  appears in the table ruler indicating a new row format.
8. Enter content in each of the table cells.
9. Include a `desc` element to comply with accessibility requirements:
   a. Position the cursor after the `title` element and press Enter.
   b. Using the Quick Tags menu, insert a **desc** element.

Include the number of rows and columns, and the heading of each column in the **desc** element.

c. Enter a table description in the new `desc` element.

After you perform the previous steps, your table should look similar to the one in Figure 12-1 .

***Figure 12-1 A Formal Table in DITA***




**Note:** The title within the table shown in Figure 12-1 shows the "Table 1" text. This text is automatically generated by Arbortext editor and corresponds only to the current document in its buffer, not the topic in the context of a publication. Tables are correctly numbered when the publication output is built in Publication Manager.

**See Also:**

Formal Tables Accessibility Checklist
: Formal tables comply with accessibility requirements by using the desc element. It includes a short description of the contents and organization of the table.

Creating Links Within a Topic
: You create links within a topic using the `xref` tag.

Creating Links Within a Topic
: You create links within a topic using the `xref` tag.

# Inserting an Informal Table

Use an informal table when you want to integrate a table to its surrounding content, such as a list item or paragraph. An informal table does not have a title or table number. You cannot create a cross-reference to an informal table from other parts of your document. Informal tables are not included in the list of tables. Use the `table` element to insert an informal table but do not insert a `title` inside.

1. In Edit view, place the cursor where you want to include an informal table, and press Enter.
2. In the Quick Tags menu, select **table**.

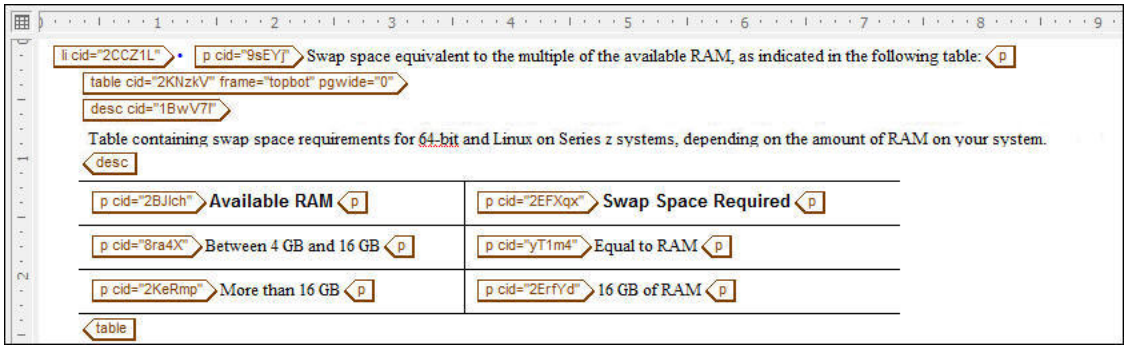
3. In the Insert Table dialog box, set the table dimensions by entering values in the **Rows** and **Columns** fields.
4. Click **OK**.

   An `table` element is inserted.
5. Optional: Convert the first row to a *Header Row*.
   a. Position the cursor in any cell of the first row.
   b. From the **Table** menu, select the **Convert to Header Row** option.

      The header row icon appears in the table ruler indicating a new row format.
6. Enter the desired content in each of the table cells.
7. Include a `desc` element to comply with accessibility requirements:
   a. Position the cursor after the `table` element and press Enter.
   b. Using the Quick Tags menu, insert a **desc** element.
   c. Enter a description in the new `desc` element.

After you follow the previous procedure, the inserted table should resemble the one in Figure 12-2 .

***Figure 12-2 Example of an Informal Table in DITA***




**See Also:**

Informal Tables Accessibility Checklist
: Informal tables comply with accessibility requirements by using the desc element.

# 13

# Inserting Figures and Graphics

In DITA, images are inserted either as figures or graphics. Use figures for reference topics and concept topics. Use graphics when you need to insert images to refer to user interface features in a task or in a reference table. For both figures and graphics, you must always include a graphic description to make the image accessible.

**See Also:**

DITA Standards for Figures and Images
: This section describes the standards for the `<fig>` and `<image>` elements. These standards are specific to the InfoDev implementation of DITA. They are intended for all authors in InfoDev.

Importing Graphics Using the Browse Repository Dialog Box
: You import a graphic by dragging it from storage on your local computer to a folder in the **Browse Repository** dialog box. For a graphic with a separate EPS version for PDF output, use the Web Client to import the EPS file.

## About Figure and Graphic Elements

Use figure and graphic elements to place images in your document. Figure elements get a title and figure number, and should be used for most images. Graphic elements are informal images, which can be embedded within content units in document, such as in a procedure.

### Figure Elements in DITA

Figure elements are formal images. They are numbered, and have titles. They are included in the list of figures following the table of contents, and can be referenced from other locations in the document.

In DITA, the element structure of a figure consists of a `fig` tag, a `title`, and an `image`. The `figure`'s `expanse` attribute can be configured to display the figure from left to right margin; the `image` tag is parent for an `longdescref` element which is used to contain a graphic description for accessibility purposes.

---

**Note:** Because figures are referenced, each `fig` tag must be assigned an element ID to direct the cross-reference. This element ID is automatically set to all figures within a topic when it is checked into Publication Manager. However, if you want to create a reference to the figure before checking the topic in then you must manually assign a value to the `ID` attribute.

---

Example 13-1 shows a typical DITA element structure for a figure.

***Example 13-1 DITA Element Structure for a Figure***

```
<fig id="1234" expanse="page">
    <title></title>
    <image href="GUID">
    </image>
</fig>
```

### Graphic Elements in DITA

Graphic elements are an informal way of inserting images. They consist of a `image` tag and its child elements. Because graphic elements are not numbered and do not have titles, they cannot be referenced from other topics, or included in the list of figures. Inline graphics are produced by setting the `inline` value to the `image`'s `placement` attribute.

If you want a graphic to fill the entire space from left to right margins then you need to place it inside a `figure` element, set its `expanse` attribute as described in the previous section, and delete the `title` tags from the element structure. By doing so, the image will take the entire space from the page but the graphic will not be included in the list of figures. Furthermore, because the `figure` element automatically inserts a break, the graphic will be placed in a new line.

**See Also:**

Accessibility in DITA
: Tables and figures, either formal or informal, must comply with accessibility requirements. Always use the correct DITA elements to make descriptive information available to all readers, including those with disabilities.

## About Image Source Files

Every graphic or figure element refers to a particular source file. Select a source file format suitable for the output that you create.

To show diagrams or flow charts, vector images are preferred for printed and PDF output, and JPG, GIF, or PNG formats are preferred for the XHTML version. In this case, provide both versions of the image. However, to display a screenshot, always use images in JPG, GIF, or PNG format.

---

**Note:** Encapsulated PostScript (EPS) is the only supported format for vector images..

---

### Images Resolutions in Publication Manager

Publication Manager supports two different graphic resolutions, **Default** and **Print**, for a single graphical object. Each resolution of a graphic is stored at a logical level below the object. This means that, although both resolutions get the same GUID, the system is able to distinguish between them and resolve which one to use when generating a particular kind of output.

---

**Note:** Previewing a topic in Publication Manager displays images only in Default resolution. If you want to see the Print version of an image object then view it in the Web Client.

---

Default images have low resolution and are used for HTML output while Print images are used for higher-quality, PDF output. However, if the Print resolution of an image does not exists, Publication Manager uses the Default resolution instead to keep the publication integrity. The following table shows the mapping between Publication Manager's graphic resolutions and the extension of images source files:

| Graphic Resolution | Source Image File Extensions |
|---|---|
| Default | • jpg<br>• jpeg<br>• png<br>• gif<br>• tif<br>• tiff |
| Print | • eps<br>• pdf<br>• cgm<br>• svg<br>• svgz<br>• wmf |

**See Also:**

Importing Graphics Using the Browse Repository Dialog Box
: You import a graphic by dragging it from storage on your local computer to a folder in the **Browse Repository** dialog box. For a graphic with a separate EPS version for PDF output, use the Web Client to import the EPS file.

## Images and Document Accessibility

Adding a graphic description to graphics and figures is an important part of making documentation accessible. Visually impaired audiences use screen reader software to access text descriptions of images, and the content in these descriptions must be equivalent to the information in the image.

In DITA, use the `alt` element to add a brief description of navigation images. For example, buttons and arrows. For longer, more complex descriptions (such as those required for diagrams) you create a separate topic, of the **Graphic Description** type, containing the text description and reference this topic using a `longdescref` element.

> **Note:** The `londescref` element is an empty element. You use its `href` attribute to reference the topic containing the graphic description.

Guidelines for creating adequate graphic descriptions:

- Use statements such as "at upper left is a..." or "the radio buttons on the left..." to help readers understand the layout of the picture.
- Do not use the word `graphic` because it is a keyword for most screen readers. Use `image`, `illustration`, or `figure` instead.
- Avoid including the image file name in the description. If you change the image file name, then you must update the description.

Example 13-2 shows a graphic followed by an alternate text description that describes the content of the image.

***Example 13-2 A Graphic Description for Accessibility Purposes***


This illustration shows a search bar with View Details enabled. From left to right, the bar consists of: a search field; a drop down list to select the kind of report; a group of buttons used to select the icon, list, and detail view; and finally the Actions menu. When configured, a View Details icon displays on the Search bar.

## Inserting a Figure

A figure contains an image, a title, and a figure number, which enables you to refer to that figure from other parts of a document. Figures are also listed by figure number and title in the table of figures, which is located after the table of contents.

1. In Edit view, place the cursor where you want to include the figure, and press Enter.
2. In the Quick Tags menu, select the **fig** option.
3. In the Insert Graphic dialog box, do one of the following to locate and select the graphic object that you want to insert:
   - Under the Repository tab, expand the hierarchy as needed and select a folder, then select the desired graphic object from the right pane.
   - Under the Search tab, search for the image by its metadata values and select it from the right pane.

   **Note:** Publication Manager displays only the Default (low) resolution version of graphic objects. However, if there is a Print (high) resolution version of the image in the repository, Architect automatically uses it when generating PDF output.
4. Click the **Insert** button.

   A `fig` element is inserted containing the corresponding `image` element within.
5. Optional: Generate an ID for the `fig` element:
   a. Select the `fig` element by double-clicking one of its tags.
   b. In the Markup toolbar, click the **Generate ID** button.

   Arbortext Editor generates an ID.

**Note:** SDL LiveContent Architect automatically assigns a GUID to every table, figure, and example when the containing topic is checked in. However, if you want to create a cross-reference to any of these elements before checking the topic in then you must manually assign an ID. When you manually assign an ID, this ID is not overwritten with a GUID upon check-in.

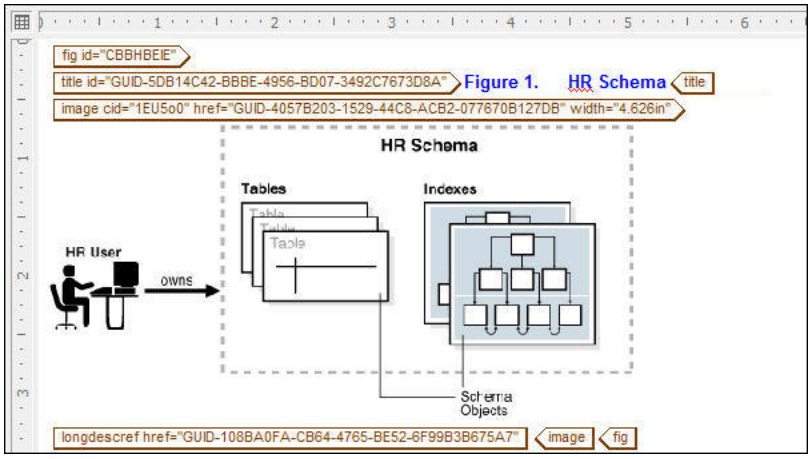
6. Optional: Configure the figure's width to fit all the space between the left and right margins.
   a. Select the `figure` element by double clicking one of its tags.
   b. Press Ctrl+D.
   c. In the Modify Attributes dialog box and under the Other tab, select the **Page** option for the Expanse field.
   d. Click **OK**.
7. Enter the name of the figure between the `title` tags.

**Note:** Do not include the Figure word or figure number as a part of the title. This data is automatically added when the document output is generated.

8. Add an alternate text description for the figure:
   a. With the editor cursor within the `image` tags, press Enter.
   b. In the Quick Tags menu, select the **longdescref** option.
   c. In the Insert Link dialog box, select the Graphic Description topic that contains the alternate text description from the repository.
   d. Click **Insert**.

   The `longdescref` element references the graphic description object through its `href` attribute.

***Figure 13-1 A Figure in DITA***



**See Also:**

Metadata for Graphics
: Metadata for maps aids in searching for graphics and tracking a graphic's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

Creating a Graphic Description Topic
: In DITA, extensive or complex graphic descriptions must be stored in separate **graphic description** topics. Use the `longdescref` element to link the graphic that is being described to the topic that contains its description.

Creating Links Within a Topic
: You create links within a topic using the `xref` tag.

## Inserting a Graphic

A graphic is an informal image. An informal image does not have a title or a figure number, so you cannot refer to an informal figure from another location in the document. Graphics are not included in the list of figures.

If the graphic is a screenshot or diagram that requires a graphic description for accessibility, then create the graphic description topic before proceeding.

1. In Edit view, position the editor cursor where you want to include the graphic and press Enter.
2. In the Quick Tags menu, select the `image` option.
3. In the Insert Graphic (browse repository) dialog box, do one of the following to locate and select the graphic object that you want to insert:
   - Under the Repository tab, expand the hierarchy as needed to select a folder. Then select the desired graphic object from the right pane.
   - Under the Search tab, search for the image by its metadata values and select it from the right pane.

   **Note:** Publication Manager displays only the Default (low) resolution version of graphic objects. However, if there is a Print (high) resolution version of the image in the repository, Architect automatically uses it when generating PDF output.
4. Click the **Insert** button.

   Arbortext inserts the new `image` element, it displays the selected graphic object.
5. Optional: Configure the `image`'s placement:
   a. With the editor cursor within the `image` tags, press Ctrl+D.
   b. In the Modify Attributes dialog box, under the **Image** tab, select one of the following options from the placement drop-down list:
      - break: Graphic will be placed on a new line.
      - inline: Graphic will be located in the same line as the surrounding text.

c. Click **OK**.

6. If you are inserting a simple graphic such as a button or icon, then add an alternate text description by doing the following:

   a. With the editor cursor within the `image` tags, press Enter.

   b. In the Quick Tags menu, select the **alt** option.

   c. Enter a description of the illustration within the new `alt` element.

      The description should just be the icon or button name. For example, if the image is a Next button, then the contents of the `alt` element should be just "Next."

7. If you are inserting a complex graphic such as a screenshot or diagram then add an alternate text description by doing the following:

   a. With the editor cursor within the `image` tags, press Enter.

   b. In the Quick Tags menu, select the **longdescref** option.

   c. In the Insert Link (browse repository) dialog box, select the graphic description topic for the image.

   d. Click **Insert**.

      The `longdescref` element references the graphic description object through its `href` attribute.

8. Generate an ID for the `image` element:

   a. Select the image by double-clicking its tag.

   b. In the Markup toolbar, click the **Generate ID** button. 

      Arbortext Editor generates an ID.

> **Note:** SDL LiveContent Architect automatically assigns a GUID to every table, figure, and example when the containing topic is checked in. However, if you want to create a cross-reference to any of these elements before checking the topic in then you must manually assign an ID. When you manually assign an ID, this ID is not overwritten with a GUID upon check-in.

After following the previous step, the markup for your new graphic should resemble the markup shown in the following figure.

***Figure 13-2 A Graphic in DITA***

**Note:** If you want to configure the graphic's width to fit all the space between the left and right margins, then follow the procedure for inserting a formal figure but delete the `title` element from the element structure. By correctly configuring the figure's `expanse` attribute, but not adding a title, the output for the graphic displays as desired and the graphic itself is not included in the list of figures.

**See Also:**

Metadata for Graphics
: Metadata for maps aids in searching for graphics and tracking a graphic's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

Graphic Accessibility Checklist
: In DITA, graphics comply with accessibility requirements by using the `alt`or `longdesrev` tag within their element structure. The element must provide a description in a way that users can understand the layout of an image, the flow of a diagram, or the action that is illustrated, without actually viewing the image.

Creating Links Within a Topic
: You create links within a topic using the `xref` tag.

## Creating a Graphic Description Topic

In DITA, extensive or complex graphic descriptions must be stored in separate **graphic description** topics. Use the `longdescref` element to link the graphic that is being described to the topic that contains its description.

1. In the Browse Repository window, under the **Repository** tab, select the folder where you want to create the new topic.

   **Note:** The selected folder must be of the Module content type.

2. Click the **New Object** button.

3. In the Select Template dialog box, under the tab for your group, select the **Graphic Description** template. For example, if you are a database writer, then select the template under the **Database Topics** tab.
4. In the Add Object dialog box, fill in the metadata fields.
5. Click **OK**.

   The new topic object is now listed in the right pane of the window.
6. In the Browser Repository window, check out the newly created topic.

   Arbortext Editor loads the document.
7. In Arbortext Editor:

a. Delete the contents of the `title` element.

> **Note:** Leave the empty `title` tags in the document. This is a required element in the topic type.

b. Enter the graphic description within the `body` of the topic.

> **Note:** You can use several element such as paragraphs, lists, or tables to enhance the graphic description.

8. Check the topic into the repository.

The graphic description object is now ready to be referenced from another topic.

**See Also:**

Metadata for Topics
: Metadata for maps aids in searching for topics and tracking a topic's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

Checking In Objects
: When you finish editing an object, you must check it back into the Repository to upload the latest data and to unlock the object, which makes it available for checkout by other users.

Inserting a Figure
: A figure contains an image, a title, and a figure number, which enables you to refer to that figure from other parts of a document. Figures are also listed by figure number and title in the table of figures, which is located after the table of contents.

Inserting a Graphic
: A graphic is an informal image. An informal image does not have a title or a figure number, so you cannot refer to an informal figure from another location in the document. Graphics are not included in the list of figures.

# 14

# Inserting Examples

Examples help readers to comprehend theoretical information by showing a real instance of what is being described. The elements contained in some examples allow them to be referenced from anywhere in the document, making them easy to find by readers.

## About Examples

Use examples to emphasize the information that you are presenting and to make it easy for readers to find. Examples can be formal or informal, they differ in the elements they contain and their ability to be referenced.

### Formal Examples

Formal examples are numbered and given titles. Formal examples are included in a list of examples, which is placed after the table of contents. Each formal example has a unique identifier, so you can refer to it from different parts of the document.

The structure of a formal example consists of an `example` tag that includes a value for its `id` attribute (it enables the example to be cross-referenced), immediately followed by a `title` tag. After the title, you can add an introductory paragraph, and then present the actual code that composes the example within a `codeblock` element. For example:

```
<example id="1234">
    <title>...</title>
        <p></p>
        <codeblock>
     ...
     </codeblock>
    <example>
```

### Informal Examples

Informal examples are not given titles or numbers, they are not included in lists of examples, and you cannot cross-reference them. Include informal examples in places where the example is closely linked to a specific action, such as a part of a list item.

The typical informal example consists of a pair of `example` tags wrapping the `codeblock` that contains the actual example.

# Inserting Formal Examples

Formal examples typically include monospace text. They have example numbers and titles, allowing them to be cross-referenced, and they appear in the list of examples following the table of contents.

1. In Edit view, place the cursor where you want to include your example, and press Enter.
2. From the Quick Tags menu, select the **example** option.

   An `example` element is inserted.
3. Create an ID for the `example` element.
   a. In Document Map, select the `example` element.
   b. Press Ctrl+D. to open the Modify Attributes dialog box.
   c. Under the Reference tab, click the **Generate ID** icon next to the id field.

   Arbortext Editor generates an ID.

> **Note:** SDL LiveContent Architect automatically assigns a GUID to every table, figure, and example when the containing topic is checked in. However, if you want to create a cross-reference to any of these elements before checking the topic in then you must manually assign an ID. When you manually assign an ID, this ID is not overwritten with a GUID upon check-in.

4. Using the Quick Tags menu, insert a `title` element immediately after the `example`.
5. Enter the name of the example between the title tags.

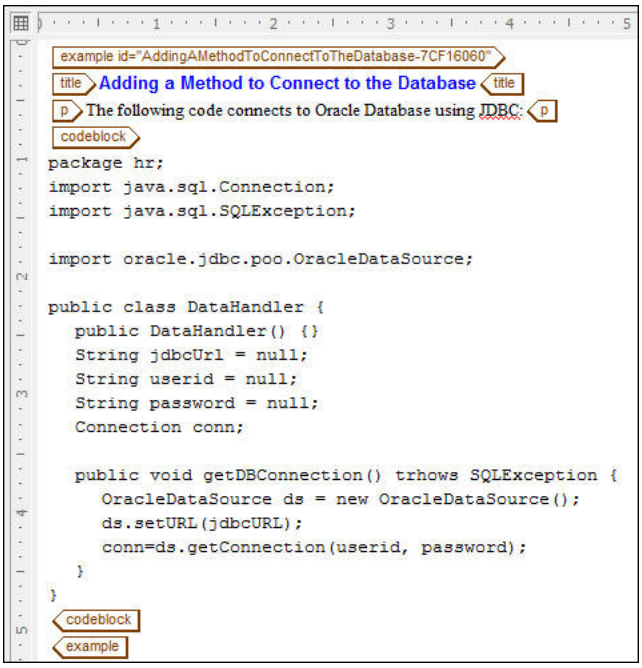
> **Note:** Do not include the Example word or the example number as a part of the title. This information is added automatically when document output is generated.

6. Use the Quick Tags menu to insert a **p** and **codeblock** elements.
7. Enter introductory text and the desired code within the `p` and `codeblock` tags, respectively.

> **Note:** White space inside the `programlisting` element is preserved. The listing can span multiple lines.

After you perform the previous steps, your example should look similar to Figure 14-1 .

*Figure 14-1 A Formal Example in DITA*




**See Also:**

Creating Links Within a Topic
: You create links within a topic using the `xref` tag.

## Inserting Informal Examples

Informal examples are monospace text that do not have example numbers, titles, or introductory text. As a result, they are not included in lists of examples and cannot be cross-referenced. They are, however, more versatile than formal examples because they can be inserted in more places in a document, such as inside a list item.

In DITA, use the `codeblock` element to contain the code in an informal example.

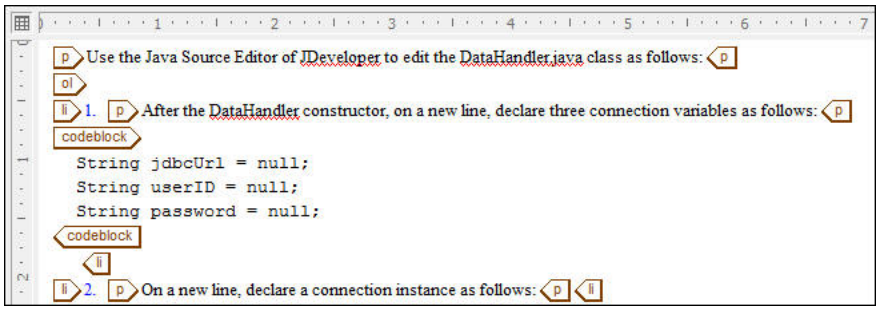
1. In the Edit View, place the cursor where you want to include the informal example, and press Enter.
2. From the Quick Tags menu, select the **codeblock** option.

   A `codeblock` is inserted.
3. Enter the desired code within the `codeblock` tags.

> **Note:** White space inside the `codeblock` is preserved. The listing can span multiple lines.

After following the previous steps, your example should resemble the on in Figure 14-2 .

***Figure 14-2 An Informal Example in p***



# 15

# Inserting Lists

Lists are used to organize, clarify, and emphasize information, and to improve readability.

In technical documentation, lists helps to :

- Identify distinct items of equal importance
- Break up text visually
- Identify sequential steps
- Compress wording that might become repetitive or confusing
- Identify and record the completion of tasks

## About Lists

Use lists to organize, clarify, and emphasize information. Ordered, unordered, simple, and definition lists are available for you to use in <ph varref="Markup"></ph>, each for a different purpose.

The types of lists are described in the following sections.

### Ordered Lists

Ordered list items are marked with numbers or letters. Use ordered lists to present a sequence of events, for example, for step-by-step instructions. The DITA element for an ordered list is `ol`.

***Example 15-1 Ordered List***

Here is a sequence of tasks you must perform to investigate and report a problem:

1. View critical error alerts in Oracle Enterprise Manager and drill down to view problem details.
2. Gather additional diagnostic data.
3. Create a service request.
4. Package and upload diagnostic data to Oracle Support Services.

### Unordered Lists

Unordered lists items are marked with bullets, dashes, asterisks, or any other symbol. Use unordered lists when order is not significant and when all entries are of equal importance. In DITA, the element for these kind of lists is `ul`.

The following is an example of an unordered list:

***Example 15-2 Unordered List***

Many types of birds can be household pets, some of the include:

- Parrots
- Canaries
- Cockatoos
- Pigeons
- Doves

## Simple Lists

Simple list items are not accompanied by any marks. The text in a simple list is shown flush with normal width text or indented to align with first-level lists. In DITA, use the `sl` element to mark up a simple list.

Example 15-3 shows a simple list.

***Example 15-3 Simple List***

Create a job and schedule it for the following weekdays:

Monday
Tuesday
Thursday
Friday

## Definition Lists

In a definition list, each entry is composed of a set of one or more terms and an associated description. In technical documentation, you can think of them as a list of arguments, parameters, or qualifiers, and their explanations.

In DITA, use the `dl` element. It consists of several `dlentry` elements each containing one or more `dt` elements and their corresponding `dd` for its definition. The XML code used to create a definition list is shown next, it is followed by its corresponding output:

```
<dl>
    <dlhead>
        <dthd>Term</dthd>
        <ddhd>Description</ddhd>
    </dlhead>
    <dlentry>
        <dt>b2b.deploy.validation</dl>
        <dd>To turn off validation during deployment, set this
property to
false. Turning off validation is useful when you are deploying a
large
number of agreements and you are certain that the data is valid.
        </dd>
    </dlentry>
    <dlentry>
        <dt>b2b.mdsCache</dt>
        <dd>To set the Metadata Service (MDS) instance cache size,
use this
property. A ration of 5:1 is recommended for the xmxtomdsCache
values.
For example, if the xmx size is 1024, set the MDS cache size to
```

```
200MB.
        </dd>
    </dlentry>
</dl>
```

***Example 15-4 Output for example Definition List***

Set the properties as follows:

**b2b.deploy.validation**
To turn off validation during deployment, set this property to false. Turning off validation is useful when you are deploying a large number of agreements and you are certain that the data is valid.

**b2b.mdsCache**
To set the Metadata Service (MDS) instance cache size, use this property. A ration of 5:1 is recommended for the xmxtomdsCache values. For example, if the xmx size is 1024, set the MDS cache size to 200MB.

## Inserting a Simple List

A simple list is an undecorated list of single words or short phrases. In DITA, use the `sl` element to insert a simple list.

1. In Edit view, place the cursor at the end of the element where you want to insert your list.
2. Press Enter to display the Quick Tags menu.
3. Select the **sl** option to insert a new tag.

   The cursor is automatically positioned at the first list item to insert text.
4. After you enter the text, press Enter to display the Quick Tags menu one more time.
5. Select the **li split** option to insert another `listitem`.
6. Continue to insert list items as needed.

---

**Note:** Although each list item can consist of many elements, such as paragraphs, tables, and more lists, you should keep lists as simple as possible.

---

## Inserting an Unordered List

Unordered lists elements are marked with bullets, dashes or any other symbol. Use unordered lists when order is not significant and when all entries all of equal importance. In DITA, use the `ul` element to insert an unordered list.

1. In Edit pane, place the editor cursor at the end of the element where you want to insert your list.
2. Press Enter to display the Quick Tags menu.
3. Select the **ul** option to insert a new tag.

   The cursor is automatically positioned at the first list item to insert text.
4. Optional: Modify the mark attribute of the list.

The default mark is a bullet but you can change this to a hyphen or other character. Oracle Style Guide recommends using only bullets and hyphens.

a. Select the **ul** element from the Document Map view.

b. Press Ctrl+D to show the Modify Attributes dialog box.

c. Enter an allowed value in the **Mark** field according to the case.

   The allowed values for the **Mark** field are:

   - bullet
   - dash
   - box
   - asterisk

d. Click **OK** to apply changes and close the dialog.

5. After entering the text in the first list item, press Enter to display the Quick Tags menu one more time.
6. Select **li split** to insert another list item.
7. Continue to insert list items as needed.

---

**Note:** Each list item can consist of multiple elements, such as paragraphs, tables, and more lists.

---

## Inserting an Ordered List

Ordered list items are marked with numbers or letters. Use ordered lists to present list items in a specific order or to rank them sequentially.

To insert an ordered list, complete these steps:

1. In Edit view, place the cursor at the end of the element where you want to insert your list.
2. Press Enter to display the Quick Tags menu.
3. Click the **ol** option to insert a new tag.

   The cursor is automatically positioned at the first list item.

4. After you enter the text in the first list item, press Enter to display the Quick Tags menu one more time.
5. Select **li split** to insert another step of the sequence.
6. Continue to insert list items as needed.

---

**Note:** Each list item can consist of many elements, such as paragraphs, tables, and more lists.

---

# Inserting a Definition List

A definition list is a list in which each entry is composed of a set of one or more terms and their associated description. In technical documentation, you can think of them as a list of parameters, or arguments, and their explanations.

In DITA, use the `dl` element to insert a definition list. A definition list consists of several `dlentry` elements. Each `dlentry` element contains a term, entered in the `dt` element, and a definition, within the `dd` element.

1. In Edit View, place the editor cursor at the end of the element where you want to insert your list.
2. Press Enter to display the Quick Tags menu
3. Select the **dl** option to insert a new tag.

   A new definition list is inserted. It is represented as a table where terms and definitions are organized by column, each of them with its default heading.
4. Replace the text in the table headings with the appropriate ones for your content.

   **Note:** You can also delete the whole row and keep a plain table.
5. Define the first term and its definition in the table cells.
6. Add definition list entries by doing one of the following:

   a. Place your cursor in the table. From the Table toolbar, use the **Insert Row Above** or **Insert Row Below** buttons.

   b. Press Tab when the cursor is located in the last cell of the row.

   The entire row is added automatically.

# 16

# Creating Links

You can create links to topics in the repository, to external documents, or within a topic.

## About Links

There are three ways that links are created using LiveContent Architect and the DITA Open Toolkit: automatically based on the topic hierarchy, using relationship tables (reltables), and using the `xref` element.

The system automatically creates links to child and parent topics in the topic hierarchy. Use reltables to create links to other topics in the repository and to external documents. Create links to elements within a topic or to elements in other topics using the `xref` element.

### Links to Child and Parent Topics

In the hierarchy within a DITA map, a parent topic contains its children topics. Our system uses the automatic related links capability of the DITA Open Toolkit, which is based on the topic hierarchy.

The system generates the following links automatically:

- Each parent topic contains links to all of its children topics.
- Each child topic has a link to its parent topic.

Do not create these types of links using reltables or the `xref` element, because it will result in duplicate links in topics.

### Links to Topics in the Repository and External Documents

Use reltables to create links to topics in the repository and external documents. A reltable is a table in a map that defines relationships between topics in the repository or relationships between topics and external documents. The system adds links to topics based on the relationships defined in each row of a reltable.

Reltables provide the following benefits:

- **Central management of links:** Reltables are only allowed in maps. Relationships between topics are defined in one location or a small number of locations instead of in hundreds or thousands of topics.
- **Easier maintenance of links:** Because links are defined in a small number of locations, updating or correcting links is easier.
- **Better reusability of topics:** Dependencies between topics are defined in maps instead in the topics. Therefore, it is easier to reuse topics without violating dependencies.

DITA supports several different types of reltables, but at Oracle, writers only create one type: a one-way reltable that has two columns. A one-way reltable creates links from one topic to another, and each link is from the source only topic to the target only topic. It does not create links from the target only topics to the source only topics.

For example, consider the following reltable:

***Table 16-1 Sample Reltable***

| Source | Target |
|---|---|
| Topicref to Topic 1<br>Topicref to Topic 2 | Topicref to Topic 3 |
| Topicref to Topic A | Topicref to Topic B<br>Topicref to Topic C |

Given this reltable, the system automatically adds the following links:

- In Topic 1, the system adds a link to Topic 3.
- In Topic 2, the system adds a link to Topic 3.
- In Topic A, the system adds a link to Topic B and a link to Topic C.

Each source only topic must be a topic in the repository. A target only topic can be a topic in the repository or an external document.

An external document might be a topic in an Oracle library that is not available in your repository. An external document might also be a publication produced by a third party, such as a technical paper about relational databases. In general, avoid linking to non-Oracle sources unless it is absolutely necessary.

You enter external links in a reltable in one of the following forms:

- To link to a document or a topic in an Oracle library that is not available in your repository, enter `olink:` and the book ID or topic ID for the topic in the form `olink:`*`BOOKID`* or `olink:`*`BOOKIDnnnnn`*, respectively. For example, the book ID for the *Oracle Database Administrator's Guide* is `ADMIN`, and the topic ID for one topic in the book is `ADMIN11013`.
- To link to a publication produced by a third party, use the URL of the publication.

### Links to Elements Within a Topic or in Other Topics

Because most topics are relatively short, links within a topic usually are not necessary. However, you can create links to the following types of elements in a topic:

- Figure
- Table
- Example
- Step

You create links to these elements by inserting an inline `xref` tag and selecting the element to which you want to link.

You also create links to elements in other topics with an inline `xref` tag. In this case, the target element in the other topic must be a figure, table, or example.

**Note:** Only use an inline `xref` tag to create links to elements within topics. Do not use an inline `xref` tag to create links to topics or publications.

**See Also:**

Creating Links to Topics Using Arbortext Editor
: Using Arbortext Editor, you add topic references (topicrefs) to a relationship table (reltable) to define relationships between topics. The system adds links to topics automatically based on these relationships.

Creating Links to Topics Using Publication Manager
: Using Publication Manager, you add topic references (topicrefs) to a relationship table (reltable) to define relationships between topics. The system adds links to topics automatically based on these relationships.

Creating Links Within a Topic
: You create links within a topic using the `xref` tag.

Creating Links to Elements in Other Topics
: You create links to elements in other topics using the `xref` tag. The element to which the link points must have GUID.

Creating Links to External Documents
: External documents include Oracle documents and topics that are not part of your repository and documents created by third parties. You link to external documents in a relationship table (reltable).

## Creating a Reltable

You add relationship tables (reltables) to maps to define relationships between topics. The system adds links to topics automatically based on these relationships.

You can create a one-way, two column reltable to define links from source only topics to target only topics.

Before creating a reltable, you must have access to the map that will contain the reltable, and you must have the required permissions to modify the map.

1. Locate and check out the map that will contain the reltable.
2. In Arbortext Editor, place your cursor after the last topicref in the map.
3. From the **Insert** menu, select **Relationship Table**.

   A reltable is inserted in the map with a `Source` column and a `Target` column.
4. Check in the map to save your changes.

**See Also:**

About Links
: There are three ways that links are created using LiveContent Architect and the DITA Open Toolkit: automatically based on the topic hierarchy, using relationship tables (reltables), and using the `xref` element.

## Creating Links to Topics Using Arbortext Editor

Using Arbortext Editor, you add topic references (topicrefs) to a relationship table (reltable) to define relationships between topics. The system adds links to topics automatically based on these relationships.

In a one-way, two column reltable, you can define links from source only topics to target only topics. The topics in the left column are source only, and the topics in the right column are target only.

Before creating links in a reltable, the reltable must exist, you must have access to the map that contains the reltable, and you must have the required permissions to modify the map.

1. In the repository, locate and check out the map that contains the reltable.
2. If necessary, in Arbortext Editor, add a row to the reltable by placing your cursor in a current row and selecting **Table** > **Insert** > **Row Below**.
3. Insert one or more source only topicrefs.
   a. Place your cursor in the left cell of a row in the reltable.
   b. Press Enter on your keyboard, and select **topicref**.
   c. In the **Insert Link** dialog box, locate and select the topic in the repository.
   d. Click the **Insert** button.

      The topicref appears in the cell of the reltable.
   e. Add more source only topicrefs if necessary by repeating the previous steps.

      Add the topicrefs to the same cell if you want to create multiple sources for the targets in the other cell in the row.

      Add the topicrefs to a new row to define new relationships between topics.
4. Insert one or more target only topicrefs.
   a. Place your cursor in the right cell of a row in the reltable.

      **Note:** The cell must correspond with the left cell that contains the source only topicrefs.
   b. Press Enter on your keyboard, and select **topicref**.
   c. In the **Insert Link** dialog box, locate and select the topic in the repository.
   d. Click the **Insert** button.

      The topicref appears in the cell of the reltable.

e. Add more target only topicrefs if necessary by repeating the previous steps.

   Add the topicrefs to the same cell if you want to create multiple targets for the sources in the other cell in the row.

   Add the topicrefs to a new row to define new relationships between topics.

**See Also:**

Creating a Reltable
: You add relationship tables (reltables) to maps to define relationships between topics. The system adds links to topics automatically based on these relationships.

About Links
: There are three ways that links are created using LiveContent Architect and the DITA Open Toolkit: automatically based on the topic hierarchy, using relationship tables (reltables), and using the `xref` element.

## Creating Links to Topics Using Publication Manager

Using Publication Manager, you add topic references (topicrefs) to a relationship table (reltable) to define relationships between topics. The system adds links to topics automatically based on these relationships.

In a one-way, two column reltable, you can define links from source only topics to target only topics. The topics in the left column are source only, and the topics in the right column are target only.

Before creating links in a reltable, the reltable must exist, you must have access to the map that contains the reltable, and you must have the required permissions to modify the map.

1. With the publication open in Publication Manager, locate the reltable to which you want to add links in the tree view.

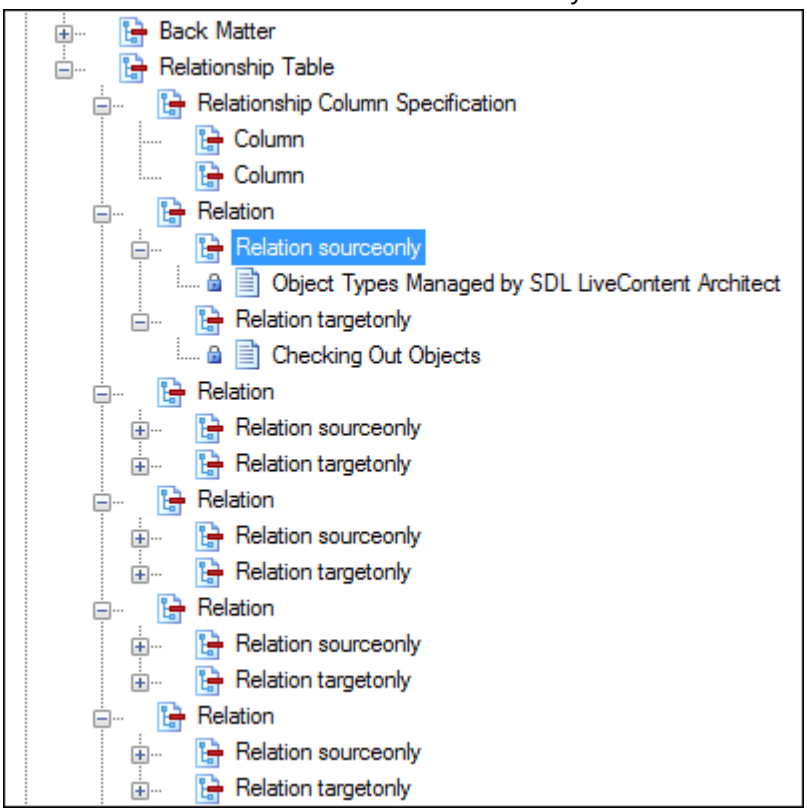
   **Note:** Reltables are located after the Back Matter of the publication in the tree view.

2. If necessary, add a row to the reltable by right-clicking the reltable and selecting **Add Within** > **Relationship Row**.

3. Insert one or more source only topicrefs.

   a. Locate the source only cell into which you want to insert the topicrefs.

A source only cell is labeled `Relation sourceonly`. The following image shows a reltable with a `Relation sourceonly` cell selected.




**b.** Right-click the source only cell, and select **Add Within** > **Topic Ref**.

**c.** In the Insert Values for 'Topic Ref' dialog box, click the **Browse** button.

**d.** In the Insert Link dialog box, locate and select the topic.

**e.** Click the **Insert** button.

**f.** In the Insert Values for 'Topic Ref' dialog, click the **OK** button.

The topicref appears in the cell of the reltable.

**g.** Add more source only topicrefs if necessary by repeating the previous steps.

Add the topicrefs to the same cell if you want to create multiple sources for the targets in the other cell in the row.

Add the topicrefs to a new row to define new relationships between topics.

4. Insert one or more target only topicrefs.

**a.** Locate the target only cell into which you want to insert the topicrefs.

A target only cell is labeled `Relation targetonly`.

**b.** Right-click the target only cell, and select **Add Within** > **Topic Ref**.

**c.** In the Insert Values for 'Topic Ref' dialog box, click the **Browse** button.

**d.** In the Insert Link dialog box, locate and select the topic.

e. Click the **Insert** button.

f. In the Insert Values for 'Topic Ref' dialog box, click the **OK** button.

   The topicref appears in the cell of the reltable.

g. Add more target only topicrefs if necessary by repeating the previous steps.

   Add the topicrefs to the same cell if you want to create multiple targets for the sources in the other cell in the row.

   Add the topicrefs to a new row to define new relationships between topics.

**See Also:**

Creating a Reltable
: You add relationship tables (reltables) to maps to define relationships between topics. The system adds links to topics automatically based on these relationships.

About Links
: There are three ways that links are created using LiveContent Architect and the DITA Open Toolkit: automatically based on the topic hierarchy, using relationship tables (reltables), and using the `xref` element.

Opening a Publication
: You open a publication to view its structure, modify it, view or modify its content, or publish it.

## Creating Links to External Documents

External documents include Oracle documents and topics that are not part of your repository and documents created by third parties. You link to external documents in a relationship table (reltable).

In a one-way, two column reltable, you can define links from source only topics to external documents. The topics in the left column are source only, and the links to external documents in the right column are target only.

Before creating links in a reltable, the reltable must exist, you must have access to the map that contains the reltable, and you must have the required permissions to modify the map.

1. Locate and check out the map that contains the reltable.
2. If necessary, in Arbortext Editor, add a row to the reltable by placing your cursor in a current row and selecting **Table** > **Insert** > **Row Below**.
3. Insert one or more source only topicrefs.

   a. Place your cursor in the left cell of a row in the reltable.

   b. Press Enter on your keyboard, and select **topicref**.

   c. In the Insert Link dialog box, locate and select the topic.

   d. Click the **Insert** button.

      The topicref appears in the cell of the reltable.

e. Add more source only topicrefs if necessary by repeating the previous steps.

   Add the topicrefs to the same cell if you want to create multiple sources for the targets in the other cell in the row.

   Add the topicrefs to a new row to define new relationships between topics.

4. Insert one or more external target only topicrefs.

   a. Place your cursor in the right cell of a row in the reltable.

   **Note:** The cell must correspond with the left cell that contains the source only topicrefs.

   b. Press Enter on your keyboard, and select **anchorref**.

      The anchorref appears in the table.

   c. Place your cursor in the newly inserted anchorref.

   d. From the **Edit** menu, select **Modify Attributes**.

   e. In the Modify Attributes dialog box, select the **Reference** tab.

   f. In the **href** field, enter an `olink:` to an Oracle topic or a URL to a third-party document.

      To link to a document or a topic in an Oracle library that is not available in your repository, enter `olink:` and the book ID or topic ID for the topic in the form `olink:`*`BOOKID`* or `olink:`*`BOOKIDnnnnn`*, respectively. For example, the book ID for the *Oracle Database Administrator's Guide* is `ADMIN`, and the topic ID for one topic in the book is `ADMIN11013`.

      To link to a publication produced by a third party, enter the URL of the publication.

   g. In the **navtitle** field, enter the title of the topic or document to which you are linking.

   h. In the **scope** list, select **external**.

   i. In the **format** field, enter `html`.

   j. Click the **OK** button.

      The anchorref appears in the cell of the reltable.

   k. Add more target only anchorrefs if necessary by repeating the previous steps.

      Add the anchorrefs to the same cell if you want to create multiple targets for the sources in the other cell in the row.

      Add the anchorrefs to a new row to define new relationships between topics.

**See Also:**

Creating a Reltable
: You add relationship tables (reltables) to maps to define relationships between topics. The system adds links to topics automatically based on these relationships.

About Links
: There are three ways that links are created using LiveContent Architect and the DITA Open Toolkit: automatically based on the topic hierarchy, using relationship tables (reltables), and using the `xref` element.

## Creating Links Within a Topic

You create links within a topic using the `xref` tag.

The link must point to one of the following types of elements:

- Figure
- Table
- Example
- Step

If the target element is a figure, table, or example, then the target element must include a title element. If the target element is a step, then the step number appears in the output. Therefore, add the word “Step” or “step” a link to a step.

1. Check out the topic.
2. Ensure that the element to which you are linking in the other topic has an ID (either an Arbortext Editor ID or a GUID). If it does not, then complete the following steps to generate one for the element:
   a. Place your cursor in the element.
   b. Click the magic wand icon.

   c. From the **File** menu, select **Save**.

> **Note:** Saving the topic populates the ID you created in the repository.

3. In the topic in Arbortext Editor, place your cursor where you want to create the link.
4. Press Enter on your keyboard, and select **xref**.
5. In the Insert Hyperlink dialog, locate and select the topic in which you are creating the link.

**Note:** The selected topic must be the same topic you checked out to start this task.

6. Click the **Bookmark** button.
7. In the Select dialog box, locate target element, and select the gray box above it.

   Select only a gray box labeled either `fig`, `table`, `example`, or `step`. Do not click the box labeled `title`.

   An orange box surrounds the element when it is selected.
8. Click the **OK** button.
9. In the Insert Hyperlink dialog box, clear the text in the **Text to display** field.
10. Click the **Insert** button.

    The `xref` tag appears in the topic. If the reference is to a step, then remember to put the word "Step" or "step" before the `xref` tag.
11. Ensure that there is no text between the opening and closing `xref` element tags. If text is present, then delete it.
12. Optional: Check in the topic to save your changes.

**See Also:**

About Links
: There are three ways that links are created using LiveContent Architect and the DITA Open Toolkit: automatically based on the topic hierarchy, using relationship tables (reltables), and using the `xref` element.

# Creating Links to Elements in Other Topics

You create links to elements in other topics using the `xref` tag. The element to which the link points must have GUID.

When you create a link to an element in a different topic, the topic that contains the link is the source topic, and the element in the other topic to which you are linking is the target element.

The link must point to one of the following types of target elements:

- Figure
- Table
- Example

In addition, the target element must include a title element and must have a GUID. These elements get a GUID automatically when the topic that contains them is checked into the repository.

1. Check out the source topic.
2. In Arbortext Editor, place your cursor where you want to create the link.
3. Press Enter on your keyboard,and select **xref**.

4. In the Insert Hyperlink dialog box, locate and select the topic that contains the target element.
5. Click the **Bookmark** button.
6. In the Select dialog box, locate the target element and select the gray box above it.

   Select only a gray box labeled either `fig`, `table`, or `example`. Do not click the box labeled `title`.

   An orange box surrounds the element when it is selected.
7. Click the **OK** button.
8. In the Insert Hyperlink dialog box, clear the text in the **Text to display** field.
9. Click **Insert**.

   The `xref` tag appears in the topic.
10. Ensure that there is no text between the opening and closing `xref` element tags. If text is present, then delete it.
11. Optional: Check in the topic to save your changes.

**See Also:**

About Links

There are three ways that links are created using LiveContent Architect and the DITA Open Toolkit: automatically based on the topic hierarchy, using relationship tables (reltables), and using the `xref` element.

# 17

# Creating Notes

Use note elements to draw attention to important information that readers may miss when they scan content quickly. The InfoDev customizations use different note types, depending on the meaning of the content in the note.

The following kinds of notes are available in DITA:

- Note
- Caution
- Warning
- Tip
- See also (created with tip)

## About Notes in DITA

In DITA, you can create Note, Tip, Caution, Warning, and See Also notes in your document. In Arbortext Editor, use the correct type of note markup to define and format the type of note you want to insert and the information you present in the note. The format of each note type is generated during document output in accordance with the Oracle Style Guide rules.

When you create notes in DITA using Arbortext Editor, you insert the `note` markup and then edit the markup to select the appropriate note for your purpose.

## Types of Notes Used in DITA

The InfoDev customizations use four types of notes, each of which has a particular semantic purpose. These note types are `note`, `warning`, `caution`, and `tip`. Use the existing semantic term for each note type. Do not create new types of notes in your document. These note types must mean the same thing across all documentation.

***Table 17-1 Types of Notes in DITA***

| Note Type | Usage | Output Label |
|---|---|---|
| Note | Includes important hits, guidance, or advice on using a program. | **Note:** |
| Caution | Indicates the possibility of damage to a program, system, or data. When used in a procedure, it should precede the step where the risk occurs. | **Caution:** |

| Note Type | Usage | Output Label |
|---|---|---|
| Warning | Indicates a potential hazard to people. When used in a procedure, it should precede the step where the risk occurs. When the output is built, the text is shown in bold typeface.<br>Legally, warning notices can only be used to convey information about threat to people. Using otherwise can cause Oracle a legal problem. | **Warning:** |
| Tip | | **Tip:** |
| See Also | Used to include a cross-reference to a related topic. | **See Also:** |

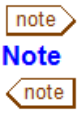
# Inserting Notes in DITA

To insert notes into your documentation, use the `note` element when editing a topic.

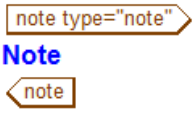
1. Working in Arbortext Editor, place the cursor where you want to insert the note in the text.
2. From the **Insert** menu, click **Markup**.

   The Insert Markup dialog box displays.
3. Select `note` from the list of markup elements and click **Insert**.

   The `note` element displays in the text, as follows:

   
   
4. Place the cursor anywhere within the `note` element tags and click **Edit** > **Modify Attributes...**.
5. Under the **Reference** tab, select **caution**, **note**, **tip**, or **warning** from the **type** dropdown menu, depending on the type of note you want to insert.

   For example, if you want to insert a Note, then the markup will be as follows:

   
   
6. Place the cursor anywhere within the `note` element tags and type your note text.

# 18

# Creating Footnotes

The DocBook `footnote` element enables you to provide complementary information without disrupting the flow of a document.

## About Footnotes in DITA

Use the DITA`fn` element to insert footnotes in your documents. The `fn` element is an inline element which means that it can be inserted within block elements such as paragraphs, tables, and lists without breaking the document flow.

Footnotes inserted in paragraphs or lists follow a numbering sequence throughout the whole document. When you insert an `fn` element in Arbortext Editor, its contents are shown as its child elements both in Document Map and Edit view. However, when document output is generated, the contents of a footnote are located at the bottom of a page.

#### Table Footnotes

Footnotes are inserted in a table at `entry` level (a single cell in the table). Their purpose is to provide explanations or complementary information to the contents of a cell. The processing of a table footnote and a footnote in the paragraph or list differs in two ways:

- Table footnote numbering is restarted in every instance of a table.
- Table footnotes are displayed immediately following the table itself, not at the bottom of a page.

## Inserting Footnotes

Use the DocBook `footnote` element to insert footnotes in your documents. The `footnote` element can be inserted within block elements such as paragraphs, tables, and lists.

1. In Edit view, position the cursor at the end of the element where you want to insert the `footnote` and press Enter.
2. In the Quick Tags menu, select the `footnote` option to insert a new tag.

   A new pair of tags is inserted, with the cursor automatically positioned next to the opening tag.
3. Using the Quick Tags menu once more, insert the elements required by the content of the footnote you want to include.

**Note:** Elements permitted within a footnote include:

- Paragraphs
- Lists
- Informal figures
- Informal tables
- Informal code examples

# 19

# Creating an Index

A book index is generated automatically after the whole document is processed and its output is generated. Every **index term** that you mark in the body of the document produces an index entry in the document index. Index entries can be **primary**, **secondary**, or **tertiary**. These categories determine the hierarchy of each term within the index list.

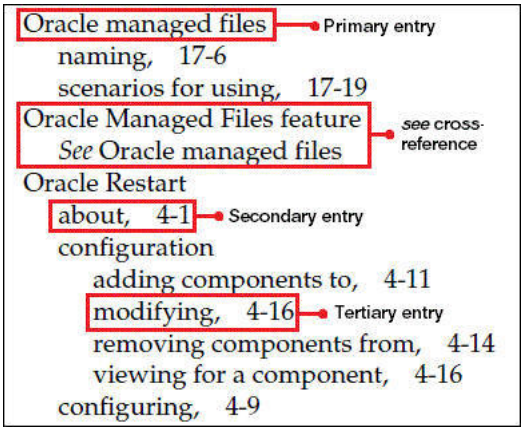
## About Index Elements

The index is a major part of a document interface. It provides different paths to any relevant information that a reader might want to consult. The document index itself is a piece of generated text. A writer produces **index entries** by first inserting **index terms** and **index cross-references** in the prolog of each topic and then publishing the publication.

An index is an organized list of terms and references. The art of designing and creating an index consists of making it accurate, complete, and consistent. Furthermore, a good index provides readers with different context and levels of experience several ways of finding a piece of information.

The process of generating an index consists of collecting **index terms** from the prolog of each topic and organizing them into a list. An **index entry** (an item in that list) is created for each index term. It directs the reader to the location in the document where the term is found. Figure 19-1 shows a portion of a printed index with its main components labeled.

***Figure 19-1 Components of a Document Index***




In DITA, you can insert index entries anywhere in a topic. However, in almost all cases, you should insert them in the `keyword` element of the `prolog` element. Placing index entries in the `prolog` element ensures that they are in a single location in each topic, which improves the maintenance of topics. Place index entries outside of the `prolog` element only when you have a very long topic, and you want a reader to be directed to a specific location when clicking the index entry in the output.

**See Also:**

Inserting Index Entries for Terms
: You insert `indexterm` elements as keywords in the prolog of a topic to insert index entries. You nest `indexterm` element to insert secondary and tertiary index entries.

Inserting See and See Also Index Entries
: You insert `index-see` and `index-see-also` elements inside of `indexterm` elements to create cross-reference index entries.

## Index Terms and Index Entries

The DITA `indexterm` element marks a term to be included in the index. One index term generates one index entry. An index term with more than one occurrence generates one index entry with one reference for each occurrence.

Different kinds of index entries generate different `indexterm` structures:

- **Primary Entry**: A key concept, topic, or feature in the document. The text of this element is shown as the main index term. The following element structure is used to produce a primary index entry. It corresponds to the entry marked as a primary entry in Figure 19-1 .

  ```
  <indexterm>Oracle managed files</indexterm>
  ```

- **Subentry**: An extension of a primary entry. Each subentry further discusses characteristics or procedures involving the main entry. The text of a subentry is shown indented below the term from which it is derived.

  In DITA, subentries can be either **secondary** or **tertiary**. The `indexterm` element is nested to produce secondary and tertiary index entries. The following element structures correspond to the item marked as a secondary entry and a tertiary entry in Figure 19-1 .

  The following element structure is used to produce a secondary index entry:

  ```
  <indexterm>Oracle Restart<indexterm>configuration</indexterm></
  indexterm>
  ```

  The following element structure is used to produce a tertiary index entry:

  ```
  <indexterm>Oracle
  Restart<indexterm>configuration<indexterm>modifying</indexterm></
  indexterm></indexterm>
  ```

Always consider the Oracle Style Guide recommendations when you design and create your document's index entries.

## Index Cross-References

Index cross-references help readers to connect terms that they already know to new or standard terms used in the documentation. Cross-references also enhance the understanding of a document by showing the relationships between different subjects.

There are two kinds of index cross-references:

- **See** reference: Directs the reader from an alternative or synonym term to the standard term chosen in the documentation.

- **See Also** reference: Suggests readers one or more additional entries that might add or complement the current index entry.

In DITA, a **See** or **See Also** cross-reference is produced by first creating an index entry and then adding an `index-see` or `index-see-also` element before the closing `indexterm` tag. The following code example is used to generate the entry marked as a see cross-reference in Figure 19-1 . The first term (Oracle Managed Files feature) is the synonym term; the `index-see` element contains the standard documentation term.

```
<indexterm>Oracle Managed Files feature<index-see>Oracle managed
files</index-see></indexterm>
```

## Inserting Index Entries for Terms

You insert `indexterm` elements as keywords in the prolog of a topic to insert index entries. You nest `indexterm` element to insert secondary and tertiary index entries.

For most topics, index entries should be placed in the `prolog` element, and these steps describe placing index entries there. However, for very long topics, you can place index entries in the body of the topic so that a reader is directed to the correct location in the topic after clicking the index entry in the output.

1. Check out the topic to which you want to add index entries.

   Arbortext Editor displays the topic.

2. If it does not exist, then insert a `prolog` element after the `shortdesc` element.

   The prolog element must contain a `metadata` element, and the `metadata` element must contain a `keywords` element. Create these elements also if they do not exist.

3. Place your cursor inside the **keywords** element and press `Enter` on your keyboard.

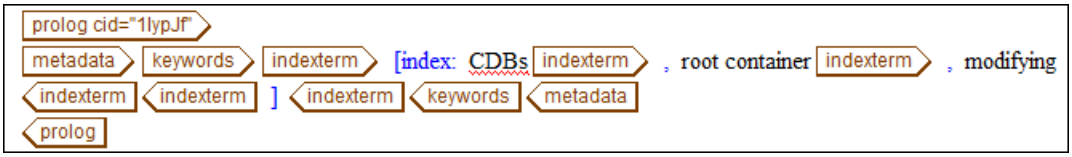
4. In the **Quick Tags** menu, select **indexterm**.

5. Enter the primary index term.

6. To create secondary and tertiary index terms, nest `indexterm` elements inside other `indexterm` elements, and enter the index entry for each level.

7. Check in the topic to save your changes.

***Figure 19-2 Sample Index Entries***




**See Also:**

About Index Elements

The index is a major part of a document interface. It provides different paths to any relevant information that a reader might want to consult. The document index itself is a piece of generated text. A writer produces

**index entries** by first inserting **index terms** and **index cross-references** in the prolog of each topic and then publishing the publication.

## Inserting See and See Also Index Entries

You insert `index-see` and `index-see-also` elements inside of `indexterm` elements to create cross-reference index entries.

1. Check out the topic to which you want to add the See and See Also index entries.

   Arbortext Editor displays the topic.

2. If it does not exist, then insert a `prolog` element after the `shortdesc` element.

   The prolog element must contain a `metadata` element, and the `metadata` element must contain a `keywords` element. Create these elements also if they do not exist.

3. Place your cursor inside the **keywords** element and press `Enter` on your keyboard.

4. In the **Quick Tags** menu, select **indexterm**.

5. Enter the index term for which you want to create a See or See Also entry.

   For a See entry, enter the synonym term. For a See Also entry, enter the related term.

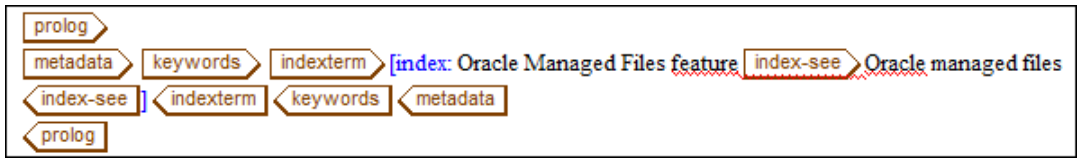
6. Place your cursor inside the `indexterm` element you created in the previous step and press `Enter` on your keyboard.

7. In the **Quick Tags** menu, select **index-see** or **index-see-also**.

8. Enter the cross-reference term.

9. Check in the topic to save your changes.

***Figure 19-3 Sample See Index Entry***




**See Also:**

About Index Elements

The index is a major part of a document interface. It provides different paths to any relevant information that a reader might want to consult. The document index itself is a piece of generated text. A writer produces **index entries** by first inserting **index terms** and **index cross-references** in the prolog of each topic and then publishing the publication.

# 20

# Creating a Glossary

A glossary is a collection of terms and their definitions inserted at the end of a document, it helps readers to comprehend specialized terminology found in the document. Create a glossary in your book where you can add, one by one, the key terms and its definitions. Afterwards, mark the first appearance of each term in the document and link it to its definition within the glossary.

## About a Glossary and Its Structure

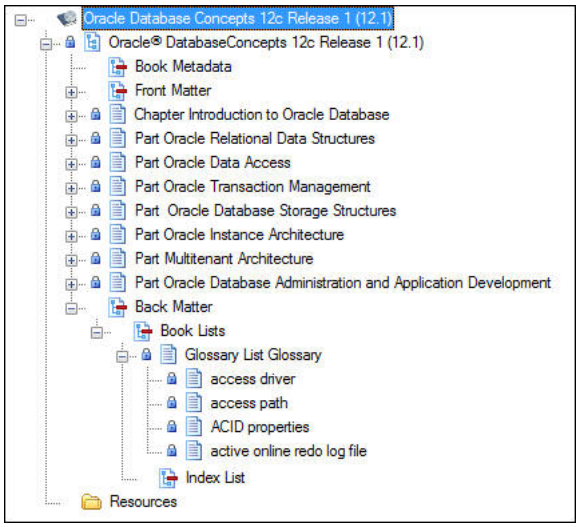
A glossary groups new or unusual terms used within a document and its corresponding definitions. Typically, these terms are key to understand the content of a document. A glossary is usually inserted at the end of a document where the reader consults it as a reference to unfamiliar terminology.

In DITA, the `glossarylist` element groups a set of references that point to different **glossary entry** topic objects. Each glossary entry topic contains an **entry** and its **definition**.

The `glossarylist` finds its place within the `backmatter` of a `bookmap`, as a child of a `booklist` element. The following example shows the typical `backmatter` element structure that generates a glossary:

```
<backmatter>
    <booklists>
        <glossarylist>
            <topicref format="dita" href=""GUID1">...</topicref>
            <topicref format="dita" href=""GUID2">...</topicref>
                .
                .
                .
            <topicref format="dita" href=""GUIDN">...</topicref>
        </glossarylist>
    </booklists>
</backmatter>
```

***Figure 20-1 A Glossary in a DITA Bookmap***




**See Also:**

About Text Formatting in DITA

To format inline text, you enclose the text in one of three kinds of tags: semantic tags, typographic tags, and the `codeph` tag. Inline text is a word or phrase within a sentence. Examples include function names, database view names, or UI elements such as menu names or button names.

## The Glossary Entry Topic Type

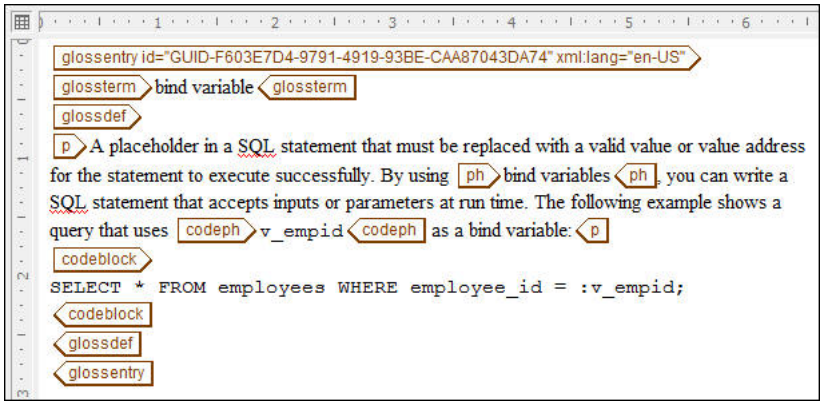
A glossary entry is a kind of topic that contains a single glossary term and its definition. The glossary entry topic contains the `glossterm` and `glossdef` elements, the former encloses the actual glossary term while the latter contains its definition. After a glossary entry topic is created, it can be included in the glossary of a bookmap by adding a `topicref` element that references it.

---

**Note:** You can expand the term definition by using different elements such as lists, tables, and code examples.

---

***Figure 20-2 A Glossary Term Defined in a Glossary Entry Topic***



## Creating a Glossary Entry Topic

A glossary entry topic contains a single glossary term and its definition. Include a glossary entry topic in a bookmap's glossary by referencing it with a `topicref` inside a `glossarylist`.

1. In the Browse Repository dialog box and under the **Repository** tab, select the folder where you want to create the new topic.

   **Note:** The selected folder must be of the Module content type.

2. Click the **New Object** button.

3. In the Select Template dialog box, under the tab for your group, select the **Glossary Entry** template. For example, if you are a database writer, then select the template under the **Database Topics** tab.
4. In the Add Object dialog box, fill the required metadata fields according to the table below:
5. Click **OK**.

   The new topic object is now listed in the right pane of the dialog box.
6. In the Browse Repository dialog box, check out the newly created topic.

   Arbortext Editor loads the document.
7. In Arbortext Editor, edit the document as needed:
   a. Enter the glossary term within the `glossentry` tags.
   b. Enter the graphic description within the `glossdef` tags.

   **Note:** You can use several element such as paragraphs, lists, or tables to enhance the term definition.

8. Check the topic into the repository.
   a. From the SDLLiveContent menu, select the **Check in** option.
   b. In the Close dialog box, select the **Check in current object?** option.
   c. Click OK.

The glossary entry object is now ready to referenced from a bookmap.

## Creating a Glossary

Create a glossary in a publication by inserting a `glossarylist` element in the publication's bookmap. Within the `glossarylist`, insert one `topicref` for each

glossary term that you want to include in the glossary, the `topicref` should reference the glossary entry object that corresponds to the glossary term.

The `glossarylist` is child of a `booklists` element which is a contained by the `backmatter`.

1. Insert a `booklists` element within the `bookmap`'s backmatter:
   a. Right-click the **Back Matter** icon inside the publication's bookmap.
   b. In the context menu, select the **Add Within** submenu followed by the **Book Lists** option.

      A `booklists` element is inserted inside the Back Matter.
2. Insert a `glossarylist` element within the Book Lists:
   a. Right click the **Book Lists** icon inside the bookmap's back matter.
   b. In the context menu, select the **Add Within** submenu followed by the **Glossary List** option.

      A `glossarylist` element is inserted within the Book Lists.
3. Add a `topicref` within the Glossary List that references a glossary entry topic:
   a. Right-click the **Glossary List** tag inside Book Lists.
   b. In the context menu, select the **Add Within** submenu followed by the **Topicref** option.
   c. In the Insert Link dialog box, do one of the following to locate and select the glossary entry object that you want to include in the glossary.
      - Under the Repository tab, expand the hierarchy as needed to select a folder. Then select the desired glossary entry object from the right pane.
      - Under the Search tab, search for a glossary entry object by its metadata values and select it from the right pane.
   d. Click the **Insert** button.

      A `topicref` is inserted within the Glossary List. It links to the selected Glossary Entry object.

      **Note:** Alternatively, you can insert `topicref`s directly to the Glossary List by dragging a Glossary Entry object from the Browse Repository dialog box and dropping it in the correct location of the publication tree in Publication Manager.
4. Continue to insert glossary entries as needed by repeating step 3 .

# Part IV

## Reusing Content

# 21

# Overview of Reusing Content

A key advantage of DITA is the ability to reuse content. Reusing content improves the efficiency of our information development group and promotes consistency in our content.

- Content can be reused at the map level, the topic level, and the element level
- Reusing content at the map level means creating a submap of topics and reusing that submap in various publications.
- Reusing content at the topic level means using a topic in multiple publications or multiple submaps. Different maps have `topicrefs` that point to the same topic in the repository.
- Reusing content at the element level is done with DITA "content references" or "conrefs." This means reusing a paragraph, note, table, list, or other element in multiple topics. A common use case is a Warning that is approved by legal and must be used in many publications with the exact wording.

  You can create libraries of conrefs (such as a library of commonly used notes), or you can conref any element from one topic to another.
- Conditionalizing content provides more opportunities for reuse.

## Overview of Conditionalizing Content

Conditionalizing content provides more opportunities for reuse.

For example, if a topic could be reused in multiple publications if only one paragraph or one product name could be different for each publication, you could conditionalize that paragraph or product name so that different versions of the paragraph or different product names appeared in the different publications. This would enable you to create just one instance of the topic and reuse it. The alternative would be to create a separate topic for each publication, where all of the content of the topics is the same except for the one paragraph. However, this alternative causes a maintenance issue: If content in a different part of the topic had to change, you would have to change multiple topics.

You should use conditionalized content judiciously. A topic where every other sentence is conditionalized is another kind of maintenance issue. For such a topic, multiple copies of the topic, where each topic differs as needed, is a better solution.

There are two ways to conditionalize content in Architect: conditions and variable.:

### Conditions

A condition is assigned to a DITA element to include or exclude it from a publication, depending on publication settings. A condition is an identifier that has a name and a value. You assign a condition to an element by adding an `ishcondition` attribute to the element. The attribute value is an expression of the form *condition_name=value*. An

example is `ishcondition="Cloud=Y"`. In this case, the condition name is `Cloud` and its value is `Y`. If the condition value does not match settings for that condition in the publication, the conditionalized element is excluded from the output.

In addition to conditionalizing individual DITA elements, you can conditionalize (include or exclude) an entire topic or submap by assigning a condition to its `topicref` in the map.

A selection in the **SDLLiveContent** menu in Arbortext makes it easy to apply conditions.

#### Variables

A variable is a DITA element whose value is determined by attaching a **library** to the publication. A library is an Architect object that is similar to a topic, but that contains only variable names and their values. For example, you could have a topic that lists prerequisites for installing Oracle Database. The elements in the topics that contain values for, say, minimum physical memory, minimum storage, and minimum operating system version are created as variables, where each variable has a different name. You would then create a library for each platform—Linux, Solaris, Windows, and so on—where each library contains the same variable names but different variable values. You attach the Linux library to the Linux install guide, the Solaris library to the Solaris install guide, and so on, and the variable values in the topic would be resolved by the attached library.

### When to Use Variables and When to Use Conditions

Comparative of the two way of conditionalizing and offer advice

Variables, as you know, are an elegant way of conditionalizing content. It is an alternative to using conditional text (CCMS conditions), and in some cases is the better solution. There are several use cases for this just in the install guides. One "installation prereqs" variable library could contain variables for minimum RAM required, minimum OS level required, list of patches required, and so on. We could then have one such library for each platform: Linux, Solaris, HP-UX, and so on. Each library would have the same variables (the same variable names), but different values. For the Solaris install guide, you attach the Solaris library to the CCMS publication as a resource, for the Linux install guide you attach the Linux library, and so on.

#### When to use variables

restrictions/benefits

#### When to use conditions

restrictions/benefits

## Concept Title

(Required) Enter introductory text here, including the definition and purpose of the concept.

#### Section Title

(Optional) Enter conceptual text here.

##### *Example 21-1 Example Title*

(Optional) Enter an example here.

# 22

# Conditionalizing Content Using Variables and Variable Libraries

Implementing variables in topics provides a mean of reusing them in different publications with minimum changes involved. For example, when writing two documents referring to related products that differ only in small chunks of content (such as product name or screenshots) variables allow to generate different publications from the same DITA source.

Use variables to conditionalize:

- Paragraphs
- List items
- Steps
- Images
- Tables
- Notes
- Code examples

## About Using Variables in SDL LiveContent Architect

SDL LiveContent Architect interface enables you to easily insert variable references when authoring content. However, the libraries that include the variable definitions must be created using the InfoDev library templates.

### The XML Structure of a Variable

A **variable** is an uniquely identified DITA element defined in a **library** that is associated with a publication. Topics that are part of the publications can include **variable references**, each pointing to a **variable definition**. When a output is generated from the publication, the **value** of a variable is inserted at the location of each reference.

#### The Variable Definition

A variable definition consists of three parts:

- An instance of a DITA element. In SDL, different DITA elements are used as variables. The kind of element that you use depends on the purpose of the variable and the content that it wraps.
- The name of the variable. The name of the variable is stored in the `varid` attribute of the DITA element used as a variable. The name must be unique within the library.

- The value of the variable. The value of a variable is all the content (text and markup) included within the opening and closing tags of the DITA element used as variable. In the case of an image variable, the value is stored in the `href` attribute while the contents of the tag remain empty.

Example 22-1 shows a `ph` element used to define a variable named `homepage` whose value is the string `Cloud Home Page`.

***Example 22-1 A DITA Phrase Element Used as a Variable Definition***

```
<ph varid="homepage">Cloud Home Page</ph>
```

### The Variable Reference

After the variable is associated to a publication, it can be referenced from any topic that is a part of the same publication. You insert a placeholder (another DITA element of the same kind as the one used to define the variable) in the topic where you want to include the variable contents and point it to the variable definition within the library. This reference is done by setting the name of the variable (`varid`) as the placeholder's `varref` attribute. For example, if you want to reference the variable defined in Example 22-1 you would insert the following reference in your topic:

```
<ph varref="homepage"></ph>
```

When the document is processed, the value of the variable is inserted at the position of the variable reference. Fortunately, the SDL Live Content Authoring Bridge facilitates the process of implementing variables by providing a graphical interface that minimizes the manipulation of XML code.

## Variable Libraries

In the CCMS context, variable libraries are generic topic objects used to store a set of related variable definitions. Libraries are associated to publications as resources; the resources attached to a publication determine the value of a variable in the publication context.

There are several advantages of using libraries (such as improving maintainability of a document by keeping all variables in a single location) but the most important one is the ability easily change the variable definitions in a publication simply by replacing an attached library for a new one.

### Types of Libraries in the InfoDev Organization

When writing documentation for the Information Development organization, you create variables using templates designed by our information architects. The template that you use to create a new library depends on the kind of content that the variable holds. Table 22-1 lists the InfoDev library templates and describes the purpose of the library along with the DITA element used in the variable definitions.

***Table 22-1 InfoDev Library Templates***

| Template | DITA Element | Description |
|---|---|---|
| Library of Image Variables | `image` | The contents are stored in the `href` attribute of the element. The element contents must be left empty. |

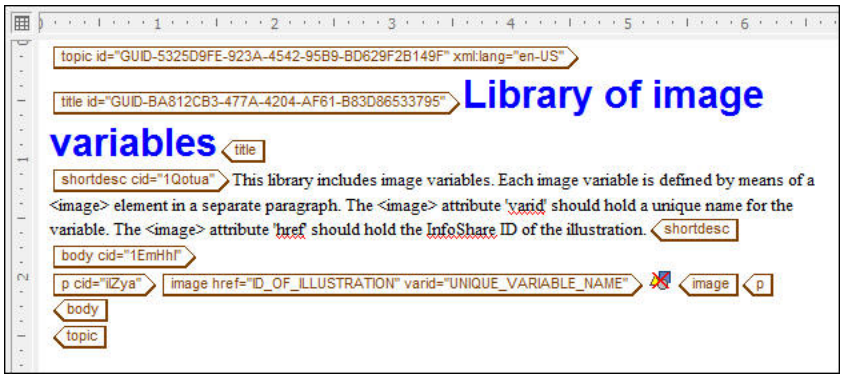
| Template | DITA Element | Description |
|---|---|---|
| Library of Text Variables | ph | |

### Structure of a Variable Library

New library objects are loaded with the basic library element structure as shown in Figure 22-1 . The `shortdesc` element contains a description of the library template and instructions on how to add variables. The most important part of the library is its `body` because it contains all the variable definitions. This is where you create, modify, and delete variables. The rest of the document should not be modified.

**Note:** Insert all variable definitions as siblings directly below the `blody` of the library topic. If you are using inline elements (such as `ph`) as variable definitions then wrap each definition in a different `p` element.

***Figure 22-1 The Library of Image Variables InfoDev Template***




## Variables in a Publication

The Publication Explorer's variables tab shows a list of all the variable definitions included in the libraries associated to a publication, their values, and the variable references found in the objects of a publication. You can use this view to identify the variable references that do not have a corresponding value in a library.

The variables tab in Publication Explorer contains different columns that describe each variable in a publication:

- Name: The value of the variable's `varid` element.
- Value: The contents of the DITA element used in the variable definition.
- Resource: The name of the library or resource where the variable definition is located.
- Used: States whether a reference to the variable (a DITA element where the `varref` attribute corresponds to the variable name) is found in the publication contents.
- Message: A brief description of an error, when one is present.

The background color of each row in the table indicates the status of a variable. The possible colors, and their meaning, are the following:

- Blue: The variable is correctly defined and referenced within the publication.
- Red: The variable is referenced (used) in the publication but has not been defined in a library.
- White: A defined variable is not used in the publication.

Figure 22-2 shows the variables tab of Publication Explorer. A sample set of variables is included to demonstrate the possible values of the columns in the table and the meaning of the row colors.

***Figure 22-2 Publication Explorer's Variables Tab***

| Name | Value | Resource | Used | Message |
|---|---|---|---|---|
| CodeExample | codeblock | DITA Element Names | Yes | |
| InlineCode | codeph | DITA Element Names | No | |
| ItemList | ul | DITA Element Names | No | |
| Markup | DITA | DITA Element Names | Yes | |
| OrderedList | ol | DITA Element Names | Yes | |
| ParaTag | p | DITA Element Names | No | |
| SimpleList | sl | DITA Element Names | Yes | |
| XREFtarget | href | DITA Element Names | No | |
| BlockCode | <no value> | <missing> | Yes | This variable is not defined. |

# Creating a Variable Library

Variables are defined by DITA elements that contain a value; each variable is identified by an unique`varid` attribute. Library objects are then used to store variable definitions and associate them to a publication.

When creating a library to store variable definitions, always use the InfoDev library template that corresponds to the type of content held by its variables. Afterwards, insert all variables within the `body` of the library assigning the name and contents of each variable.

---

**Note:** Insert all variable definitions as siblings directly below the `body` of the library topic.

---

1. In the Browse Repository dialog box and under the **Repository** tab, select the folder where you want to create the new variable library.

---

**Note:** The selected folder must be of the Library content type.

---

2. Click the **New Object** button.

3. In the Select Template dialog box, under the tab for your group, select the kind of library that you want to create. For example, if you are a database writer, then select a template under the **Database Libraries** tab.
4. Click **Next**.
5. In the Add Object dialog box, fill in the metadata fields.

6. Click **OK**.

   The new library object is now listed in the right pane of the dialog box.

7. Insert the variable definitions in the new Library topic:

   a. In the Browse Repository dialog box, right-click the new Library topic and select the **Check-out** option.

   b. In Arbortext Editor, position the editor's cursor right after the opening tag of the library's body.

   c. Using the Quick Tags menu, insert an element according to the kind of variable library that are creating:

   ***Table 22-2 InfoDev Library Templates***

| Template | DITA Element | Description |
|---|---|---|
| Library of Image Variables | `image` | The contents are stored in the `href` attribute of the element. The element contents must be left empty. |
| Library of Text Variables | `ph` | |

   **Note:** If you are using inline elements (such as `ph`) as variable definitions then wrap each definition in a different `p` element.

   d. Set the variable value.

   - If you are creating an image library then the value is stored in its `href` attribute. It is set in the Insert Graphic dialog box that is automatically shown when you insert the `image` element.
   - If you are creating a textual or numerical variable then insert the contents directly between the element's opening and closing tags.

   e. With the cursor between the variable opening and closing tags, press Ctrl+D.

   f. In the Modify Attributes dialog box, under the **Other** tab, set a value for the `varid` attribute.

   **Note:** The `varid` value should be unique within the library.This is the name of the variable that you use to reference its content from any topic within the publication where the library is associated.

   g. Click **OK**.

   A value is assigned to the `varid` attribute.

8. Repeat step 7 for each variable definition that you want to insert in your library.

After following the previous procedure, your new variable library should contain a number of variable definitions, each of them resembling the one in #unique_214/unique_214_Connect_42_FIG-1421-6AEEC7BA. Notice how the `ph` element wraps the variable contents and its `varid` attribute contains the name of the variable.


**See Also:**

Metadata for Libraries
: Metadata for libraries aids in searching for libraries and tracking a library's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

## Attaching a Variable Library to a Publication

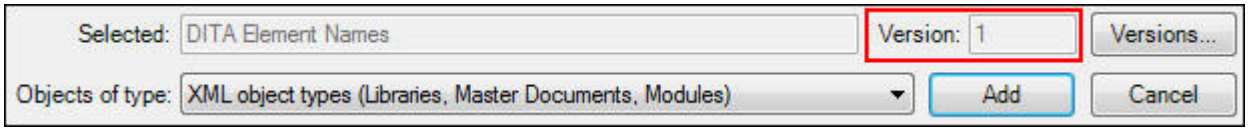
Libraries are attached to publications as resources. They determine the value that variable references resolve to when a document is published. When attaching libraries to a publication, always make sure that you are selecting the version of the object that you need.

Attach a variable library to a publication tree by doing one of the following:

- Use the context menu:

  1. In Publication Explorer , under the **Content** tab, right-click the **Resources** folder.
  2. From the context menu, select the **Add** submenu and then select the **Resource** option.
  3. In the Add Resource dialog box, browse the Repository and select the folder containing the library that you want to attach to the publication.
  4. From right pane, select the library object.

     Notice how the **Version** field indicates the version of the library object that is being selected.




  5. Select a different version of the library.

     1. Click the **Versions** button.
     2. In the Select Version dialog box, select the version from the version list.

     **Note:** Click the **Views** button to preview the contents of the selected library version.

     3. Click **OK**.

        The new version number is shown in the Version field.

  6. Click the **Add** button

- Drag and drop the library object onto the Resources folder:

  1. In Publication Manager, under the **Tools** menu, select the **Browse Repository** option.

     The Browse Repository dialog box opens.

  2. If necessary, arrange the Publication Manager window and the Browse Repository dialog box so they can be seen side by side in your display.

  3. In the Browse Repository dialog box, browse the repository hierarchy and select the folder containing the library that you want to attach to the publication.

  4. In the right pane, locate the library object that you want to attach to your publication and drag-and-drop it onto the Resources folder in the publication tree.

     **Note:** If necessary, follow the procedure described in 5 to select a different library version. Verify that the right version number appears in the version field before drag and dropping the object.

The library is now attached to your publication. It displays in the publication tree view as a child of the Resources folder.

## Modifying a Variable Definition

You can modify a variable definition in two ways: by changing it's name ( its `varid` attribute) or the variable value.

To modify a variable definition, you locate the corresponding library in the repository, check it out, perform the necessary changes, and save them by checking the library object back in the repository.

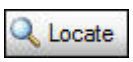
**Important:** Take into account that modifying a variable definition will impact the way all topics in the repository (not only a single publication) that refer to the variable resolve. Work together with the appropriate content team to verify the implications of your changes.

1. Do one of the following to locate the library object where the variable definition resides:

   - Complete the next steps if you are currently working in a publication and don't know where the variable is defined:

     1. In Publication Explorer, under the **Variables** tab, select the variable that you want to modify.

     2. Click the **Locate in Publication** button.

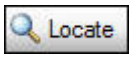
        The library that contains the variable definition is selected within the Content tab of Publication Explorer.

- If you already know the location of the library in the repository then use the Browse Repository dialog box to find the library object and select it.

2. Right-click the library and select the **Check Out** option from the context menu.
3. In Arbortext editor, locate the variable definition:
   a. From the **Find** menu, select the **Find Tag/Attribute** option.
   b. Enter the variable's name (the `varid` attribute) in the **Attribute Value** field.
   c. Click **Find Next**.
4. Optional: To change the name of the variable:
   a. Select the element whose `varid` attribute matches the variable's name.
   b. Press Ctrl+D.
   c. In the Modify Attributes dialog box, select the **Other** tab and change the value of the `varid` field.

   **Note:** The `varid` attribute value should be unique within the library.

   d. Click **OK**.
5. To change the value of the variable:
   - Replace the contents of the DITA element serving as the variable.
   - If you are modifying an image variable then do the following:
     1. Position the editor cursor between the `image` tags.
     2. Press Ctrl+D.
     3. In the Modify Attributes dialog box, under the Reference tab, click the **[...]** button next to the href field.
     4. In the Insert Graphic dialog box, browse the repository folder hierarchy and select the new graphic object for you image variable.
     5. Click **Insert**.

     
     
6. Check in the library.

You can see the variable changes in the **Variables** tab of Publication Explorer.

**See Also:**

The Browse Repository Dialog Box
: The Browse Repository dialog box provides a graphical user interface to the repository. It enables you to complete several tasks, such as creating folders and objects, previewing objects, and checking objects out.

## Deleting a Variable Definition from a Library

Delete a variable definition by removing the corresponding DITA element from the library object that contains it.

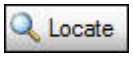
Removing a variable definition that is still used in any topic will lead to producing incomplete documents from publications that include those topics. Therefore, delete every reference from the topics in the repository before removing the variable definition.

> **Note:** You might not be able to modify some topics given their current workflow state. Work together with the appropriate content team to verify the implications of removing a variable definition.

1. In Publication Manager, open a publication that includes the variable as a resource.
2. In Publication Explorer and under the **Variables** tab, select the variable that you want to delete.
3. Optional: Run a Where Used report on the variable to locate all topics in the repository that include a reference to it.

   > **Note:** Make sure to select the **Entire Repository** option as scope of the report.

4. Delete each variable reference from the topics included in the report.
   a. Right-click an object in the report and select the **Check Out** option.
   b. In Arbortext editor, locate and delete the variable definition:
      1. From the **Find** menu, select the **Find Tag/Attribute** option.
      2. Enter the variable's name (the `varref` attribute value) in the **Attribute Value** field.
      3. Click **Find Next**.
   c. Delete the DITA element that serves as the variable reference.
   d. Check in the current topic
   e. Repeat steps 4.a to 4.d for each object in the Where Used report.
   f. Close the Where Used report.
5. With the variable to be deleted selected in the Variables tab, click the **Locate in Publication** button.




The library that contains the variable is selected in the Content tab of Publication Explorer.

6. Right-click the library and select the **Check Out** option from the context menu.
7. Using the **Find Tag/Attribute** option again, locate and delete the variable definition in the library object.

   **Note:** Search by the variable name (its `varid` attribute).

8. Check in the library.

The variable is now deleted from your from the library.

**See Also:**

Locating Variable References in the Repository
Use the Where Used report to find topics that reference a variable, this is useful to evaluate the impact of modifying or deleting a variable definition and make the necessary changes. The scope of this action can be at publication level or across the entire repository.

## Locating Variable References in the Repository

Use the Where Used report to find topics that reference a variable, this is useful to evaluate the impact of modifying or deleting a variable definition and make the necessary changes. The scope of this action can be at publication level or across the entire repository.

1. In Publication Manager, open a publication that includes the variable as a resource.
2. In Publication Explorer and under the **Variables** tab, select the variable whose references you want to locate in the publication.
3. From the **View** menu, select the **Where Used** option.

   The Where Used report appears in a pane at the right side of Publication Explorer.
4. In the drop-down list, at the top-right corner of the report, select one of the following values:
   - **Current Publication**
   - **Entire Repository**

The Where Used report lists all the objects that include a reference to the selected variable.

## Viewing the Variable Values for a Publication

For each publication, you can view **variables explorer** that lists both variables found in the publication's contents and variables found in the variable libraries (resources)

attached to the publication. You can use the list to identify any variable references that do not have a corresponding value in a resource.

- In Publication Explorer, click the **Variables** tab.

  The publication's list of variables is shown, with the following columns:

  - Name —the variable identifier (`varid` attribute).
  - Value—the variable value as specified in an attached library.
  - Resource—the name of the library that contains the variable.
  - Used—"Yes" if the variable is referenced by any topics in the publication; "No", otherwise.
  - Message—an error message associated with the variable, if any.

**See Also:**

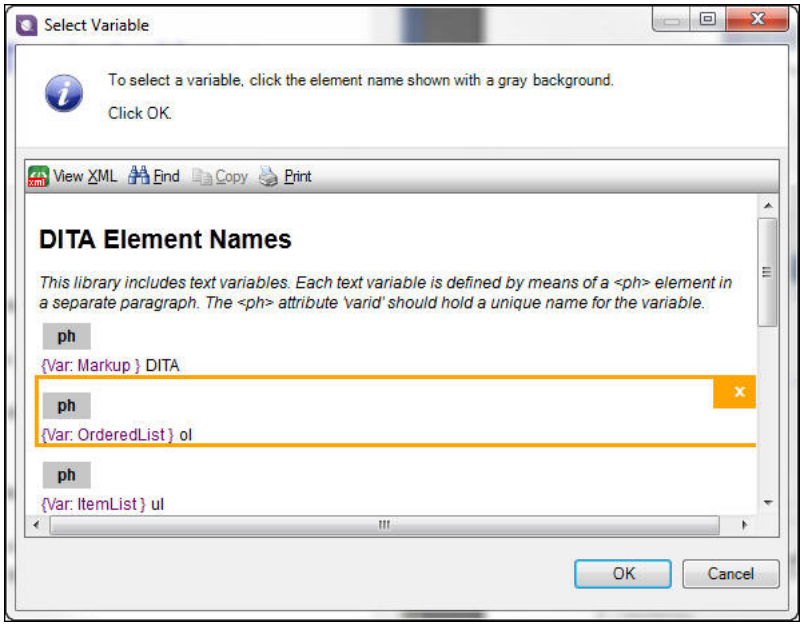
Variables in a Publication
: The Publication Explorer's variables tab shows a list of all the variable definitions included in the libraries associated to a publication, their values, and the variable references found in the objects of a publication. You can use this view to identify the variable references that do not have a corresponding value in a library.

## Inserting a Variable Reference in a Topic

Use variables in a topic by inserting variable references. The SDL Authoring Bridge enables you to reference variables without dealing with the XML complexities.

1. In Edit View, position the editor cursor where you want to insert a variable reference.
2. From the **SDLLiveContent** menu, select **Insert Variable**.
3. Do one of the following to select the library where the variable resides:
   - If you checked the topic out directly from the publication tree in Publication Manager then select the library from the Variable Resources dialog box.
   - If you checked the topic out from the Browse Repository dialog box then browse the repository hierarchy in the Select dialog box and select the library object.
4. Click **Select**.
5. In the **Select Variable** dialog box, click the gray box that precedes the variable that you want to reference.

   An orange box indicates your selection.




6. Click **OK**.
7. Click **Insert**.

The variable reference is now inserted into the topic.

# 23

# Conditionalizing Content Using Conditions

Conditions enable you to mark a DITA element, topic, or submap as conditionalized. Conditionalized content is either included or excluded from a publication, depending on publication settings.

## About Using Conditions in SDL Live Content Architect

You **conditionalize** DITA elements, topics, or submaps by marking them with **condition expressions** that are stored in the `ishcondition` attribute.

Conditions have names and values. If the value of the condition is different from the value called for by the publication, then the conditionalized element, topic, or submap is excluded from the output.

A typical condition attribute looks like this:

```
ishcondition="Cloud=Y"
```

This condition applied to a paragraph looks like this:

```
<p ishcondition="Cloud=Y">You start by logging in to your Cloud
account.</p>
```

A typical use for conditionalizing elements within a topic is to select between two variations of an element. For example, in the APEX User's Guide, if you wanted to include a paragraph that is different for the on-premise version and the cloud version of the publication, you would add both variations of the paragraph to the topic, one right after the other, and conditionalize each. This would look like this:

```
<p ishcondition="Cloud=Y">You start by logging in to your Cloud
account.</p>

<p ishcondition="Cloud=N">You start by launching the Application
Builder.</p>
```

### Conditionalizing Levels

You can conditionalize content at three different levels: at the submap, topic, or element level. The main difference is where the condition is applied. To conditionalize an entire submap or entire topic, you add a condition to the topic reference (`topicref` element) in the DITA map that includes the submap or topic in the publication. To conditionalize content inside a topic, you add a condition to the DITA element that wraps the content.

## Condition Expressions

The simplest condition expression is *condition_name=value*. You can logically combine a simple expression with another expression to create a **complex condition expression**.

For example, you use the following simple condition expression to include an element only in the variation of a document for the Oracle Linux 5 operating system (OS).

```
OS=OL5
```

In SDL LiveContent Architect, condition expressions are stored in the Architect-defined `ishcondition` attribute of a DITA element or `topicref`. Using this special attribute enhances the conditional processing capabilities of DITA by enabling you to assign data types to conditions (text, number, date, and version). You use the **SDLLiveContent** menu in Arbortext to assign the expression to an element that you want to conditionalize. For example:

```
<p ishcondition="OS=OL5"></p>
```

> **Note:** "ish" stands for "InfoShare" which used to be the product name of SDL LiveContent Architect.

Operators available for conditions of type number include `>=` and `<`.

### Complex Condition Expressions

A **complex condition expression** is the logical combination of two or more condition expressions. For example, you could create the following complex condition expression to include an image only in the client version of a product running on Windows Server 2008:

```
<image ishcondition="(OS=Windows Server 2008) and
(Install_Type=Client)"></image>
```

Available logical operators are `and`, `or`, and `in`.

The `in` operator is a convenience operator for long `or` expressions. The following two expressions are equivalent:

```
Install_type in (Grid,RAC,SI)
Install_type=Grid or Install_type=RAC or Install_type=SI
```

## The Condition Builder Dialog Box

SDL LiveContent Architect uses its graphical interface to hide the underlying XML complexities of assigning condition attributes to elements. The Condition Builder dialog box enables you to easily select a condition expression and apply it to content. You can also use Condition Builder to create complex condition expressions.

The Condition Builder dialog box consists of two tabs:

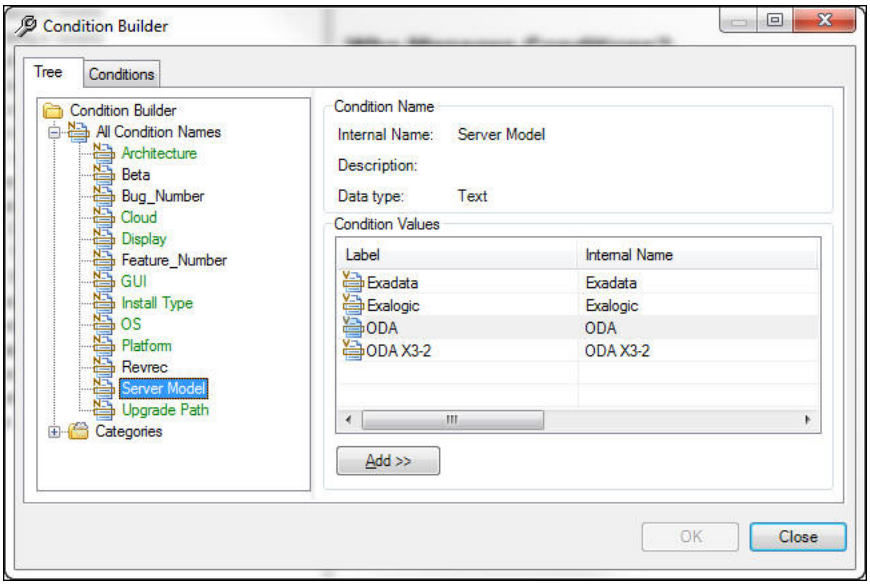
- **Tree tab**: Contains the condition tree, a hierarchical structure that organizes all the conditions you can use to conditionalize content, and a pane where you select

values for the selected condition. This tab is used to select a condition and values and add a condition expression to the Conditions tab.

- **Conditions tab**: Lists condition expressions ready to by applied to elements. This is where complex condition expressions are created.

A typical way of conditionalizing content consists of selecting the condition name and one or more values from the Tree tab, adding the resulting expression to the list of expressions in the Conditions tab, and applying the selected expression to an element from the Conditions tab. You can also apply a condition expression that was previously added to the Conditions tab. You can also create a complex condition expression (using the **OR** or **AND** buttons in the Conditions tab) and then apply it to content.

***Figure 23-1 The Condition Builder Dialog Box***

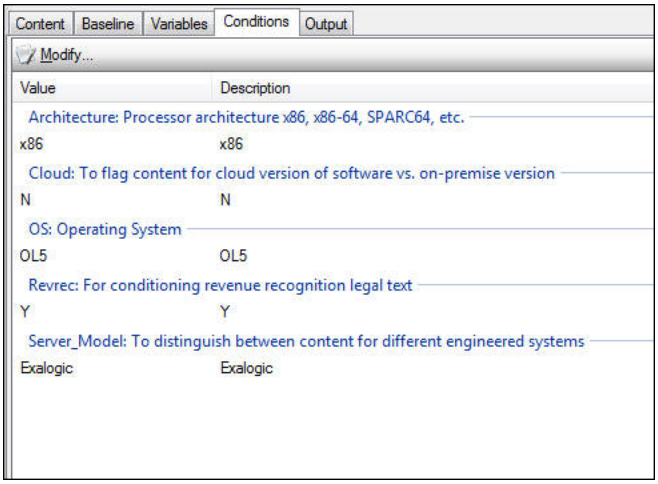



## About the Condition Context of a Publication

In SDL LiveContent Architect, condition values are a key component of a publication's **context**. By specifying condition values, you customize the document generated from a publication.

The Conditions tab in Publication Explorer lists the conditions that are used to include or exclude conditionalized content when the publication is published. You use the **Modify** button in this tab to add, modify, and remove conditions.

*Figure 23-2 The Conditions Tab in Publication Explorer*




Clicking the **Modify** button also enables you to choose different options for displaying conditions:

- Display all conditions
- Display conditions used in a publication
- Display unused conditions
- Display selected conditions

This feature enables you, for example, to double-check if the conditions that you use to define the publication context are actually used in any object within the publication or if you forgot to select a condition value that you were planning to use.

### Who Manages Conditions?

The set of allowed conditions and conditions values that are used in the InfoDev documentation is designed and maintained by the database architects committee. The purpose of this committee is to keep control and standardize the use of conditions across of the InfoDev organization.

Writers can request the creation of new conditions and values when there is a valid reason to justify the enhancement of the current set of conditions. Send a e-mail message to `arborhelp_ww_grp@oracle.com` to ask for a new condition that fits your needs better.

## Conditionalizing an Entire Topic

Conditionalize a topic object by marking the `topicref` element that references the topic (within the corresponding ditamap) with a condition expression. Perform this process from Publication Explorer.

1. In the Content tab of Publication Explorer, right click the topic that you want to conditionalize.
2. From the context menu, select the **XML Attributes** option.

   The Insert Values for 'Topic Ref' dialog box appears. It shows fields used to set values to the attributes of the selected `topicref` element.

> **Note:** Notice how the map or submap that contains the topic is automatically checked out (a red lock is shown to the left of the map icon).

3. Click the **[...]** button next to the Condition field.

   The Condition Builder dialog box appears. It shows the list of condition expressions.

4. Add a new condition expression to the list of expressions in the Conditions tab.

   a. In the **Tree** tab, expand **All Condition Names** and select the name of condition that you want to use in the expression.

   b. In the Condition Values pane, select the value or values of the condition that you want to use in the expression.

   > **Note:** You can select multiple values to create an IN expression, such as `Beta in (internal, external)`. This example is equivalent to `Beta=internal or Beta=external`.

   c. Click **Add**.

      The new condition expression is added to the list of expressions in the Conditions tab.

   d. Repeat steps#unique_227/unique_227_Connect_42_INTHETREETABEXPANDTHECONDITIONTREEA-6E8FDA58 through#unique_227/unique_227_Connect_42_CLICKADD.THENEWCONDITIONVALUEISADDE-6E8FDD67 to add new condition values.

5. In the **Conditions** tab, select the condition expression that you want to add to the element.

   > **Note:** You can create complex condition expressions by joining expressions (or other complex expressions) with a logical operator

6. Click **OK**.

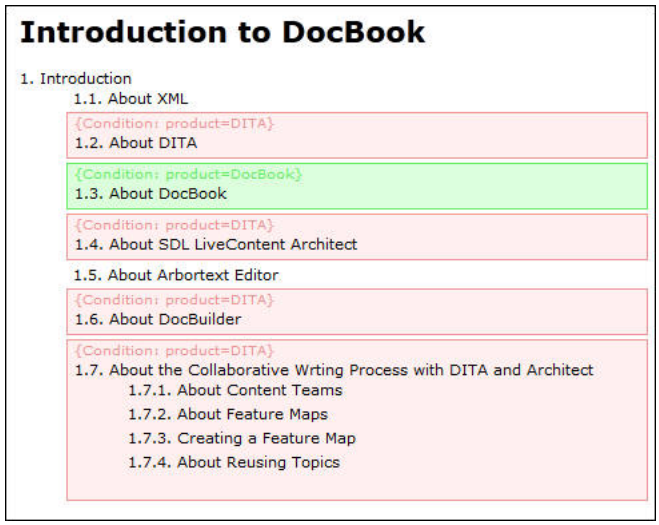
   The Condition field in the Insert Values for 'Topic Ref" dialog box is automatically set.

7. Click **OK**.

8. Right-click the map or submap that includes the topic and select the **Check In** option.

The topic is now conditionalized. If you preview the map that includes the topic then you can see the result of applying the new condition. Figure 23-3 shows a sample submap where several topics have been conditionalized. Those `topicref` elements inside a green box are included in the publication output; those inside a pink box are excluded.

*Figure 23-3 Preview of a Submap With Several Conditionalized Topics*




**See Also:**

Creating Complex Condition Expressions

A complex condition expression is a combination of two or more expressions using logical operators. Creating complex condition expressions enables you to address very specific needs when filtering or conditionalizing content in a publication.

# Conditionalizing Content Within a Topic

Conditionalize chunks of content included inside a topic by marking the DITA element that contains it with a condition expression. Perform this process from Arbortext Editor.

1. In the Arbortext Edit view, select the element that you want to conditionalize.
2. From the **SDLLiveContent** menu, select **Conditional Text** and then **Set Condition**.

   The Condition Builder dialog box appears.
3. Add a new condition expression to the list of expressions in the Conditions tab.
   a. In the **Tree** tab, expand **All Condition Names** and select the name of condition that you want to use in the expression.
   b. In the Condition Values pane, select the value or values of the condition that you want to use in the expression.

   **Note:** You can select multiple values to create an IN expression, such as `Beta in (internal, external)`. This example is equivalent to `Beta=internal or Beta=external`.

   c. Click **Add**.

      The new condition expression is added to the list of expressions in the Conditions tab.

d. Repeat steps#unique_227/unique_227_Connect_42_INTHETREETABEXPANDTHECONDITIONTREEA-6E8FDA58 through#unique_227/unique_227_Connect_42_CLICKADD.THENEWCONDITIONVALUEISADDE-6E8FDD67 to add new condition values.

4. In the **Conditions** tab, select the condition expression that you want to add to the element.

> **Note:** You can create complex condition expressions by joining expressions (or other complex expressions) with a logical operator

5. Click **OK**.

The element is now conditionalized, as shown by the new `ishcondition` attribute of the element.

Figure 23-4 `phstepCloud`

***Figure 23-4 Conditionalized Instructions Within a Step***


**See Also:**

Creating Complex Condition Expressions
: A complex condition expression is a combination of two or more expressions using logical operators. Creating complex condition expressions enables you to address very specific needs when filtering or conditionalizing content in a publication.

## Creating Complex Condition Expressions

A complex condition expression is a combination of two or more expressions using logical operators. Creating complex condition expressions enables you to address very specific needs when filtering or conditionalizing content in a publication.

1. In the Condition Builder dialog box, ensure that all the condition expressions to include in the complex expression appear in the Expressions list under the Conditions tab. If one of the desired expressions is not there, complete the following steps to create it:

   a. Under the **Tree** tab, select the condition name from the conditions tree.

   b. Select the value that you want from the **Condition Values** pane.

   c. Click the **Add** button.

2. Under the **Conditions** tab, select the first condition that you want to include in the complex condition expression.

3. Press and hold the Ctrl key and select the second expression to include in your complex condition expression.

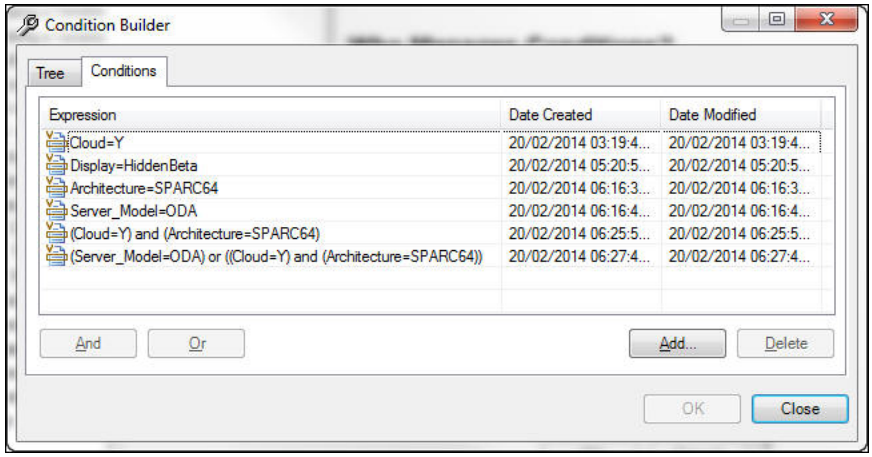
4. Click the **And** or **Or** button.


The new complex condition expression is added to the list of expression.

5. Repeat steps 2 through 4 to add another condition expression to the complex expression.

Figure 23-5 `Server_Model=ODA`

***Figure 23-5 Composition of Complex Condition Expressions***




# Including or Excluding Conditions in a Publication

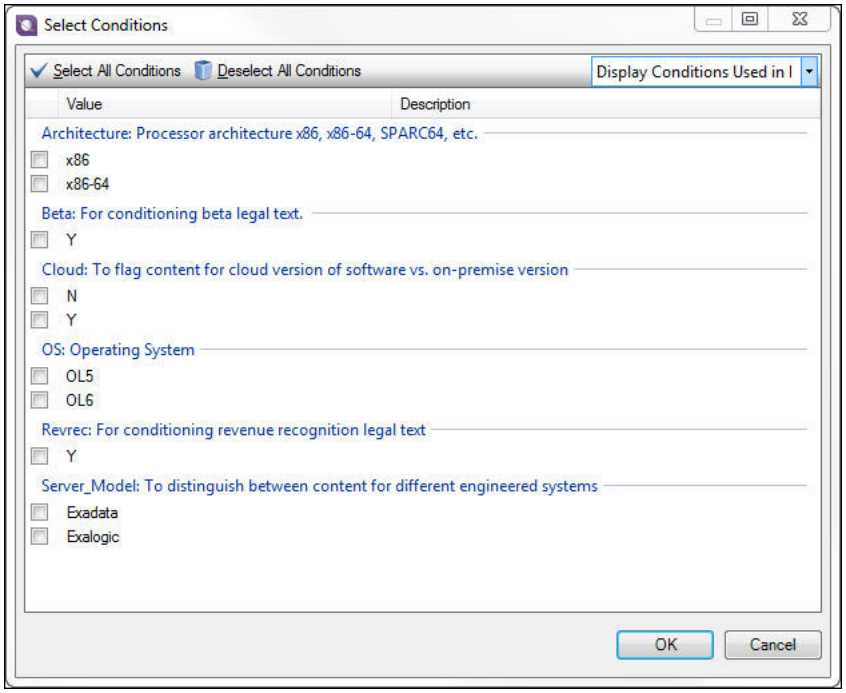
The Content's tab in Publication Explorer shows a list of the condition expressions that define the context of a publication. Use the Select Conditions dialog box to choose which conditions to include or exclude from the output of a publication.

1. In Publication Explorer click the Conditions tab.

   The Conditions tab displays a list of the conditions, and their values, that are used in the contents of the current publication.

2. Click the **Modify** button.

   The Select Conditions dialog box appears. By default, it lists all the conditions values that are used to conditionalize content that is a part of the publication.




**Note:** You can change the filter used to list conditions by selecting a different option from the drop-down list at the top-right corner of the dialog box.

3. In the condition list, select all the condition values that you want to include in the output of the publication.

**Note:** To make sure that you include only the correct elements, you can start by clicking the **Deselect All Conditions** button and then individually selecting each condition to include.

4. Click the **OK** button.

The list of conditions in the Conditions tab is updated according to your new selection.

## Removing a Condition

If an element no longer needs to be conditionalized then you can remove its condition.

Removing conditions involves a different procedure depending on where the condition resides. Use Publication Explorer when removing conditions from `topicref` elements pointing to maps and topics in a publication; DITA elements within topics are removed using the Arbortext Editor interface.

Do one of the following to remove a condition:

- If you want to remove a condition from a piece of content inside a topic:

  1. Check out the topic containing the condition that you want to delete.

  2. In Arbortext Editor, select the conditionalized element.

3. From the **SDLLiveContent** menu, select **Conditional Text**, then **Remove Condition**.

   The element's `ishcondition` attribute is removed.

- If you want to remove a condition from a `topicref` (pointing to either a topic or a submap):

  1. In Publication Explorer, right-click the `topicref` whose condition you want to remove.
  2. In the context menu, select the **XML Attributes** option.
  3. In the Insert Values for Topic Ref dialog box, clear the Condition field.
  4. Click **OK**.

     The condition is removed from the topicref.

# InfoDev Conditions Reference

When writing InfoDev documentation use only the approved conditions and conditions values to create condition expressions. The InfoDev architects committee is in charge of maintaining the set of conditions, if you have a valid reason to request the creation of a new condition then write an e-mail to `arborhelp_ww_grp@oracle.com`.

***Table 23-1 Condition Names and Values Allowed in InfoDev Documentation***

| Condition Name | Condition Values | Description |
|---|---|---|
| Architecture | • SPARC<br>• SPARC64<br>• x86<br>• x86–64 | Processor architecture (x86, x86-64, SPARC64, etc.) |
| Beta | • Y | For conditioning beta legal text |
| Bug Num | String | Not used |
| Cloud | • N<br>• Y | To flag content for cloud version of software vs. on-premise version |
| Feature Num | String | Not used |
| GUI | • CLI<br>• EMCC<br>• EMGC<br>• EMX | To distinguish between command line and Enterprise Manager tasks. (CC=Cloud Control, GC=Grid Control, X=EM Express) |
| Install Type | • Client<br>• Grid<br>• RAC<br>• SI | For install/upgrade tasks that differ for Client, grid, RAC, single instance |
| OS | • Asianux3<br>• Asianux4<br>• OL5<br>• OL6 | Operating System |

| Condition Name | Condition Values | Description |
|---|---|---|
| | • RHEL4<br>• RHEL5<br>• RHEL6<br>• RHEL7<br>• Solaris 10<br>• Solaris 11<br>• Solaris 12<br>• Solaris 13<br>• Solaris 9<br>• SUSE10<br>• SUSE11<br>• SUSE12<br>• Windows 7<br>• Windows Server 2003<br>• Windows Server 2003 SP2<br>• Windows Server 2008<br>• Windows Server 2008 R2<br>• Windows Server 2012<br>• Windows Server 2012 R2 | |
| Platform | • AIX<br>• HP-UX<br>• Linux<br>• Solaris<br>• UNIX<br>• Windows | Platform: Linux, UNIX, Solaris, Windows, etc. |
| Revrec | • Y | For conditioning revenue recognition legal text |
| Server Model | • Exadata<br>• Exalogic<br>• ODA<br>• ODA X3–2 | To distinguish between content for different engineered systems |
| Upgrade Path | • 102_111<br>• 102_112<br>• 102_121<br>• 102_122<br>• 111_112<br>• 111_121<br>• 111_122<br>• 112_121<br>• 112_122<br>• 121_122 | For the Upgrade Guide |

Simple conditions expression follow the form `ConditionName=Value`, they are stored in every element's `ishattribute`. You can also create complex condition expression (logical combinations of simple condition expressions from the Condition Builder dialog box and then adding them to an element.

The following example shows a section that is conditionalized so it is included in the Linux version of a document in a cloud environment:

```
<section ishcondition="(Platform=Linux) AND (Cloud=Y)"></section>
```

# 24

# Reusing Small Chunks of Content with Conrefs

A content reference (conref) is a reference to a reusable element in a library or topic. Conrefs enable writers to share small chunks of content in multiple topics.

You can create libraries of elements and reference the elements in these libraries from topics using conrefs. You can also reference elements in regular topics as conrefs.

**See Also:**

About Content Teams
: The content team works together to determine the topics that must be written to document a specific product feature.

## About Conrefs

A content reference (conref) is a reference to a reusable element in a library or in a topic. Multiple topics can reference a single element. Conrefs provide a way to reuse content at a more granular level than a topic.

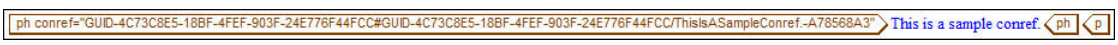
A conref library is a collection of elements that can be referenced individually in topics. A conref can also reference an element in a different topic. Regardless of whether the element is in a library or a topic, the element being referenced must be assigned an ID. This ID can be a GUID or an ID generated by Arbortext Editor.

Conrefs provide the following benefits:

- **Availability:** Conref elements can be stored in libraries and can be referenced by multiple topics and publications. Reusable content is available to different consumers.
- **Consistency:** A conref element contains the same content in every topic that references it.
- **Easy maintenance:** When a conref element is changed, all topics that reference the element reflect the change.

When a conref is inserted in a topic, the element referenced in the conref is read-only in the topic. To modify the content of a conref, you must modify the element referenced by the conref in a library or in the topic that contains the source element. When the source element changes, all of the conrefs that reference the element reflect the change.

***Example 24-1 Sample Conref***



Notice that the element is referenced using GUIDs and an internal ID generated by Arbortext Editor. In this case, the IDs identify the library and element that is referenced.

You can view a conref library in the repository or in Arbortext Editor in the same way that you view other objects.

---

**Note:** When you preview a topic in the repository, conrefs are not visible. A gray box appears for each conref, but it does not show the content of the conref. Conrefs are visible when you view a topic in Arbortext Editor or in Publication Manager.

---

**See Also:**

Types of Conref Libraries
: A conref library is a collection of elements that can be referenced individually in topics. Several types of conref libraries are available.

Conrefs in a Publication
: When a publication is open, the Conrefs in publication pane in Publication Explorer shows all of the content references (conrefs) in a publication. You can use this pane to locate and validate each conref.

Creating a Conref Library
: You can organize DITA elements that are conref targets into an Architect library. You can have any number of conref libraries. Unlike variable libraries, you do not need to add conref libraries to the resources folder of your publication.

Inserting a Conref in a Topic
: A content reference (conref) is a reference to a reusable element. You insert a conref that points to an element in a library or another topic.

Validating a Conref
: Validate a content reference (conref) to verify that the conref definition exists and is available to the topic that references it.

Modifying the Element Referenced in a Conref
: You modify the element referenced by a content reference (conref) in the library or topic that contains the element.

Deleting a Conref
: Deleting a content reference (conref) removes it from a topic, but the referenced element is not modified or deleted. All other topics referencing it remain the same.

# Conrefs in a Publication

When a publication is open, the Conrefs in publication pane in Publication Explorer shows all of the content references (conrefs) in a publication. You can use this pane to locate and validate each conref.

The Conrefs in publication pane in contains different columns that describe each conref in a publication:

- Name: The ID that identifies the element that is referenced.
- Resource: The resource that contains the element that is referenced.

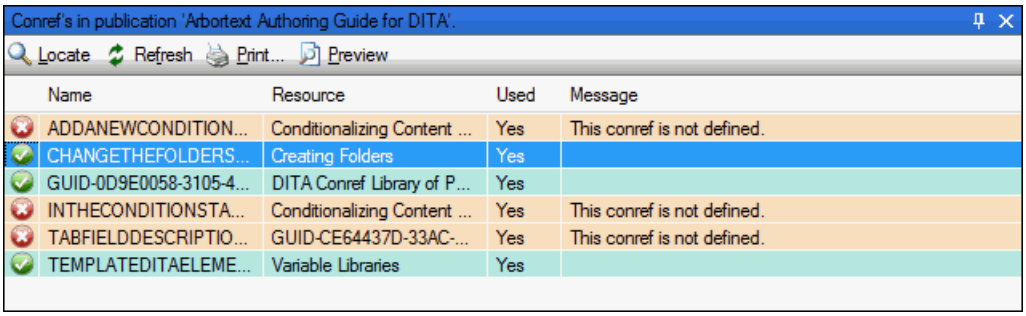
- Used: Yes if the element is referenced as a conref. No if the element is not referenced as a conref.
- Message: A brief description of an error, when one is present.

The background color of each row in the table indicates the status of a conref. The possible colors, and their meaning, are the following:

- Blue: The conref is correctly defined and referenced within the publication.
- Red: The conref is referenced (used) in the publication but cannot be validated.
- White: A defined conref element is not used in the publication.

The following figure shows the Conrefs in publication pane of Publication Explorer. A sample set of conrefs is included to demonstrate the possible values of the columns in the table and the meaning of the row colors.

***Figure 24-1 Conrefs in publication Pane***




**See Also:**

About Conrefs
: A content reference (conref) is a reference to a reusable element in a library or in a topic. Multiple topics can reference a single element. Conrefs provide a way to reuse content at a more granular level than a topic.

Validating a Conref
: Validate a content reference (conref) to verify that the conref definition exists and is available to the topic that references it.

# Types of Conref Libraries

A conref library is a collection of elements that can be referenced individually in topics. Several types of conref libraries are available.

| Library Type | Description |
|---|---|
| Book links | Contains a collection of `xref` elements enclosed in `p` elements. Each `p` element can be referenced in reltables and topics to create a link to a book. |
| Feature descriptions | Contains a collection of `p` elements. Each `p` element can be referenced in topics for consistent feature descriptions. |
| Notes | Contains a collection of `note` elements. Each `note` element can be referenced in topics for consistent notes. |
| Paragraphs | Contains a collection of `p` elements. Each `p` element can be referenced in topics for a consistent chunk of reusable content. |

| Library Type | Description |
|---|---|
| | **Note:** You can include any element that is valid within a `p` element, including ordered lists, unordered lists, tables, code blocks, and so on. |
| Phrases | Contains a collection of `ph` elements enclosed in `p` elements. Each `p` element can be referenced in topics for consistent phrases. |
| Product version labels | Contains a collection of `ph` or `keyword` elements. Each `ph` or `keyword` element can be referenced in topics for consistent product and release text. |
| Steps | Contains a collection of `step` elements. Each `step` element can be referenced in topics for consistency when documenting a specific action. |

**See Also:**

About Conrefs
: A content reference (conref) is a reference to a reusable element in a library or in a topic. Multiple topics can reference a single element. Conrefs provide a way to reuse content at a more granular level than a topic.

## Creating a Conref Library

You can organize DITA elements that are conref targets into an Architect library. You can have any number of conref libraries. Unlike variable libraries, you do not need to add conref libraries to the resources folder of your publication.

Although there are only a handful of templates for creating conref libraries, where each template is named for an element type (such as paragraph and phrase), you can create a conref library of almost any DITA element type.

**Note:** All elements in a conref library must be the same type. You must generate an ID for each element. The ID enables conrefs in topics to reference the elements.

1. In Publication Manager, from the **Tools** menu, select **Browse Repository**.
2. In the **Repository** tab, locate and select the folder that will contain the new library. If the desired folder does not sexist, create it and then select it.

   **Note:** The content type of the folder must be "Library." To check the content type of a folder, right-click the folder and select **Properties**. The content type is on the **General** tab.
3. Click the **New Object** button.

4. In the Select Template dialog box, under the tab for your product group, select one of the conref library templates. For example, if you are a database writer, then select a template under the **Database Libraries** tab

   The following conref library templates are available (conref library template titles do not end with the word "Variables"):

   - Library of Book Links
   - Library of Feature Descriptions
   - Library of Notes
   - Library of Paragraphs
   - Library of Phrases
   - Library of Product Version Labels
   - Library of Steps

   **Note:** If no template matches the kind of element you want to have in your library, select the template that is closest to that type of element. For example, if you want to create a library of inline (phrase) elements such as `<uicontrol>`, then select the Library of Phrases template.

5. Click **Next**.
6. In the Add Object dialog box, enter a title for the library in the **Title** field within the **General** tab, and enter other metadata if it is appropriate for the new conref library.
7. Click **OK** .
8. Check out the new library.
9. In Arbortext Editor, add an element to the library:
   a. Add an element of the correct type. For example, in a Library of Notes, each element must be a `note`.

      **Note:** Each template has instructions for inserting the correct types of elements. However, you can use different elements where appropriate if no template includes the type of element you want to use.

   b. Place your cursor in the element.
   c. Click the **Generate ID** button.

      **Note:** The **Generate ID** button generates an Arbortext Editor ID for the element so that it can be referenced in conrefs.

10. Repeat the previous step to add more elements.

> **Note:** Ensure that you use the **Generate ID** button to generate an ID for each element.

11. Check in the library.

**See Also:**

Metadata for Libraries
: Metadata for libraries aids in searching for libraries and tracking a library's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

Inserting a Conref in a Topic
: A content reference (conref) is a reference to a reusable element. You insert a conref that points to an element in a library or another topic.

About Conrefs
: A content reference (conref) is a reference to a reusable element in a library or in a topic. Multiple topics can reference a single element. Conrefs provide a way to reuse content at a more granular level than a topic.

# Inserting a Conref in a Topic

A content reference (conref) is a reference to a reusable element. You insert a conref that points to an element in a library or another topic.

A conref is read-only after it is inserted in a topic. To modify a conref, you must change the source element in the library or topic that contains it. When a source element is changed, all conrefs that reference the source element reflect the change.

1. Check out the topic in which you want to insert the conref.
2. In Arbortext Editor, place your cursor in the topic where the conref should be inserted.

> **Note:** A conref is a substitute for an element. Therefore, you can insert a conref to an element only where the element type is valid. For example, if you want to insert a conref to a step element, then you can insert it only where a step element is valid.

3. From the **SDL LiveContent** menu, select **Conref** > **Insert Conref**.
4. In the Insert Conref dialog box, locate and select the library or topic that contains the element you want to reference.
5. Click the **Select** button.
6. In the Select dialog box, select the element you want to reference.

> **Note:** An orange box surrounds the selected element.

7. Click the **OK** button.

8. In the Insert Conref dialog box, click the **Insert** button.
9. Check in the topic to save your changes.

**See Also:**

Creating a Conref Library
: You can organize DITA elements that are conref targets into an Architect library. You can have any number of conref libraries. Unlike variable libraries, you do not need to add conref libraries to the resources folder of your publication.

About Conrefs
: A content reference (conref) is a reference to a reusable element in a library or in a topic. Multiple topics can reference a single element. Conrefs provide a way to reuse content at a more granular level than a topic.

## Validating a Conref

Validate a content reference (conref) to verify that the conref definition exists and is available to the topic that references it.

- To validate a conref in Arbortext Editor:
  1. Check out the topic that contains the conref you want to validate.
  2. Place the cursor inside the conref.
  3. From the **SDL LiveContent** menu, select **Conref** > **View Source Object**.

     The library or topic that contains the conref opens in Arbortext Editor with the cursor on the referenced element. The library or topic is read-only if it is not checked out.
- To validate a conref in Publication Manager:
  1. Open the publication that includes the topic that contains the conref.
  2. From the **View** menu, select **Conref**.

     The **Conref** pane opens at the bottom and lists the conrefs in the publication.
  3. Select the conref you want to validate from the list.
  4. Click **Preview** at the top of the conref pane.

     The Preview of Conref dialog box displays the conref. The library or topic is read-only if it is not checked out.

If the conref cannot be validated, then ensure that the conref exists and that the library/topic that contains it is in the repository.

**See Also:**

About Conrefs
: A content reference (conref) is a reference to a reusable element in a library or in a topic. Multiple topics can reference a single element.

Conrefs provide a way to reuse content at a more granular level than a topic.

Conrefs in a Publication
: When a publication is open, the Conrefs in publication pane in Publication Explorer shows all of the content references (conrefs) in a publication. You can use this pane to locate and validate each conref.

# Modifying the Element Referenced in a Conref

You modify the element referenced by a content reference (conref) in the library or topic that contains the element.

When you modify an element in a conref library or in a topic, all topics that reference the element reflect the change.

**Note:** You can validate a conref to determine the library or topic that contains it.

1. Check out the conref library or the topic that contains the element.
2. In Arbortext Editor, modify the contents of the element being referenced.
3. Check in the conref library or topic.

**See Also:**

About Conrefs
: A content reference (conref) is a reference to a reusable element in a library or in a topic. Multiple topics can reference a single element. Conrefs provide a way to reuse content at a more granular level than a topic.

Validating a Conref
: Validate a content reference (conref) to verify that the conref definition exists and is available to the topic that references it.

# Deleting a Conref

Deleting a content reference (conref) removes it from a topic, but the referenced element is not modified or deleted. All other topics referencing it remain the same.

1. Check out the topic from which you want to delete the conref.
2. In Arbortext Editor, select the conref, including the opening tag, the referenced element, and the closing tag.
3. From the **Edit** menu, select **Delete**.
4. Check in the topic to save your changes.

**See Also:**

About Conrefs
: A content reference (conref) is a reference to a reusable element in a library or in a topic. Multiple topics can reference a single element.

Conrefs provide a way to reuse content at a more granular level than a topic.

# Part V

## Reference

# 25

# About InfoDev DITA Standards

InfoDev uses a specific set of information types, and writers use templates designed to create specific types of topics. Writers must have the skills necessary to write these topics, such as minimalist writing skills and topic-based writing skills.

## InfoDev DITA Information Types

InfoDev DITA information types, also called topic types, are the building blocks for documentation. The types are task, concept, reference, and orientation.

***Table 25-1 Information Types***

| Information Type | Answers the Questions . . . | Description | Tag |
|---|---|---|---|
| Task | How do I? | Provides a step-by-step procedure to complete a single task. Often a task includes an example. | <task> |
| Concept | What is it? Why is it important? How does it work? | Describes a feature or component (including its key terms), states its purpose or benefits, and explains how it works. A concept is analogous to an encyclopedia entry. | <concept> |
| Reference | What is its definition? What are the requirements or guidelines? What are the valid values? | Describes either the components of a user interface, an API, or a list of feature requirements or guidelines. Unlike concepts, reference topics do not explain in detail how something works. Unlike tasks, reference topics do not describe specific procedures. A reference topic is analogous to a dictionary. | <reference> |
| Orientation | What do I need to know about this | Provides a parent topic for a set of related topics, and | <topic> |

| Information Type | Answers the Questions . . . | Description | Tag |
|---|---|---|---|
| | collection of subtopics? | introduces these topics. It often serves as a high-level task topic. Frequently, an orientation topic contains only the short description. The DITA Open Toolkit automatically builds an internal TOC from the organization of the orientation topic's child topics in the map. | |

**See Also:**

Creating Topics from Publication Manager
: Create a topic from the Browse Repository dialog box in Publication Manager.

Creating Topics from Arbortext Editor
: Create a topic from the **SDL LiveContent** menu in Arbortext Editor as an alternative to the Browse Repository dialog box in Publication Manager.

# DITA Topic Templates

InfoDev supplies a set of templates to use as the basis for new topics. It is strongly recommended that you use these templates rather than creating topics from scratch. The DITA Standards document is intended as accompanying documentation for the templates, and assumes that you are familiar with the templates.

All topic templates are located in `System/Editor template/Topics`. From within ArborText, clicking `SDLLivecontent > New` opens a dialog box in which you can select one of the following templates.

***Table 25-2 Topic Templates***

| Template Name | Topic Type (Element) | Purpose |
|---|---|---|
| Concept | <concept> | Use for all concept topics: overview, introduction, detailed explanation, and so on. |
| Glossary Entry | <glossentry> | Use for a glossary entry consisting of a glossary term (`<glossterm>`) and definition (`<glossdef>`). |
| Glossary Group | <glossgroup> | Use to group multiple `<glossentry>` topics within a single collection. |
| Graphic Description | <topic> | Use for an accessible description of a figure or image. |
| Orientation | <topic> | Use as a parent topic that contains subtopics. Note that there is no `<orientation>` topic type. An |

| Template Name | Topic Type (Element) | Purpose |
|---|---|---|
| | | orientation topic is a specialized topic of general type `<topic>`. |
| Reference | <reference> | Use as a general reference template when the specific reference templates do not meet your requirements. In most cases, you will use a specific reference template. |
| Reference API | <reference> | Use for any programmatic APIs, for example, Java or C++. |
| Reference Command | <reference> | Use for any non-SQL commands, for example, RMAN commands. |
| Reference Parameter | <reference> | Use for initialization parameters. |
| Reference PL/SQL Package | <reference> | Use for PL/SQL packages. |
| Reference SQL Statement | <reference> | Use for SQL statements. |
| Reference View | <reference> | Use for data dictionary and V$ views. |
| Task | <task> | Use for all task topics except task code examples (which have their own template). |
| Task Code Example | <task> | Use only for task code examples. |
| Topic | <topic> | Use only when one of the other templates is not adequate, which should rarely if ever occur. |

**See Also:**

Creating Topics from Publication Manager
: Create a topic from the Browse Repository dialog box in Publication Manager.

Creating Topics from Arbortext Editor
: Create a topic from the **SDL LiveContent** menu in Arbortext Editor as an alternative to the Browse Repository dialog box in Publication Manager.

# DITA Skill Set Requirements

To write effective documentation in DITA, writers need minimalist writing skills and topic-based writing skills.

## Minimalist Writing Concepts

Minimalist writing focuses on the audience, not the product, emphasizes learning by doing, and eliminates nonessential content.

Minimalist documentation includes only the information required by users to complete tasks and excludes nonessential information. Research shows that users are more interested in completing tasks than in reading documentation. They use documentation for specific information, such as how to complete a task or for the

answer to a question. Unnecessary information in documentation can make it more difficult for users to find the information they need to accomplish their goals.

Minimalist documentation is based on the following principles:

- Focus on the audience and user goals.
- Emphasize tasks.
- Use fewer words.
- Remove nonessential information.

Minimalist documentation provides the following benefits:

- More usable documentation
- More clear and concise documentation
- Decreased costs
- Ability to cover more with less time and effort

Follow these guidelines for writing minimalist topics:

- Structure topics around user goals.
- Include plenty of goal-oriented examples.
- Limit conceptual information to what is needed to understand tasks.
- Include instructions that correct common problems that arise during tasks.
- Use techniques that reduce the number of words.
- Avoid describing how a feature works unless it is important for completing tasks.
- Begin each topic with the most important information.
- Use active voice.
- Write in the second person.
- Describe the user goal in the topic heading.
- Create concise, specific, and informative topic headings.

**See Also:**

Topic-Based Writing Concepts
: Writers separate content into topics of the following types: concept, task, reference, and orientation. Topic-based writing enables topic reuse and promotes consistency.

Minimalist Class Slides

## Topic-Based Writing Concepts

Writers separate content into topics of the following types: concept, task, reference, and orientation. Topic-based writing enables topic reuse and promotes consistency.

A topic is self-contained content that meets the following requirements:

- It can stand on its own.
- It focuses on a single point without expanding into other topics.
- It is reusable.

Each topic must be short enough to explain one subject, describe one task, or answer one question. Each topic must also be long enough to be useful on its own and understood without other content. Good topics also have meaningful titles and links to related content.

**See Also:**

Minimalist Writing Concepts
: Minimalist writing focuses on the audience, not the product, emphasizes learning by doing, and eliminates nonessential content.

InfoDev DITA Information Types
: InfoDev DITA information types, also called topic types, are the building blocks for documentation. The types are task, concept, reference, and orientation.

DITA Standards for Topic Types
: All InfoDev documentation is built using only four topic types: task (`<task>`), concept (`<concept>`), reference (`<reference>`), and orientation (which uses `<topic>`). A glossary uses a different element (`<glossgroup>`), but is not considered a separate topic type.

Topic-Based Writing Slides

# 26

# DITA Standards for Topic Types

All InfoDev documentation is built using only four topic types: task (<task>), concept (<concept>), reference (<reference>), and orientation (which uses <topic>). A glossary uses a different element (<glossgroup>), but is not considered a separate topic type.

## DITA Standards for Task Topics

This section describes the standards for the <task> element. These standards are specific to the InfoDev implementation of DITA. They are intended for all authors in InfoDev.

For an introduction to task topics, see Chapter 2 in *DITA Best Practices* by Laura Bellamy. See also the DITA specification for task elements.

### About Task Topics

A task topic, which uses the <task> element, explains how to achieve a goal using a procedure. A task answers the question, "How do I?" It is the most common topic type.

An effective task is short, retrievable, and reusable. If too much conceptual or reference information appears in a task, then expert users must wade through information that they might not need.

A task topic contains exactly one task. A task contains exactly one procedure. A template for a task topic is located in `System/Editor template/Topics`.

#### About Procedures

A procedure is a sequence of steps. Every task must have exactly one procedure, wrapped in either <steps> for a multi-step procedure, or <steps-unordered> for a single-step procedure.

As shown in the template, a procedure may be preceded by context and prerequisites, and followed by postrequisites, expected results, and examples. Each procedure step includes the following:

- A command in active voice (such as "Click **Go**" )
- Optional information, usually in the form of a paragraph element under the step
- Optional step results, usually the system response

#### About Complicated Tasks

Procedures that are more than 9 steps can be difficult to follow, especially when some steps are complex or have many sub-steps. If a complicated task consists of multiple separate tasks, then place each task in a separate topic, and use an orientation topic type to order the subordinate tasks. A benefit of this approach is that you can more easily reuse these shorter task topics elsewhere in the documentation set.

## Main Elements in Task Topics

The main elements used for task elements include the task element, which encompasses the standard title, shortdesc, and prolog elements, followed by the task element. Within the task element is a collection of task-specific elements that are designed to accommodate prerequisites, the task itself, results, and examples.

### Purpose

The `<task>` element is the top-level element for a task topic. A task provides either single-step or multi-step instructions. A task answers the question of "how to?" by explaining what to do and the order in which to do it.

### Element Prototype

The following shows the typical sequence of elements for a two-step procedure:

```
<task>
  <title></title>
  <shortdesc></shortdesc>
  <prolog></prolog>
  <taskbody>
    <context></context>
    <prereq></prereq>
    <steps>
      <step><cmd></cmd><stepresult></stepresult></step>
      <step><cmd></cmd><stepresult></stepresult></step>
    </steps>
    <result></result>
    <example></example>
  </taskbody>
</task>
```

### Main Elements

This table shows the most relevant elements for tasks.

| Task Topic Element | Mandatory ? | Description |
|---|---|---|
| `<task>` | Yes | Parent element, which encompasses all elements in the task topic |
| `<shortdesc>` | Yes | Standard short description element |
| `<prolog>` | No | Standard prolog element |
| `<taskbody>` | Yes | Includes a set of task-specific elements, which are described in the remainder of this table |
| `<prereq>` | No | Describe prerequisites the procedure may need. If you must refer the user to another section, then put the reference links needed for prerequisite in the related-links section, not in the prerequisite paragraph. You can use many common elements in the <prereq> element, such as ordered lists, unordered lists, and paragraphs. |
| `<context>` | No | Provides a purpose for the task. Use this element to describe what users will gain from completing the task. Though the context element may have some conceptual |

| Task Topic Element | Mandatory ? | Description |
|---|---|---|
| | | information, do not use it in place of a related concept topic. |
| <steps> | Yes unless <steps-unordered> used | Provides an ordered list of steps that the user must follow. The steps element contains individual step elements. Each step in a steps element appears as a numbered step.<br><steps> and <steps-unordered> are mutually exclusive. |
| <steps-unordered> | No unless <steps> not used | Used for a single-step procedure. Each step element under the steps-unordered element renders as a bulleted item. You can use one or more step element within the steps-unordered element.<br><steps> and <steps-unordered> are mutually exclusive. |
| <step> | Yes | Displays a step in either a multi-step (<steps>) or single-step (<steps-unordered>) procedure. |
| <substeps> | No | Describes each sub-step that a user must follow to complete the step. It has the same structure as the step element, but it cannot contain another level of <substeps> or the choice or choice-table element. |
| <substep> | No | Breaks a step into a series of actions. This element contains one or more individual <substep> elements. Use the <substeps> element only when necessary. |
| <cmd> | No | Describes the action that the user must take in a step element. Write this action using imperative voice. This element must be the first element in the step element. |
| <choices> | No | Used within the <steps> element; the choices element provides a choice between one or more actions. It contains individual choice elements. |
| <choice> | No | Includes each choice that a user is presented with making. The choice items can be bullets when they are rendered. |
| <choicetable> | No | Similar to the choices element except that it provides two-column table so that you can include a description for each choice |
| <tutorialinfo> | NOT USED | Enables you to provide additional instructional information if the task is part of a tutorial. It appears after the <cmdname> element in the step element. |
| <stepresult> | No | Describes the result of completing a step, such as a dialog box opening. Use <stepresult> to assure users that they are on the right track, but do not use them for every step.<br>For example, 'The Create User page appears, with CREATE SESSION listed as the system privilege for user mweiss." |

| Task Topic Element | Mandatory ? | Description |
|---|---|---|
| <info> | No | Provides additional information that users might need when they complete a step. The description must be brief and only contain minimal conceptual information.<br>For example: "You do not need to specify additional users." |
| <stepxmp> | No | Provides an example of what happens when a step is completed. The example can include a few words, a paragraph, a figure, a table, or other information to illustrate the task. Step examples often provide specific data-entry characters. |
| <stepsection> | No | Provides expository text before a step element. Though the element is specialized from the li element and has the same content model as a list item, it is not designed to represent a step in a task. You can use either declarative or imperative voice.<br>For example: "The remaining steps are very important."<br>• "The remaining steps are very important.<br>• "If you had installed the DBMS_CRMB package, then complete the remaining steps." |
| <result> | No | Provides the expected outcome of the task. It can include a final description that uses figures, tables, or audiovisual cues that show the user that he or she has successfully completed the task. Add the result element after the steps element.<br>For example, "After you complete this task, all user accounts are created." |
| <example> | No | Displays the expected outcome of the entire task. It can include code samples, figures, screen shots, and other samples that show users that they have completed the task. The example element includes a title element, and a p (paragraph) element. From there, you can add the necessary elements you need, such as the <codeblock> element if you need to show code output. |
| <postreq> | No | Provides information that users should know after they have completed a task, or information about tasks that must be completed next. This kind of information can include actions that must be completed before the user can see expected results, such as restarting the computer. It can include information about what the user must read or cross-reference to verify that they properly completed the task. The <postreq> element often has links to the next task or tasks in the related-links section.<br>For example: "After you enable Oracle Label Security, you must unlock the LBACSYS user account." |

### Templates

Templates for a task and task example are located in `System/Editor template/Topics`. InfoDev recommends that you create new task topics using the templates.

## Guidelines for Task Topics

These guidelines provide both requirements (what you must do) and best practices (what it is recommended to do) when creating task topics.

Consider the following guidelines when creating a task topic:

- Create a new task using the task template.

  When creating a new topic, remember to delete unused elements inherited from the template. If you do not, empty lines can appear in the HTML and PDF output.

- Begin the `<title>` element with a gerund, as in "Creating a User" or "Backing Up a Tablespace."

  If relevant, put the goal of the task in the title, as in "Creating a User to Administer a PDB." The goal is to make the title as informative as possible.

- Create a short description for the task topic (mandatory).

  The short description appears in the HTML and PDF output as the first paragraph of the topic. Do not create a first paragraph for your topic that merely restates the short description.

- Create exactly one step-by-step procedure for every task topic (mandatory).

  For multi-step procedures, use the `<steps>` element. In all other cases, including single-step procedures, use the `<steps-unordered>` element.

- Break complex procedures into smaller, individual task topics.

  Procedures with more than 9 steps are difficult to follow. Create a parent topic, sometimes known as a high-level task, that describes the overall task flow. You can then nest the child topics under the high-level task in a logical order.

### Guidelines for Content Surrounding a Procedure

Follow these guidelines when putting text before or after the procedure:

- Provide no more content before the procedure than is necessary to make sense of the procedure.

  Sometimes the title and short description are sufficient in themselves to provide context. If the goal of the task is not apparent in the title or the short description, then you can introduce the procedure with the "To do X:" convention. In all other cases, use the `<context>` element for background information, and the `<prereq>` element for prerequisites. The `<prereq>` element is wrapped inside the `<taskbody>` element.

- Write prerequisites as imperative statements that call users to action.

  Include links to tasks that users must complete before beginning the current task.

- Provide content after the procedure when appropriate using `<postreq>` (what to do next), and `<result>` (the change brought about by the procedure).

  Often these elements are not needed. Ask, "Does the user really need to know this?"

- To create an example after a procedure, use the `<example>` element; to create an example after a step within a procedure, use the`<stepxmp>` element.

### Guidelines for Steps in a Procedure

Follow these guidelines for elements within `<steps>` and `<steps-unordered>`:

- Begin each step with location context (when necessary), followed by an imperative.

  For example, write "In the User window, click `Create`." If the context is not needed, as in a sequence of steps performed on the command line, then begin with an imperative, as in "Run the `BACKUP DATABASE` command." Place commands within the `<cmdname>` element. Ensure that a step never contains only an indicative statement such as "Now you back up the database."

- When a step is optional, select `Modify Attributes`, select `Other`, and then set `importance` to `optional`.

  In the HTML and PDF output, modifying this attribute generates a bold **Optional:** flag at the beginning of the step. Do not insert the literal word "Optionally" at the beginning of the step. Otherwise, the word "Optional" appears twice in the output.

- In a step that introduces sub-steps, use an imperative that states the goal of the sub-steps, such as "Load the statements into the SQL tuning set as follows."

  Avoid sentences such as "Follow these steps" as the introduction to sub-steps because such sentences give users no idea what action the sub-steps will describe.

- When a step presents the user with options, as in "Choose one of the following," use the `<choices>` or `<choicetable>` element within the `<step>` element.

- When a step has sub-steps, use the `<substep>` element within the `<step>` element.

- To describe the outcome of a single step, use the `<stepresult>` element within `<step>`.

  Use `<stepresult>` only for results are important or not obvious. For example, if the step closes a dialog, then do not add the step result "The dialog box closes." Consider not inserting step results such as "The User dialog box appears." Instead, in the next step, start with UI location, as in "In the User dialog box, create Create."

- When instructing the user to select an item from a drop-down list, use the following form: "From the *drop_down_list_name* drop-down list, select *item_name*." For example: From the **Content type** drop-down list, select **Module**.

## Examples of Task Topics

These examples show some of the different elements that you can use within `<task>`.

***Example 26-1 Simple Multistep Task: Example***

```
<task>
<title>Backing Up Server Parameter Files with RMAN</title>
<abstract>RMAN automatically backs up the current server parameter
file in certain cases. <shortdesc>The BACKUP SPFILE command backs
up the parameter file explicitly. The server parameter file that is
backed up is the one currently in use by the instance.</shortdesc></
abstract>
<taskbody>
<steps>
  <step>Start RMAN and connect to a target database and a recovery
catalog (if used).</step>
  <step>Ensure that the target database is mounted or open.
    <info>The database must have been started with a server
```

```
parameter file. If the instance is started with a client-side
initialization parameter file, then RMAN issues an error if you
execute <cmdname>BACKUP ... SPFILE</cmdname>.</info>
  </step>
  <step>Execute the <cmdname>BACKUP ... SPFILE</cmdname> command.
    <info>The following example backs up the server parameter file
to tape:</info>
    <stepxmp>BACKUP DEVICE TYPE sbt SPFILE;</stepxmp>
  </step>
</steps>
</taskbody>
</task>
```

***Example 26-2 Simple Single-Step Task: Example***

```
<task>
<title>Changing the Fast Recovery Area to a New Location</title>
<shortdesc>The fast recovery area is an Oracle Database managed
space that can be used to hold RMAN disk backups, control file
autobackups and archived redo log files.</shortdesc>
<taskbody>
<p>To move the fast recovery area of your database to a new
location:</p>
<steps-unordered>
  <step>In SQL*Plus, change the DB_RECOVERY_FILE_DEST
initialization parameter.</step>
    <info>For example, enter the following command to set the
destination to the ASM disk group disk1:</info>
    <stepxcmd>ALTER SYSTEM SET DB_RECOVERY_FILE_DEST='+disk1'
SCOPE=BOTH SID='*';</stepxcmd>
  </step>
</steps-unordered>
<result>After you change the DB_RECOVERY_FILE_DEST initialization
parameter, all new fast recovery area files are created in the new
location.</result>
</taskbody>
</task>
```

# DITA Standards for Concept Topics

This section describes the standards for the `<concept>` element. These standards are specific to the InfoDev implementation of DITA. They are intended for all authors in InfoDev.

For an introduction to concept topics, see Chapter 3 in *DITA Best Practices* by Laura Bellamy. See also the DITA specification for concepts elements.

## About Concept Topics

A concept topic, which uses the `<concept>` element, explains or defines an idea as a means of providing background for tasks or reference topics.

A concept answers the following questions:

- "What is it?"
- "What is its purpose?"
- "How does it work?"

A good concept topic supports a task or reference, rather than the other way around. Because one of the functions of a concept is to define terms, a concept often cross-references glossary entries. However, a concept topic is not a list of glossary terms.

A concept differs from a glossary entry because of its length, the type of elements it contains, and its purpose. A typical glossary entry might define the concept and optionally state its purpose in more than three sentences. A concept topic usually has multiple paragraphs, and may contain bulleted lists, graphics, and tables.

A concept differs from a reference topic in its purpose. A reference topic either defines a UI component, or lists guidelines. A concept explains what something is and why it is important. A good analogy is the difference between a dictionary entry (reference topic) and an encyclopedia entry (concept topic).

A concept topic contains exactly one concept. A concept must never contain a step-by-step procedure or any other task information. A template for a concept topic is located in `System/Editor template/Topics`.

### About Complicated Concepts

If you find that a topic describes more than one concept, then break this information into separate concept topics. Use an orientation topic to mention how these topics are related. The advantage of separating concepts is that users can read only what they need, and you can more easily reuse those concept topics elsewhere in your documentation.

### About Irrelevant Concepts

Concept topics must only have information that is relevant to user tasks or reference. Because the primary source of doc is the functional spec, it is tempting to dump the concepts in a spec into a topic. However, concepts relevant for developers may not be relevant for users. Ask, "Does this concept have implications for a user task?" If a user cannot act on the information, then although the concept may be interesting, it is probably irrelevant and should be removed.

## Main Elements in Concept Topics

The `<concept>` element is the top-level element for a concept topic. It encompasses the standard `<title>`, `<shortdesc>`, and `<prolog>` elements, followed by the `<conbody>` element. Within `<conbody>` you can use elements such as paragraphs, lists, tables, figures, examples, and sometimes sections.

### Purpose

A concept provides background information for a task or reference topic, or introduces a feature. Unlike a reference topic, a concept explains and illustrates. A concept answers the following questions:

- "What is it?"
- "What is its purpose?"
- "How does it work?"

### Element Prototype

The following shows a typical sequence of elements for a concept topic:

```
<concept>
  <title></title>
  <shortdesc></shortdesc>
  <prolog></prolog>
  <conbody>
    <p></p>
    <ul>
```

```
      <li><p></p></li>
      <li><p></p></li>
    </ul>
  </conbody>
</concept>
```

### Main Elements

This table shows the most relevant elements for concepts.

| Concept Topic Element | Mandatory ? | Description |
|---|---|---|
| <concept> | Yes | Parent element, which encompasses all elements in the concept topic |
| <shortdesc> | Yes | Standard short description element |
| <prolog> | No | Standard prolog element |
| <conbody> | Yes | Includes a set of concept-specific elements, which are described in the remainder of this table |
| <section> | No | Represents an organizational division in a topic. Use sections to organize subsets of information that are directly related to the concept. Sections are not hierarchical and cannot be nested. A section may have an optional title. |
| <sl> | No | Contains a simple list of items of short, phrase-like content, such as a list of characteristics. The output has no bullets. |
| <ul> | No | Provides an unordered, bulleted list of items. |
| <dl> | No | Provides 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.<br>You can also provide an optional heading for the terms and definitions, using the <dlheadelement>, which contains header elements for those columns. |
| <codeblock> | No | Displays of program code. Like the <pre> element, content of this element has preserved line endings and is output in a monospace font. |
| <lines> | No | Represents dialogs, lists, text fragments, and so on. 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. |
| <msgblock> | No | 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. |
| <screen> | No | Contains or refers to a textual representation of a computer screen or UI panel (window). Use <screen> to contain representations of text-based online panels, text consoles ("term" or "curses" windows, for example), or other text- |

| Concept Topic Element | Mandatory ? | Description |
|---|---|---|
| | | based UI components. The default print representation is to enclose the screen within a box, suggesting a computer display screen. In contrast to graphical screen captures normally used to represent GUI parts (see the image element description), this element specifically supports constructions for which text is the primary content. |
| `<fig> and <image>` | No | A `<fig>` is a formal graphic element, requiring a title and an id, whereas `<image>` is informal. A `<fig>`, but not an `<image>`, can be cross-referenced.<br>Use the `<image>` element to insert a graphic into a `<fig>`. The `@href` attribute points to the relative path location of the graphic file. Optionally, use the `<alt>` child element for alt text, similar to the FrameMaker `AltText` attribute. For a graphic description, use either `<alt>` or `<longdescref>`, both of which are associated with `<image>`.<br>Set `@expanse` to "page" to get a wide figure. Set `@placement` to "break". |
| `<table>` | No | 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. In addition, the table includes a `<desc>` element, which enables table description that is parallel with figure description. |
| `<term>` | No | Identifies glossary entries. To create a link for a glossary term, wrap the text in your topic in a `<term>` element. Modify the `<term>` attribute `@keyref` to associate the `keyref` with a `<glosskey>` element in the bookmap. A `<glossarylist>` is located in the bookmap, and has multiple `<glossentry>` elements as children. Each `<glossentry>` must be in its own topic, and typically, its own file. Create a `<glossentry>` topic by selecting the topic type "InfoDev DITA Glossary." A `<glossentry>` contains a `<glossterm>` and a `<glossdeff>` . |

### Templates

There is only one concept template. It is located in `System/Editor template/Topics`. InfoDev recommends that you create new concept topics using the template.

## Guidelines for Concept Topics

These guidelines provide both requirements (what you must do) and best practices (what it is recommended to do) when creating concept topics.

Consider the following guidelines when creating a concept topic:

- Write one concept for each topic.

  If a topic becomes too complex, break it into multiple concept topics, and group them using an orientation topic.

- Begin the title with a noun or noun phrase, as in "About User Privileges," "Overview of User Privileges," or "User Privileges." Whether you use "About" or "Overview" is up to your discretion.
- Never include a step-by-step procedure for the user to follow. Tasks must go in a task topic.

  However, a concept topic may use ordered lists to describe stages in a process (Oracle Database creates the SGA and background processes, and then opens the control file, and then opens the database).
- For `<section>` elements, give each a `<title>` element using the same naming conventions as the concept topic. Use sections judiciously.

  The `<conbody>` element is the main body-level element for a concept. Like the `<body>` element of a general `<topic>`, `<conbody>` allows paragraphs, lists, and other elements as well as sections and examples. However, `<conbody>` has a constraint such that a section or an example can be followed only by other sections or examples.
- Most concept topics benefit from a figure, although it is not required.

### Guidelines for Ordering Conceptual Information

When deciding how to organize a concept topic, consider these guidelines:

- Use an inverted pyramid style, that is, put the most important idea at the beginning of the topic, and supplemental explanation afterward. Do not build up to the point. Get to the point straight away.
- Use the acronym "DRIES" as a rule of thumb. DRIES stands for: Definition, Reason (purpose), Implementation (how it works, what it contains), Example, and See Also (a `<reltable>`). For example, in "About Tablespaces," begin by defining "tablespace," then explain what the point of a tablespace is, explain how tablespaces work and what their components are, give an example of a `CREATE TABLESPACE` statement, and end with a `<reltable>` of related links.
- If a concept is too big to fit in one topic, create subtopics for Purpose, How It Works, and so on.

### Reasons for Creating Concept Topics

When considering whether to create a concept topic, consider these guidelines:

- Create a concept only if a glossary entry is inadequate.

  If a concept is adequately covered in one or two sentences, then make it part of a task, reference, or glossary instead. If you need more than a paragraph, or you need to explain how something works, then create a separate concept topic.
- Create concept topics to support tasks and goals, not the other way around.

  Users read technical information to achieve a goal. The goal for users of most technical products is not to understand a concept but to complete a task, such as creating a tablespace, backing up a database, or dropping a user.

## Examples of Concept Topics

These examples show some of the different elements that you can use within <concept>.

### *Example 26-3 Simplest Concept: Example*

In its simplest form, a concept topic has a short description, but no text within the <conbody>. The short description appears in the output as the first paragraph of the topic.

```
<concept>
<title>Savepoints</title>
<prolog>
  <metadata>
    <keywords>
      <indexterm>[index: savepoints]
      </indexterm>
      <indexterm>[index: transactions
                 <indexterm>, savepoints</indexterm>]
      </indexterm>
    </keywords>
  </metadata>
</prolog>
<shortdesc>A savepoint is a user-declared intermediate marker
within the context of a transaction. Internally, this marker
resolves to an SCN. Savepoints divide a long transaction into
smaller parts. If you use savepoints in a long transaction, then
you have the option later of rolling back work performed before the
current point in the transaction but after a declared savepoint
within the transaction.</shortdesc>
<conbody></conbody>
</concept>
```

### *Example 26-4 Concept with Bulleted Lists: Example*

This topic provides an overview of tables. There is no "how to" information, which must never appear in a concept. Notice also how the topic is relatively short even though tables are a huge topic. The related conceptual information appears in separate subtopics. For example, there is one subtopic on columns, another on rows, another on data types, another on integrity constraints, and so on.

```
<concept>
<title>Overview of Tables</title>
<shortdesc>A table is the basic unit of data organization in an
Oracle database. A table describes an entity, which is something of
significance about which information must be recorded. For example,
an employee could be an entity.</shortdesc>
<prolog>
  <metadata>
    <keywords>
      <keyword>tables</keyword>
    </keywords>
  </metadata>
</prolog>
<conbody>
<p>Oracle Database tables fall into the following basic categories:
<ul>
    <li><p>Relational tables</p>
        <p>Relational tables have simple columns and are the most
common table type. Example 2-1Example 2-1 shows a CREATE TABLE
statement for a relational table.</p></li>
    <li><p>Object tables</p>
        <p>The columns correspond to the top-level attributes of an
```

```
object type. See Overview of Object Tables.</p></li>
</ul>
<p>You can create a relational table with the following
organizational characteristics:
<ul>
    <li><p>A heap-organized table does not store rows in any
particular order. The CREATE TABLE statement creates a heap-
organized table by default.</p></li>
    <li><p>An index-organized table orders rows according to the
primary key values. For some applications, index-organized tables
enhance performance and use disk space more efficiently. See
Overview of Index-Organized Tables.</p></li>
    <li><p>An external table is a read-only table whose metadata is
stored in the database but whose data is stored outside the
database. See Overview of External Tables.</p></li>
</ul>
<p>A table is either permanent or temporary. A permanent table
definition and data persist across sessions. A temporary table
definition persists in the same way as a permanent table
definition, but the data exists only for the duration of a
transaction or session. Temporary tables are useful in applications
where a result set must be held temporarily, perhaps because the
result is constructed by running multiple operations.</p>
</conbody>
</concept>
```

# DITA Standards for Reference Topics

This section describes the standards for the `<reference>` element. These reference topic standards are specific to the InfoDev implementation of DITA. They are intended for all authors in InfoDev.

For an introduction to reference topics, see Chapter 4 in *DITA Best Practices* by Laura Bellamy. See also the DITA specification for concepts elements.

## About Reference Topics

A reference topic, which uses the `<reference>` element, provides quick access to a set of facts (UI components or rules), without providing explanation.

A reference topic answers either of the following questions:

- What are the syntax and semantics of this UI component?
- What is the list of rules, restrictions, or guidelines for a task or reference topic?

A reference topic gives “facts without explanation,” which distinguishes it from a concept. A UI reference functions as a dictionary: users want to quickly find a definition, and then move on. A rules reference is analogous to a home appliance data sheet: a list of sentences of the form “Do this” and “Don’t do that.” A concept topic is similar to an wikipedia entry, which introduces a topic, states its purpose, and explains how it works using lists, figures, tables, and so on.

A reference also gives “facts without steps.” Thus, in contrast to task topics, reference topics never have procedures that explain how to achieve a goal.

In software documentation, UI reference topics describe components such as the following:

- SQL statements
- Commands
- Packages

- Utilities
- Error Messages
- CLI or GUI components

Topics enable users to learn the specifics of APIs, such as their syntax, input, expected output. A UI reference topic has a short description, followed by the reference material, which can include sections for items such as Overview, Security Model, Constants, Restrictions, Exceptions, Operational Notes, and Examples. In most cases, place each API, if it is small, into its own reference topic.

A reference topic contains exactly one type of reference information. A reference must never contain a step-by-step procedure or any other task information. The templates for reference topics are located in `System/Editor template/Topics`.

## Main Elements in Reference Topics

The main elements for reference topics include the `<reference>` element, which encompasses the standard `<title>`, `<shortdesc>`, and `<prolog>` elements, followed by the `<refbody>` element. Within `<refbody>` is a collection of reference-specific elements that present prerequisites, purpose sections, syntax, properties, examples, and so on.

### Purpose

Reference topics document facts (either UI components, or lists of rules) without explanation. A reference topic answers either of the following questions:

- "What does this UI component do?"
- "What are the rules associated with the task or reference?"

### Element Prototype

The following shows a typical sequence of elements for a reference topic:

```
<reference>
  <title></title>
  <shortdesc></shortdesc>
  <prolog></prolog>
  <refbody>
    <section><title>Purpose</title></section>
    <refsyn><title>Syntax</title></refsyn>
    <section><title>Parameters</title></section>
  </refbody>
</reference>
```

### Main Elements

This table shows the most relevant elements for reference topics. Note that the `<p>` element is not permitted unless embedded in wrapping elements such as `<section>` and `<refsyn>`.

| Concept Topic Element | Mandatory? | Description |
|---|---|---|
| `<reference>` | Yes | Parent element, which encompasses all elements in the reference topic |

| Concept Topic Element | Mandatory? | Description |
|---|---|---|
| <shortdesc> | Yes | Standard short description element |
| <prolog> | No | Standard prolog element |
| <refbody> | Yes | Includes a set of reference-specific elements, which are described in the remainder of this table |
| <section> | No | Represents an organizational division in a topic. Use this element each subsection in the topic (such as Prerequisites, Properties, Syntax, Examples). Sections are not hierarchical and cannot be nested. A reference section must have a title. Do not embed <p> elements in sections without titles as a workaround for adding paragraphs. |
| <table> | No | Contains information such as parameter descriptions. |
| <properties> | No | Lists properties in a table by type, value, and description. For example, if you document how to design HTML pages, you might create a reference topic called "Cascading style sheet properties for body text" that includes a properties table describing the different cascading style sheet (CSS) properties for fonts. The table could describe font properties such as family, style, weight, or size.<br>To represent multiple values for a type, create additional property elements, use only the <propvalue> element (and <propdesc> when needed) for each successive value, and specify the <proptype> element only the first time. |
| <refsyn> | No | Contains (usually) syntax or signature content. Examples are a command-line utility's calling syntax, and an API's signature. Typically, <refsyn> contains a brief, possibly diagrammatic description of the subject's interface or high-level structure. Use the <syntaxdiagram> element inside a <refsyn> element to structure syntax diagrams. The syntax diagram may be included as an image, or coded in DITA within <synph>. |
| <example> | No | Contains examples that illustrate or support the current topic. An <example> can contain both discussion and sample code or outputs.<br>Use <codeblock> for sample code. Use <systemoutput> for output examples. Use <lines> for lines of text. For pre-formatted text such as email headers, use the <pre> element. |

### Templates

Templates for reference topics are located in `System/Editor template/Topics`. InfoDev recommends that you create new concept topics using the template. The following reference templates are available:

- Reference
- Reference API
- Reference Command
- Reference Parameter

- Reference PL/SQL Package
- Reference SQL Statement
- Reference View

## Guidelines for Reference Topics

These guidelines provide both requirements (what you must do) and best practices (what it is recommended to do) when creating reference topics.

Consider the following general guidelines when creating a reference topic:

- Use a noun phrase for the title. Examples include "Analytic Functions," "ALTER TABLE," "Database Resource Manager Data Dictionary Views," and "DBMS_REDACT". API reference topics must use the name of a command or API program unit.
- Do not include explanations or procedures. Reference topics are intended for quick lookups of facts. The mantra is "facts without explanation."
- For most reference information, use lists and tables rather than series of paragraphs.
- Paragraphs are not valid unless embedded in other elements. Do not create a section and then delete the `<title>` just to get around this restriction. (See "Guidelines for Sections" for a legitimate case in which you can delete `<title>`.)
- For very short reference topics, do not use the section element. An example might a short topic that contains one type of reference information, such as a table of predefined user roles.
- Do not use the `<ul>` element when a list has two parts. Use a table instead.
- Format the reference material consistently throughout.

  For example, if you create several reference topics that have tables of procedure parameters, then use the same formatting for the tables, use the same introduction to the tables, and provide the same amount of information. Even small inconsistencies are distracting and potentially confusing.

### Guidelines for Chunking

When deciding whether or how to break up reference information into separate topics, consider the following guidelines:

- Describe one type of reference material per topic, even for small APIs, as long as it is 1–2 pages.

  For example, for the`DBMS_ALERT` topic, you would describe only API settings for the `DBMS_ALERT` package.

- If the material is long (the rule of thumb is over 2 pages), then use multiple tables or multiple topics for each subcategory.

  For example, for a PL/SQL package, create one topic for each subprogram, not one topic for the entire package. Consider how the information will be searched for. For example, for large APIs, you could put each method into its own reference topic.

- When you are unsure whether to create a separate topic, consider how users will search for the information, and also whether the information is likely to grow over releases. If the information grows by 50% every release, then consider putting it into its own topic.

### Guidelines for Tables

When creating tables in a reference topic, consider the following guidelines:

- Always use the `<table>` element, even if the table is very simple. Do not use the simpletable element. Provide the required table summary as a desc element.
- For table titles, use specific names, such as "System Privileges (Organized by the Database Object Operated Upon)". For a very simple table in a small reference topic, omit the title. It is not necessary.
- Do not have too much information in the table. It becomes difficult for users to read.
- Do not have more than four or five columns in a table.
- Do not make the last column a catch-all column for comments, examples, and descriptions.

### Guidelines for Syntax Diagrams

When creating a syntax section, consider the following guidelines:

- Place API calling syntax and method signatures within a `<refsyn>` section.
- You can include syntax as an image, or code it in DITA within a `<synph>` element.

### Guidelines for Sections

When creating a `<section>`, consider the following guidelines:

- Use the `<section>` element in the `<refbody>` element for each section, such as "Prerequisites," "Syntax," and "Examples."
- Always use exactly one `<title>` within each `<section>`. The only exception is when a topic is so short that it has no sections, and you cannot use necessary elements without deleting the `<title>`.
- You cannot nest `<section>` elements. They are not hierarchical.

### Guidelines for Examples

When using the `<example>` element, consider the following guidelines:

- Use `<example>` to create a section that contains examples that illustrate or support the current topic. You can title an example, and use many elements such as lists, tables, and paragraphs within `<example>`.
- Use `<codeblock>` for sample code.
- Use `<systemoutput>` for output examples.
- Use `<lines>` for lines of text.

- Use the <pre> element for preformatted text such as email headers.

#### Guidelines for Property Tables

When creating a list of properties, consider the following guidelines:

- Use <properties> to provide a table of properties for the topic.
- Each property can include the type, value, and a description.
- To represent multiple values for a type, create additional property elements. Use only the <propvalue> element (and <propdesc> when needed) for each successive value. Specify the <proptype> element only the first time.

### Examples of Reference Topics

These examples show some of the different elements that you can use within <concept>.

#### PL/SQL Procedure: Example

```
<reference>
  <title>INITIALIZE Procedure</title>
  <shortdesc>This procedure initializes the generator.</shortdesc>
<refbody>
<refsyn>
  <synph><kwd>DBMS_RANDOM.INITIALIZE (</kwd><var>val</var><kwd>IN
BINARY INTEGER);</kwd></synph>
</refsyn>
<section>
  <title>Usage Notes</title>
  <p>his procedure is obsolete because it calls the SEED
Procedures.</p>
</section>
</refbody>
</reference>
```

## DITA Standards for Orientation Topics

This section describes the standards for orientation topics, which use the <topic> element (there is no <orientation> element). These reference topic standards are specific to the InfoDev implementation of DITA. They are intended for all authors in InfoDev.

For an introduction to reference topics, see Chapter 1 in DITA Best Practices by Laura Bellamy. See also the DITA specification for <topic>.

### About Orientation Topics

An orientation topic, which uses the <topic> element, provides a parent topic in the table of contents for a set of related topics, and introduces those topics. Thus, it serves as a high-level topic organizer.

An orientation topic can serve as a high-level task topic. It typically has a heading similar to that of a task topic (starting with a gerund), and is a parent topic for a series of topics that help users complete a high-level task such as installing, configuring, administering, and so on. Such a high-level task is explained further by a series of conceptual topics, task topics, and reference topics, all children of the high-level task topic.

Frequently, an orientation topic contains only the short description as its content. The short description provides introduction to the chapter or section. In some cases, the topic can include a few paragraphs of background or introductory information prior to the internal table of contents.

The topic also contains set of cross references to the contained subsections, which are automatically generated by the DITA Open Toolkit from the titles and short descriptions of the child topics of the orientation topic in the map. The title (as a link) and short description of each child topic are automatically added to the parent (orientation) topic during publishing. In addition, the DITA OT automatically adds a link to the parent topic in each child.

No `<orientation>` tag exists. Instead, use the generic `<topic>` tag for orientation topics.

## Main Elements in Orientation Topics

The main elements for reference topics include the `<topic>` element, which encompasses the standard `<title>`, `<shortdesc>`, and `<prolog>` elements, followed by the `<body>` element. Within `<body>` you can place elements such as paragraphs, lists, and so on.

### Purpose

The `<topic>` element is the top-level element for an orientation topic. An orientation topic is the high-level container for other topics. An orientation topic answers the question of "How is this group of related topics organized?"

### Syntax

The following shows the typical sequence of elements in a simple orientation topic:

```
<topic>
  <title></title>
  <shortdesc></shortdesc>
  <prolog></prolog>
<body></body>
</topic>
```

### Main Elements

This table shows the most relevant elements for tasks.

| Task Topic Element | Mandatory ? | Description |
|---|---|---|
| `<topic>` | Yes | Parent element, which encompasses all elements in the task topic. |
| `<shortdesc>` | Yes | Standard short description element. Often, the short description is the only text required in the orientation topic. |
| `<prolog>` | No | Standard prolog element. |
| `<body>` | Yes | Wraps any additional text necessary to orient the reader. |
| `<ul>, <p>, etc.` | No | Use any of the standard elements to provide more orientation, if necessary. |

### Templates

The template for the orientation topic is located in `System/Editor template/Topics`. InfoDev recommends that you create new task topics using the templates.

## Guidelines for Orientation Topics

These guidelines provide both requirements (what you must do) and best practices (what it is recommended to do) when creating orientation topics topics.

The orientation topic type provides a parent topic in the table of contents for a set of related topics, and introduces those topics. Consider the following guidelines when creating an orientation topic:

- Try to put all necessary introductory text in the `<shortdesc>`. If you need to add more paragraphs in the `<body>`, then it is completely acceptable; however, in most cases, the `<shortdesc>` is all that is necessary.
- Because the DITA Open Toolkit automatically creates an "internal table of contents," do not create a list of subtopics manually.

  The title (as a link) and short description (link preview) of each child topic are automatically added to the parent (orientation) topic during publishing. In addition, the DITA OT automatically adds a link to the parent topic in each child topic.

  **Note:** If you want to manually put in an internal TOC, then you must disable the automatic generation of these links in the build settings in the output object of the publication or in DocBuilder.

- In the bookmap, set the `@collection-type` attribute of the `<topicref>` for the orientation topic to the value `family`.

  This setting causes additional links to be generated within the child topics. Each child topic cross-references all its sibling topics in automatically generated lists of "Related Concepts," "Related Tasks," and "Related References."

- When creating an orientation topic to serve as a high-level task topic, choose a title similar to that of a task topic (starting with a gerund).

  An orientation topic is the parent topic for a series of topics that help users complete a high-level task such as installing, configuring, administering, and so on. Such a high-level task is explained by a series of conceptual topics, task topics, and reference topics, all of which are children of the high-level task topic. For example, an orientation topic named "Administering Tablespaces" might be the parent of topics named "About Tablespaces" (a concept), "Creating Tablespaces" (a task), "Taking a Tablespace Offline" (a task), and so on.

- If the orientation topic is not a high-level task topic, then use the same heading title rules as the orientation topic's child topics. For example, if the orientation topic is the parent of conceptual topics, then use the title rules for conceptual topics.

## Examples of Orientation Topics

These examples show some of the elements that you can use within `<topic>` to create an orientation topic.

### High-Level Task Orientation: Example

Notice that there is no explicit internal table of contents, which might have been implemented as an unordered list. The heading begins with a gerund because the orientation topic contains task topics.

```
<topic>
  <title>Checking Memory Requirements</title>
  <shortdesc>Your target system must satisfy specific memory
requirements, which include RAM, swap space, and shared memory.</
shortdesc>
<body>
</body>
</topic>
```

# 27

# DITA Standards for Common Elements

Some DITA elements are common to most or all topics. For example, all topics must have short descriptions (<shortdesc>). This section describes some of the most important of these elements.

## DITA Standards for Short Descriptions

This section describes the standards for the <shortdesc> element. These reference topic standards are specific to the InfoDev implementation of DITA. They are intended for all authors in InfoDev.

For short description guidelines, see Chapter 5 in DITA Best Practices by Laura Bellamy. See also the DITA specification for <shortdesc>.

### About Short Descriptions

A short description (<shortdesc>) is a mandatory element that summarizes a topic and explains its purpose. The descriptions appear in links and in search engine results. A short description gives just enough information so that users can determine whether they need to read the topic itself.

The <shortdesc> element is valid between the <title> and main body of the topic. The short description is output in the following locations:

- The first paragraph of a topic
- In HTML, hover text for related links created either in a related-links element or in a relationship table (reltable) in the map
- The text under automatically generated child topic links at the end of a topic (The "link preview")
- Search engine results

Within <shortdesc>, you may use <keyword> or any other valid elements. Typically, a short description contains one to three sentences, and contains no more than 50 words. As much as possible within these parameters, the short description should answer the five W's (who, what, when, where, why), and sometimes the one H (how).

Use the <abstract> element when the initial paragraph of a topic is unsuitable for use as a link preview or for summaries, because, for example, it contains lists or tables, or because only a portion of the paragraph is suitable. In this case, place the <shortdesc> element within the <abstract> element. The output contains all text within the <abstract> element, including whatever is wrapped in the <shortdesc> element, but links and search results show only the short description.

***Example 27-1 Short Description: Example***

This example shows the short description for a conceptual topic about permanent tablespaces. The `<shortdesc>` appears after the `<title>` element, and before the `<prolog>` element.

```
<concept>
  <title>Permanent Tablespaces</title>
  <shortdesc>A permanent tablespace groups persistent schema
objects. The segments for objects in the tablespace are stored
physically in data files.</shortdesc>
  <prolog></prolog>
  <conbody>
    <p>Each database user is assigned a default permanent
tablespace.</p>
  </conbody>
</concept>
```

In the output, the text would appear in the following order (formatting is approximate):

**Permanent Tablespaces**

A permanent tablespace groups persistent schema objects. The segments for objects in the tablespace are stored physically in data files.

Each database user is assigned a default permanent tablespace.

***Example 27-2 Short Description Using <abstract>: Example***

`<abstract><shortdesc><shortdesc>`

```
<concept>
<title>Space Management in Data Blocks</title>
<abstract>As the database fills a data block from the bottom up,
the amount of free space between the row data and the block header
decreases. This free space can also shrink during updates, as when
changing a trailing null to a nonnull value. <shortdesc>The
database manages free space in the data block to optimize
performance and avoid wasted space.</shortdesc></abstract>
...
```

In the output, the first paragraph would look as follows (formatting is approximate):

**Space Management in Data Blocks**

As the database fills a data block from the bottom up, the amount of free space between the row data and the block header decreases. This free space can also shrink during updates, as when changing a trailing null to a non-null value. The database manages free space in the data block to optimize performance and avoid wasted space.

...

## Guidelines for Short Descriptions

These guidelines explain what to do and what not to do in short descriptions, and provide examples.

The primary goal of short descriptions is to provide context for readers so that they can decide whether to keep reading. These guidelines explain how to achieve this goal:

- Use complete, grammatical sentences.

For example, write "A tablespace groups related tables into a single logical container" rather than "Groups related tables." Remember that the short description appears as the first paragraph of your topic.

- Do not repeat or restate the title, or provide information that is not useful.

  For example, the short description for "Managing Storage with Tablespaces" must not be "A tablespace enables you to manage storage." Also, a description such as "DB2 and other non-Oracle relational databases use a data structure that serves the same function as a tablespace" is completely irrelevant to the task.

- Make the description short enough to avoid bogging down the reader.

  The short description must be short enough so that readers who do not need to read the topic do not waste time. For example, do not force the user to plough through five sentences to determine that the topic is irrelevant. Get to the point immediately. If the short description is too long, move the excess verbiage to the main body of the topic.

- Make the description long enough to be useful.

  The short description must be long enough so that readers can determine whether the topic is useful. For example, a description that merely repeats or restates the title wastes the reader's time.

- Do not make the description self-referential.

  For example, avoid the phrases "In this topic" and "This topic describes."

- Use the phrase "these" rather than the phrase "the following."

  Because a short description can appear in an orientation topic, "the following" is out of context. Use "these" instead. For example, for a task topic, a good short description might be: "Follow these steps to display and change kernel parameters." The word "these" encourages the user to click the link, whereas "the following" may cause the user to try to find the steps within the orientation topic.

- Do not introduce other elements, such as lists ("The following is a list of..."), procedures ("To create a synonym:"), figures ("Figure 1 shows the..."), and tables ("Table 1 explains...").

- Do not insert cross-references.

### Guidelines for the <abstract> Element

When using the `<abstract>` element with `<shortdesc>`, follow these guidelines:

- Only use <abstract> when the initial paragraph is not suitable as a summary. If it is suitable, use `<shortdesc>` instead.

- To avoid making `<shortdesc>` a separate paragraph in the output, wrap `<shortdesc>` around phrase-level content. When the contained `<shortdesc>` occurs as a peer to paragraph-level content, it is treated as block-level content and creates a separate paragraph on output of the topic.

### Guidelines for Task Topics

When writing short descriptions for task topics, follow these guidelines:

- State the goal of the task.

For example, a short description for a task topic entitled “Creating a Tablespace” might be “To place related tables in the same logical container, create a tablespace,” or “Use tablespaces to place related tables in the same logical container.” The short description “The goal of this task is to create a tablespace” merely restates the title, and so is not useful.

- If relevant, provide conceptual information that is necessary to understand the task.

  For example, if the topic describes how to create widgets to protect the database, and if the user may not know what widgets are, then it would be appropriate to include a sentence defining “widget.”

- If relevant, explain when the task must be performed.

  For example, the short description for the task “Creating a PDB” might be: “Within a single CDB, you can create a PDB for each separate application. Multiple PDBs enable you to consolidate data and improve manageability.”

- If relevant, describe restrictions or limitations.

#### Guidelines for Concept Topics

When writing short descriptions for concept topics, follow these guidelines:

- Answer the questions “What is this?” and “What is its purpose?”
- Use a definition when appropriate.

  For example, a short description for an overview of tablespaces might be “A tablespace is a logical container that stores related schema objects. ”

#### Guidelines for Reference Topics

When writing short descriptions for reference topics, follow these guidelines:

- Answer the question “How is the UI component defined?” or “What are the rules?”
- For UI descriptions, state what it does and what it is used for, and (if relevant) what it returns, and whether it is deprecated.

  For example, a short description for the BACKUP command might be “The BACKUP command backs up either database files (or backups of these files) to disk or tape. The command generates either image copies, which are bit-for-bit physical copies, or backup sets, which are proprietary logical containers.”

## Main Elements in Short Descriptions

The `<shortdesc>` element is mandatory in all topics. It expresses the purpose or theme of the topic, and serves as a link preview and in search results. Use the `<abstract>` element when the initial paragraph of a topic is unsuitable for use as a link preview or for summaries, or because only a portion of the paragraph is suitable.

#### Element Prototype

```
<title>
<abstract><shortdesc></shortdesc></abstract>
<prolog>
```

### Main Elements

This table shows the most relevant elements for short descriptions.

| Element | Mandatory? | Description |
|---|---|---|
| <shortdesc> | Yes | Occurs between the topic title and the topic body, as the initial paragraph-like content of a topic, or it can be embedded in an abstract element. The short description, which represents the purpose or theme of the topic, is also intended to be used as a link preview and for searching.<br>Use the <shortdesc> element when the first paragraph of topic content is simple enough to be suitable for use as a link preview or for summaries. Otherwise use the <abstract> element to provide richer content around the <shortdesc>. |
| <abstract> | No | Occurs between the topic title and the topic body as the initial content of a topic. It can contain paragraph-level content as well as one or more <shortdesc> elements that can be used for providing link previews or summaries.<br>Only use the <abstract> element when the initial paragraph of a topic is unsuitable for use as a link preview or for summaries, because, for example, it contains lists or tables, or because only a portion of the paragraph is suitable. |

### Usage Notes for <shortdesc>

The short description (<shortdesc>) is a mandatory element that occurs between the topic title and the topic body. You can also use the <shortdesc> element in a DITA map.

A short description serves different functions depending on the topic:

- In a task topic, it explains what the task information helps users accomplish, the benefits of the task, or the purpose of the task.
- In a concept topic, it introduces the concept and provides a concise answer to the question "What is this?" and in some cases "Why do I care about this?”
- In a reference topic, it briefly describes what the reference item does, what it is, or what it is used for.

## Short Description Examples

These examples show different types of short descriptions.

***Example 27-3 Short Description for Task: Example***

```
<concept>
<title>Setting Optimizer Statistics Preferences</title>
<shortdesc>You can set the default values of the parameters used by
automatic statistics collection and the DBMS_STATS statistics
gathering procedures. To automatically maintain optimizer
statistics for objects requiring non-default settings, set
optimizer statistics preferences at the table, schemaa, database,
or global levels.</shortdesc>
<conbody>
```

***Example 27-4 Short Description for Concept: Example***

```
<concept>
<title>Overview of Temporary Tables</title>
<shortdesc>A temporary table holds data that exists only for the
duration of a transaction or session. Temporary tables are useful
in applications where a result set must be buffered.</shortdesc>
<conbody>
```

***Example 27-5 Short Description for Concept Using <abstract>: Example***

In this example, the first sentence is appropriate for the short description, but the next three sentences in the paragraph provide irrelevant much information. By wrapping the entire paragraph in `<abstract>` but wrapping only the first sentence in `<shortdesc>`, the output appears as a single paragraph, but link previews and search results show only the single sentence in `<shortdesc>`.

```
<reference>
<title>About Processes</title>
<abstract><shortdesc>A process is a mechanism in an operating
system that can run a series of steps.</shortdesc> The process
execution architecture depends on the operating system. For
example, on Windows an Oracle background process is a thread of
execution within a process. On Linux and UNIX, an Oracle process is
either an operating system process or a thread within an operating
system process.</abstract>
```

***Example 27-6 Short Description for API Reference: Example***

```
<reference>
<title>CREATE_SQLWKLD Procedure</title>
<shortdesc>This DBMS_ADVISOR procedure, which is deprecated
starting in Oracle Database 11g, creates a new private SQL Workload
object for the user. A SQL Workload object manages a SQL workload
on behalf of the SQL Access Advisor.</shortdesc>
```

***Example 27-7 Short Description for PL/SQL Package: Example***

```
<reference>
<title>DBMS_STATS</title>
<shortdesc>This PL/SQL package enables you to view and modify
optimizer statistics gathered for database objects.</shortdesc>
```

***Example 27-8 Short Description for SQL Statement Reference: Example***

```
<reference>
<title>CREATE TABLE</title>
<shortdesc>This SQL statement creates either a relational table,
which is the basic structure to hold user data, or an object table.
Tables are created with no data unless a subquery is specified.</
shortdesc>
```

***Example 27-9 Short Description for Command Reference: Example***

```
<reference>
<title>RECOVER</title>
<shortdesc>This RMAN command recovers either data files, data file
copies, or corrupt data blocks.</shortdesc>
```

# DITA Standards for Figures and Images

This section describes the standards for the <fig> and <image> elements. These standards are specific to the InfoDev implementation of DITA. They are intended for all authors in InfoDev.

## About Figures and Images

In DITA, insert graphics either as figures (<fig>), which have titles, or untitled images (<image>). Nearly all readers prefer to view an illustration when it helps them avoid reading text.

For graphics in concept topics and reference topics, use a <fig>. To insert images that refer to UI features in a task or in a reference table, use an <image>.

For both figures and images, you must include alternate text to make the illustrations accessible. Adding alternate text is important for making documentation accessible to disabled users. For navigation images and icons such as buttons and arrows, use the <alt> element to add a brief description. For all other graphic descriptions, create a separate topic using the Graphic Description template to contain the text description. Reference this topic using a <longdescref> element.

DITA supports the inclusion of screenshots. The InfoDev standard is to store the screenshot as a JPG rather a GIF to enable better scaling, avoiding image distortion. The screenshot is scaled in the PDF, but renders full size in HTML. The InfoDev recommendation is to take the screenshot using HyperSnap, and then change the horizontal and vertical resolutions of the image by multiplying by 1.25, 1.33, 1.5, 1.66, or 2.0.

## Main Elements for Figures and Images

The <fig> and <image> elements may appear in concept, task, and reference topics. For graphic descriptions, use either <alt> or <longdescref>, both of which are associated with <image>.

### Purpose

Use <fig> to create a formal (titled) figure in concept and reference topics. To insert images that refer to UI features in a task or in a reference table, insert an <image>within <fig>. To create an informal (untitled) figure, use <image> instead of <fig>. Note that <fig> can be cross-referenced, but <image> cannot.

### Element Prototype

The following shows a figure with a mandatory title and an ID. Note that the long description is not included as text, but is contained in a separate topic that is cross-referenced.

```
<concept>
  <title></title>
  <shortdesc></shortdesc>
  <conbody>
    <p></p>
    <fig id="1234">
      <title>FIGURE_TITLE_HERE</title>
      <image href="GUID-00E19710-0225-4FCF-B622-962F59172533"
placement="break">IMAGE_APPEARS_HERE
        <longdescref href="GUID-AB354DDD-226A-4B2D-B5A1-
E6B98A22F553"/>
```

```
      </image>
    </fig>
  </conbody>
</concept>
```

### Main Elements

This table shows the most relevant elements for figures and images.

| Element | Child of | Mandatory ? | Description |
|---|---|---|---|
| <fig> | N/A | No | Displays a figure for a wide variety of content. Most commonly, the figure element contains an <image> element (a graphic or artwork), but it can also contain several kinds of text objects. A <fig>, but not an <image>, can be cross-referenced using the @id attribute. |
| <image> | <fig>, or used standalone | No | Use the <image> element to insert a graphic into a <fig>, or use it as an alternative of <fig>. The @href attribute points to the relative path location of the graphic file.<br>Set @expanse to "page" to get a wide figure. Set @placement to "break" to avoid placing the image inline.<br>For screenshots, use HyperSnap 5.x or later, and save as a JPG. Add a 1–pixel wide frame around the image. Change the horizontal and vertical resolutions of the image by multiplying by 1.25, 1.33, 1.5, 1.66, or 2.0. For example, if the image resolution is 96 dpi, change it to 120, 128, 144, 160, or 192, depending on how much scaling you want. The higher the number, the smaller the image. |
| <title> | <fig> | Yes | Provides an required title for a figure. Note that an <image> cannot contain a <title>. |
| <alt> | <image> | Only if <longdescref> not used | Provides optional alternative text for , similar to the FrameMaker AltText attribute. For a graphic description, use either <alt> or <longdescref>, both of which are associated with <image>.<br>Do not use the @alt attribute for alternate text. Use <alt> instead. |
| <longdescref> | <image> | Only if <alt> not used | Links the graphic that is being described to the topic object that contains its description. Graphic descriptions must be stored in separate Graphic Description topics. |

### Templates

There is only one graphic description template. It is located in System/Editor template/Topics. InfoDev recommends that you create new graphic descriptions using the template.

## Guidelines for Figures and Images

These guidelines provide both requirements (what you must do) and best practices (what it is recommended to do) when creating figures and images.

When creating images and figures, consider the following guidelines:

- Use graphics when you need to insert images to refer to UI features in a task or in a reference table.
- For both figures and graphics, you must always include an alternate text description to make the illustrations accessible.
- Use `<fig>` for a formal figure, that is, a figure that is titled and can be cross-referenced. Use `<image>` for an informal (untitled) graphic.

  Figure elements require a title and number, and should be used for most images. Graphic elements are informal images, which can be embedded within content units in document, such as in a procedure.
- For both figures and graphics, you must include an alternate text description to make the illustrations accessible.

  Use the `<alt>` element to add a brief description of navigation images such as buttons and arrows. For longer, more complex descriptions (such as those required for diagrams), create a separate topic, of the Graphic Description type, containing the text description and reference this topic using a `<longdescref>` element. The `<longdescref>` element is an empty element. Use its `@href` attribute to reference the topic containing the graphic description.
- To get a wide figure, set the `@expanse` attribute to “page”.
- To avoid placing the figure inline, set the `@placement` attribute to “break”.
- When setting `@href`, specify a relative location.
- Do not use the `@alt` attribute. Use `<alt>` instead.

### Guidelines for Writing Graphic Descriptions

When writing graphic descriptions, follow these guidelines:

- Create a new graphic description topic using the Graphic Description template.
- Use statements such as “at upper left is a...” or “the radio buttons on the left...” to help readers understand the layout of the picture.
- Do not use the word “graphic” because it is a keyword for most screen readers. Use “image,” “illustration,” or “figure” instead.
- Avoid including the image file name in the description. If you change the image file name, then you must update the description.

### Guidelines for Taking Screenshots

When taking screenshots for use in by an `<image>`, follow these guidelines:

- Use HyperSnap 5.5 or later as your capture tool.

- Add a 1–pixel wide frame around the image.
- Change the horizontal and vertical resolutions of the image by multiplying by 1.25, 1.33, 1.5, 1.66, or 2.0.
- Save the image as a JPG.

## Examples of Figures and Images

These examples show some of the ways that you can use `<fig>` and `<image>` elements.

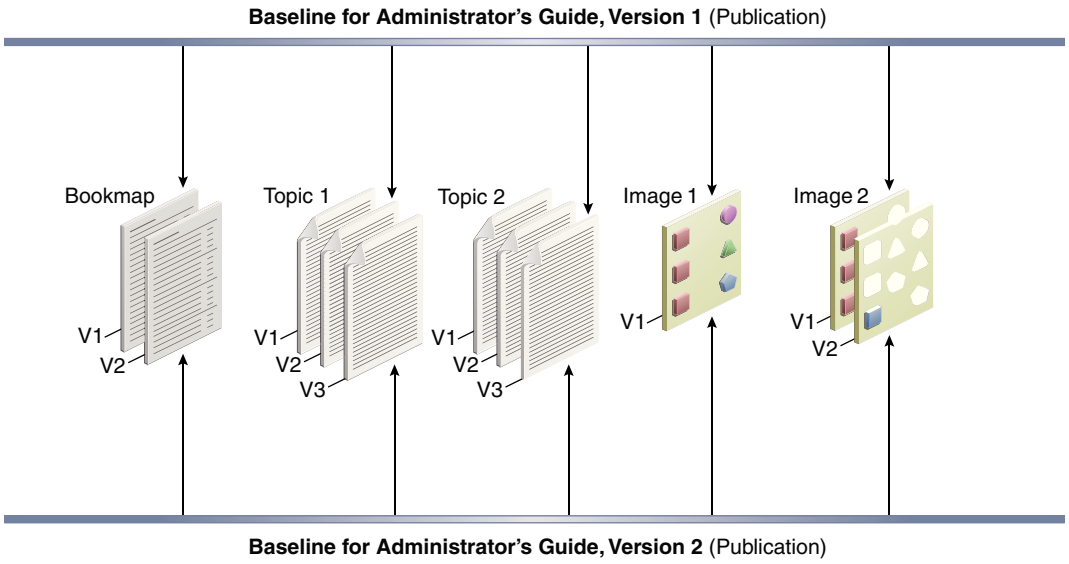
### Figure Elements

Note that in figures, the`@id` attribute and the mandatory `<title>` element, which causes the figure to be assigned a figure number in the output. `<longdescref>` is an empty element that cross-references a graphic description topic.

```
<fig id="fig-1414-809293B8">
  <title>Baseline for Administrator's Guide</title>
  <image href="GUID-BBCC053C-95BE-40A9-9C9F-5623E27C53DA"
placement="break">
    <longdescref href="GUID-5896583D-A981-4E26-B648-0EFE498D3C79"/>
  </image>
</fig>
```

The figure appears like this in the output:

***Figure 27-1 Baseline for Administrator's Guide***



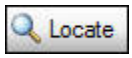

### Image Elements

Note that, when inserting an image (for example, to show a GUI button), the `<title>` element and `@id` attribute are absent. Because this is a simple GUI element and not a diagram or screenshot, instead of a `<longdescref>`, an `<alt>` element contains the alternate text.

```
<image align="left" href="GUID-3500E97A-2A0B-4A17-9573-
FC5A68D7DF43" placement="break">
```

```
    <alt>This image shows a Locate button. A magnifying glass icon
appears to the left of the word "Locate."</alt>
</image>
```

The image appears like this in the output:




# Guidelines for Index Entries

This section describes the standards for the <indexterm> element. These standards are specific to the InfoDev implementation of DITA. They are intended for all authors in InfoDev.

## About Index Entries

The DITA <indexterm> element marks a term and its location in the document to be included in the index. One index term generates one index entry. An index term with more than one occurrence generates one index entry with one reference per occurrence.

You may only place an <indexterm> within <prolog><metadata><keywords>. For example, if you wanted to index the top-level terms tasks and procedures, which would appear as siblings in the index, you would create the <prolog> as follows:

```
<prolog>
  <metadata>
    <keywords>
      <indexterm>tasks</indexterm>
      <indexterm>procedures</indexterm>
    </keywords>
  </metadata>
</prolog>
```

Achieve indentation accomplished by nesting <indexterm> elements. For example, to make the word tasks the primary index entry, and the word procedures the secondary (child) element underneath tasks, you would achieve it as follows:

```
<prolog>
  <metadata>
    <keywords>
      <indexterm>tasks
        <indexterm>procedures</indexterm>
      </indexterm>
    </keywords>
  </metadata>
</prolog>
```

For a *See* entry such as "*See* ASM," use the <index-see> element. For a *See Also* entry such as "*See Also* ASM," use the <index-see-also> element.

**See Also:**

Creating an Index
: A book index is generated automatically after the whole document is processed and its output is generated. Every **index term** that you mark in the body of the document produces an index entry in the document

index. Index entries can be **primary**, **secondary**, or **tertiary**. These categories determine the hierarchy of each term within the index list.

## Main Elements for Index Entries

The `<indexentry>`, `<index-see>`, and `<index-see-also>` elements may appear in any topic. The CMS generates an index for a document from these entries.

### Purpose

The `<indexentry>` element creates an index entry. DITA does not provide separate elements for primary and secondary (child) entries: instead, you nest `<indexentry>` elements to create hierarchy. For a *See* entry such as "*See* ASM," use the `<index-see>` element. For a *See Also* entry such as "*See Also* ASM," use the <index-see-also> element.

### Element Prototype

The following prolog shows a primary index entry `CDBs` with a secondary entry `about`. Note how the `<indexterm>` elements are nested. The prolog also contains a primary entry `multitenant container databases` with a "*See*" reference for `CDBs`.

```
<prolog>
  <metadata>
    <keywords>
      <indexterm>CDBs
        <indexterm>about</indexterm>
      </indexterm>
      <indexterm>multitenant container databases
        <index-see>CDBs</index-see>
      </indexterm>
   </keywords>
  </metadata>
</prolog>
```

### Main Elements

This table shows the most relevant elements for figures and images.

| Element | Child of | Mandatory? | Description |
|---|---|---|---|
| `<indexterm>` | | No | |
| `<index-see>` | | No | |
| `<index-see-also>` | | No | |

**See Also:**

Creating an Index

A book index is generated automatically after the whole document is processed and its output is generated. Every **index term** that you mark in the body of the document produces an index entry in the document

index. Index entries can be **primary**, **secondary**, or **tertiary**. These categories determine the hierarchy of each term within the index list.

## Guidelines for Index Entries

These guidelines explain what to do and what not to do in index entries, and provide examples.

The primary goal of index entries is to provide an alphabetical list of words and phrases with links that reference important information in the documentation. These guidelines explain how to achieve this goal:

- Only create index entries in the `<prolog>` element.
- Do not create index entries with more than two levels.

**See Also:**

Creating an Index

A book index is generated automatically after the whole document is processed and its output is generated. Every **index term** that you mark in the body of the document produces an index entry in the document index. Index entries can be **primary**, **secondary**, or **tertiary**. These categories determine the hierarchy of each term within the index list.

Indexing Guidelines in the Oracle Documentation Standards Oracle Style Guide

## Examples of Index Entries

These examples show different types of index entries.

***Example 27-10 Index Entry with One Level***

```
<prolog>
    <metadata>
      <keywords>
         <indexterm>Architect</indexterm>
      </keywords>
    </metadata>
</prolog>
```

***Example 27-11 Index Entry with Two Levels***

```
<prolog>
    <metadata>
      <keywords>
        <indexterm>Arbortext Editor
          <indexterm>Authoring Bridge</indexterm>
        </indexterm>
      </keywords>
    </metadata>
  </prolog>
```

***Example 27-12 Index Entry with a See Entry***

```
<prolog>
    <metadata>
      <keywords>
         <indexterm>SDL LiveContent Architect Repository
           <index-see>repository</index-see>
         </indexterm>
```

```
      </keywords>
    </metadata>
  </prolog>
```

**See Also:**

Creating an Index

A book index is generated automatically after the whole document is processed and its output is generated. Every **index term** that you mark in the body of the document produces an index entry in the document index. Index entries can be **primary**, **secondary**, or **tertiary**. These categories determine the hierarchy of each term within the index list.

# 28

# Metadata for Objects

SDL LiveContent Architect tracks and maintains metadata for each object in the repository. The metadata contains relevant information about each object. It is used to describe each object in the repository, to search for objects, and to track each object through its workflow.

**See Also:**

Object Types Managed by SDL LiveContent Architect
: All of the components that make up a document are objects in the SDL LiveContent Architect (Architect) repository.

Viewing an Object in Arbortext Editor
: You can view a topic, map, publication, or library in Arbortext Editor. Although viewing an object in Arbortext Editor limits your ability to modify its contents it is useful when you want to review the contents of an object that you don't have permissions to modify.

## Metadata for Publications

Metadata for publications aids in searching for publications and tracking a publication's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

***Table 28-1 Metadata Common to All Object Types***

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| General | Title | The title of the object If the object is a topic, then ensure that the title matches the title in the topic. | Mandatory |
| | In Folder | The folder path to the object in the repository | Mandatory |
| | Identifier | The global unique identifier (GUID) for the object | Mandatory |
| | Description | Text that describes the object | Optional |
| | Date Created | The date and time when the object was created | Mandatory |
| | Date Modified | The date and time when the object was last modified | Optional |

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| | | A modification is either the last workflow operation or the last revision change. | |
| Version | Version | Version number of the object | Mandatory |
| | Changes | Text that describes the reason for a new version<br>The text is removed automatically when a new version is created. The object owner enters new text to describe the reason for the new version. | Optional |

***Table 28-2 Metadata for Publications***

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| General | Publication type | The type of the publication, such as Administrator's Guide, Developer's Guide, Reference, and so on | Mandatory |
| Version | Baseline | The baseline used in the publication | Mandatory |
| | Working language | The language in which the publication is assembled | Mandatory |
| | Working resolution | The resolution used for assembling the publication | Mandatory |
| | Product name | Name of the major product or product line covered by the publication, such as Oracle Database, Database Machine, and so on | Optional |
| | Doc ID | The five digit document ID, such as ADMIN or CNCPT | Optional |
| | Release # | The product release for the publication | Optional |
| | Platform | The platform for the publication | Optional |

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| Workflow | Publication Owner | The name of the person who owns the publication | Optional |
| | Peer Reviewer | The name of the person responsible for peer reviewing the publication | Optional |
| | Peer Reviewed | Indicates whether the publication's peer review is complete | Optional |
| Front Matter | Copyright Year, First Year | First copyright year | Optional |
| | Copyright Year, Last Year | Last copyright year | Optional |

**See Also:**

Creating a Publication
You create a publication to assemble the content in a document for output.

# Metadata for Maps

Metadata for maps aids in searching for maps and tracking a map's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

***Table 28-3 Metadata Common to All Object Types***

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| General | Title | The title of the object<br>If the object is a topic, then ensure that the title matches the title in the topic. | Mandatory |
| | In Folder | The folder path to the object in the repository | Mandatory |
| | Identifier | The global unique identifier (GUID) for the object | Mandatory |
| | Description | Text that describes the object | Optional |
| | Date Created | The date and time when the object was created | Mandatory |
| | Date Modified | The date and time when the object was last modified<br>A modification is either the last workflow operation | Optional |

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| | | or the last revision change. | |
| Version | Version | Version number of the object | Mandatory |
| | Changes | Text that describes the reason for a new version<br>The text is removed automatically when a new version is created. The object owner enters new text to describe the reason for the new version. | Optional |

***Table 28-4 Metadata for Maps***

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| General | Map type | The type of the map, such as bookmap, feature map, submap, and so on | Mandatory |
| Version | Candidate for baseline | Used to complete a baseline with the "Candidate for baseline" option. | Optional |
| | Baseline label | Generated when the "Candidate for baseline" option is used | Mandatory (when the "Candidate for baseline" option is used) |
| | Working resolution | The resolution used for assembling the publication | Mandatory |
| | Product name | Name of the major product or product line covered by the map, such as Oracle Database, Database Machine, and so on | Optional |
| | Product Component | Name of the product component, such as Fast Recovery Area, Optimizer, and so on<br>The Product Component is logically dependent on the Product Name. | Optional |
| | Functional Area | The name of the major functional area, such as Administration, | Optional |

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| | | Installation and Upgrade, High Availability, Performance, and so on | |
| | Functional Subarea | The name of the minor functional area, such as Backup and Recovery, SQL*Plus, and so on The Functional Subarea is logically dependent on the Functional Area. | Optional |
| | Keywords | A comma-delimited list of keywords Keywords can improve search results when users search the repository. | Optional |
| | Feature # | A comma-delimited list of feature numbers that correspond with features in ST-project Enter feature numbers when the map covers a particular feature or features. Ensure that one or more feature numbers are entered for feature maps. | Optional |
| | Bug # | A comma-delimited list of bug numbers that correspond with bugs in the Bug database | Optional |
| | Release # | The product release for the publication | Optional |
| Workflow | Status | Indicator of the progress of the object through the workflow | Mandatory |
| | Author | The name of the author of the map | Mandatory |
| | Last modified by | Name of the person who last modified the map | Mandatory |
| | Comments | Informational text | Optional |
| | Functional Area Owner | The name of the functional area owner | Mandatory |

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| | | The functional area owner is the main person responsible for the documentation that covers a particular feature. | |
| | Architect | The name of the architect<br>The architect is the primary reviewer of the map. | Optional |
| | Publication Owner | The name of the person who owns the publication | Optional |
| | Content Group Member | The name of the content group member | Optional |

**See Also:**

Creating Maps

Use maps to logically organize objects, including other maps and topics, in a publication.

# Metadata for Libraries

Metadata for libraries aids in searching for libraries and tracking a library's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

***Table 28-5 Metadata Common to All Object Types***

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| General | Title | The title of the object<br>If the object is a topic, then ensure that the title matches the title in the topic. | Mandatory |
| | In Folder | The folder path to the object in the repository | Mandatory |
| | Identifier | The global unique identifier (GUID) for the object | Mandatory |
| | Description | Text that describes the object | Optional |
| | Date Created | The date and time when the object was created | Mandatory |
| | Date Modified | The date and time when the object was last modified | Optional |

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| | | A modification is either the last workflow operation or the last revision change. | |
| Version | Version | Version number of the object | Mandatory |
| | Changes | Text that describes the reason for a new version<br>The text is removed automatically when a new version is created. The object owner enters new text to describe the reason for the new version. | Optional |

***Table 28-6 Metadata for Libraries***

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| General | Library type | The type of the library, such as text variables, book links, notes, and so on | Mandatory |
| Version | Candidate for baseline | Used to complete a baseline with the "Candidate for baseline" option. | Optional |
| | Baseline label | Generated when the "Candidate for baseline" option is used | Mandatory (when the "Candidate for baseline" option is used) |
| | Working resolution | The resolution used for assembling the publication | Mandatory |
| | Product name | Name of the major product or product line covered by the library, such as Oracle Database, Database Machine, and so on | Optional |
| | Product Component | Name of the product component, such as Fast Recovery Area, Optimizer, and so on<br>The Product Component is logically dependent on the Product Name. | Optional |

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| | Functional Area | The name of the major functional area, such as Administration, Installation and Upgrade, High Availability, Performance, and so on | Optional |
| | Functional Subarea | The name of the minor functional area, such as Backup and Recovery, SQL*Plus, and so on<br>The Functional Subarea is logically dependent on the Functional Area. | Optional |
| | Keywords | A comma-delimited list of keywords<br>Keywords can improve search results when users search the repository. | Optional |
| | Feature # | A comma-delimited list of feature numbers that correspond with features in ST-project<br>Enter feature numbers when the library is used to cover a particular feature or features. | Optional |
| | Bug # | A comma-delimited list of bug numbers that correspond with bugs in the Bug database<br>Enter bug numbers when the library is used to correct a particular documentation bug or bugs. | Optional |
| | Release # | The product release for the publication | Optional |
| Workflow | Status | Indicator of the progress of the object through the workflow | Mandatory |
| | Author | The name of the author of the library | Mandatory |

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| | Last modified by | Name of the person who last modified the library | Mandatory |
| | Comments | Informational text | Optional |
| | Architect | The name of the architect<br>The architect is the primary reviewer of the library. | Optional |
| | Publication Owner | The name of the person who owns the publication | Optional |
| | Content Group Member | The name of the content group member | Optional |

**See Also:**

Creating a Variable Library
: Variables are defined by DITA elements that contain a value; each variable is identified by an unique`varid` attribute. Library objects are then used to store variable definitions and associate them to a publication.

Creating a Conref Library
: You can organize DITA elements that are conref targets into an Architect library. You can have any number of conref libraries. Unlike variable libraries, you do not need to add conref libraries to the resources folder of your publication.

# Metadata for Topics

Metadata for maps aids in searching for topics and tracking a topic's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

***Table 28-7 Metadata Common to All Object Types***

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| General | Title | The title of the object<br>If the object is a topic, then ensure that the title matches the title in the topic. | Mandatory |
| | In Folder | The folder path to the object in the repository | Mandatory |
| | Identifier | The global unique identifier (GUID) for the object | Mandatory |
| | Description | Text that describes the object | Optional |

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| | Date Created | The date and time when the object was created | Mandatory |
| | Date Modified | The date and time when the object was last modified<br>A modification is either the last workflow operation or the last revision change. | Optional |
| Version | Version | Version number of the object | Mandatory |
| | Changes | Text that describes the reason for a new version<br>The text is removed automatically when a new version is created. The object owner enters new text to describe the reason for the new version. | Optional |

***Table 28-8 Metadata for Topics***

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| General | Topic type | The type of the topic, such as concept, task, reference, and so on | Mandatory |
| | Output file | Name of the XHTML output file for the topic<br>Enter a name when you want the output file to have a different name than the automatically generated name. | Optional |
| Version | Candidate for baseline | Used to complete a baseline with the "Candidate for baseline" option. | Optional |
| | Baseline label | Generated when the "Candidate for baseline" option is used | Mandatory (when the "Candidate for baseline" option is used) |
| | Product name | Name of the major product or product line covered by the publication, such as Oracle Database, | Optional |

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| | | Database Machine, and so on | |
| | Product Component | Name of the product component, such as Fast Recovery Area, Optimizer, and so on<br>The Product Component is logically dependent on the Product Name. | Optional |
| | Functional Area | The name of the major functional area, such as Administration, Installation and Upgrade, High Availability, Performance, and so on | Optional |
| | Functional Subarea | The name of the minor functional area, such as Backup and Recovery, SQL*Plus, and so on<br>The Functional Subarea is logically dependent on the Functional Area. | Optional |
| | Keywords | A comma-delimited list of keywords<br>Keywords can improve search results when users search the repository. | Optional |
| | Feature # | A comma-delimited list of feature numbers that correspond with features in ST-project<br>Enter feature numbers when the topic covers a particular feature or features. | Optional |
| | Bug # | A comma-delimited list of bug numbers that correspond with bugs in the Bug database<br>Enter bug numbers when the topic corrects a particular documentation bug or bugs. | Optional |

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| Workflow | Status | Indicator of the progress of the object through the workflow | Mandatory |
| | Author | The name of the author of the topic | Mandatory |
| | Last modified by | Name of the person who last modified the topic | Mandatory |
| | Comments | Informational text | Optional |
| | Functional Area Owner | The name of the functional area owner<br>The functional area owner is the main person responsible for the documentation that covers a particular feature. | Mandatory |
| | Architect | The name of the architect<br>The architect is the primary reviewer of the topic. | Optional |
| | Editor | The name editor<br>The editor is the person in on the editing team who is responsible for editing the topic. | Optional |
| | Manager | The name of the manager<br>The manager is the manager responsible for the topic. Typically, this field is set to the manager of the author. | Optional |
| Deprecated/ Desupported | Deprecated In | Release in which the topic is deprecated<br>This field is populated when the topic covers a feature that has been deprecated. | Optional |
| | Desupported In | Release in which the topic is desupported<br>This field is populated when the topic covers a feature that has been desupported. | Optional |

**See Also:**

Creating Topics from Publication Manager
: Create a topic from the Browse Repository dialog box in Publication Manager.

Creating Topics from Arbortext Editor
: Create a topic from the **SDL LiveContent** menu in Arbortext Editor as an alternative to the Browse Repository dialog box in Publication Manager.

## Metadata for Graphics

Metadata for maps aids in searching for graphics and tracking a graphic's progress. Some metadata fields apply to all objects types, and some metadata fields only apply to some object types.

***Table 28-9 Metadata Common to All Object Types***

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| General | Title | The title of the object<br>If the object is a topic, then ensure that the title matches the title in the topic. | Mandatory |
| | In Folder | The folder path to the object in the repository | Mandatory |
| | Identifier | The global unique identifier (GUID) for the object | Mandatory |
| | Description | Text that describes the object | Optional |
| | Date Created | The date and time when the object was created | Mandatory |
| | Date Modified | The date and time when the object was last modified<br>A modification is either the last workflow operation or the last revision change. | Optional |
| Version | Version | Version number of the object | Mandatory |
| | Changes | Text that describes the reason for a new version<br>The text is removed automatically when a new version is created. The object owner enters new text to describe the | Optional |

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| | | reason for the new version. | |

***Table 28-10 Metadata for Graphics***

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| General | Illustration type | The type of the graphic, such as diagram, screenshot, icon, and so on | Mandatory |
| Version | Candidate for baseline | Used to complete a baseline with the "Candidate for baseline" option. | Optional |
| | Baseline label | Generated when the "Candidate for baseline" option is used | Mandatory (when the "Candidate for baseline" option is used) |
| | Product name | Name of the major product or product line covered by the graphic, such as Oracle Database, Database Machine, and so on | Optional |
| | Product Component | Name of the product component, such as Fast Recovery Area, Optimizer, and so on The Product Component is logically dependent on the Product Name. | Optional |
| | Functional Area | The name of the major functional area, such as Administration, Installation and Upgrade, High Availability, Performance, and so on | Optional |
| | Functional Subarea | The name of the minor functional area, such as Backup and Recovery, SQL*Plus, and so on The Functional Subarea is logically dependent on the Functional Area. | Optional |
| | Keywords | A comma-delimited list of keywords | Optional |

| Tab | Field | Description | Mandatory/Optional |
|---|---|---|---|
| | | Keywords can improve search results when users search the repository. | |
| | Feature # | A comma-delimited list of feature numbers that correspond with features in ST-project<br>Enter feature numbers when the graphic covers a particular feature or features. | Optional |
| | Bug # | A comma-delimited list of bug numbers that correspond with bugs in the Bug database<br>Enter bug numbers when the graphic corrects a particular documentation bug or bugs. | Optional |
| | Platform | The platform for the image | Optional |
| Workflow | Status | Indicator of the progress of the object through the workflow | Mandatory |
| | Author | The name of the author who uses the graphic in on or more topics | Mandatory |
| | Last modified by | Name of the person who last modified the graphic | Mandatory |
| | Resolution | The resolution of the graphic | Mandatory |
| | Comments | Informational text | Optional |
| | Graphic Artist | The name of the graphic artist who owns the graphic<br>When the graphic is a screen shot, then typically the author and the graphic artist are the same person. | Mandatory |

**See Also:**

Inserting a Figure
: A figure contains an image, a title, and a figure number, which enables you to refer to that figure from other parts of a document. Figures are also listed by figure number and title in the table of figures, which is located after the table of contents.

Inserting a Graphic
: A graphic is an informal image. An informal image does not have a title or a figure number, so you cannot refer to an informal figure from another location in the document. Graphics are not included in the list of figures.

# 29

# Accessibility in DITA

Tables and figures, either formal or informal, must comply with accessibility requirements. Always use the correct DITA elements to make descriptive information available to all readers, including those with disabilities.

## Formal Tables Accessibility Checklist

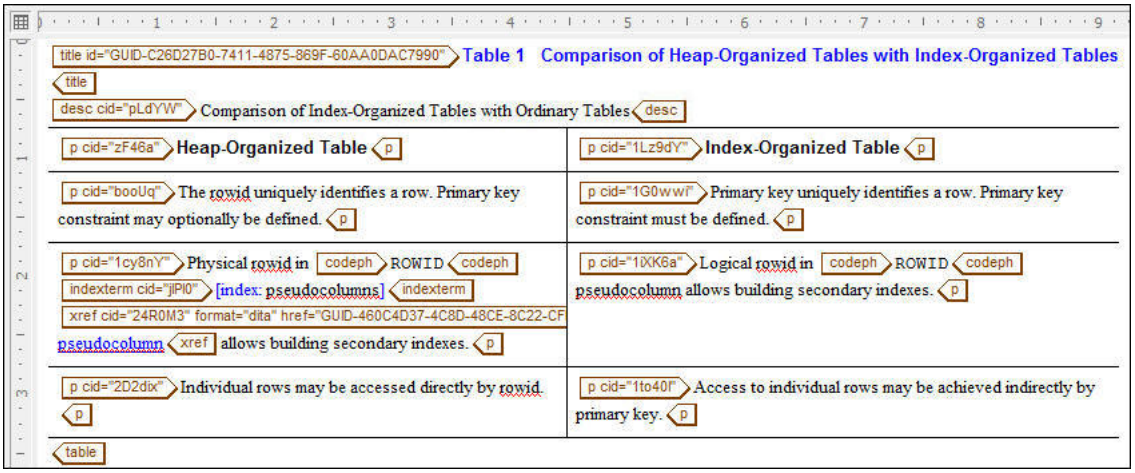
Formal tables comply with accessibility requirements by using the desc element. It includes a short description of the contents and organization of the table.

### Required Accessibility Elements for Formal Tables

The following table indicates the element or attributes that are required to make formal tables accessible:

| Element | Purpose |
|---|---|
| `desc` | Includes a short description about the organization of the table (rows and columns) and its contents. Depending on the length of the description you may use a single ph to enclose the contents or different p elements. |

***Figure 29-1 Accessibility Elements in a Formal Table***



# Informal Tables Accessibility Checklist

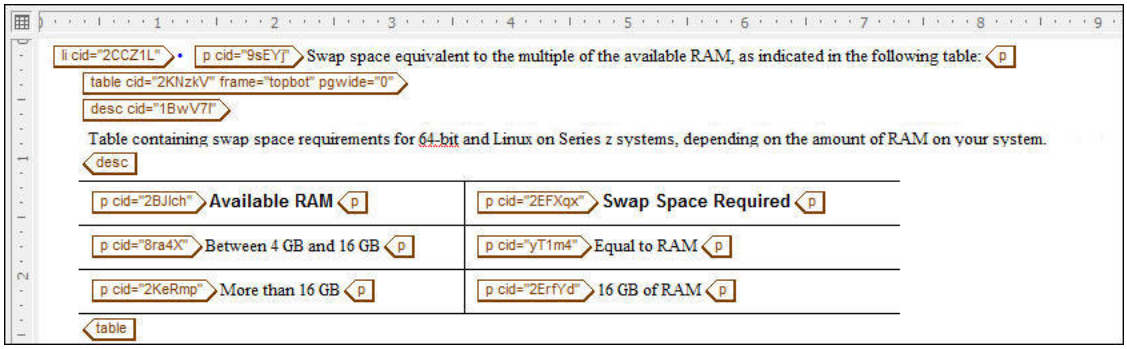
Informal tables comply with accessibility requirements by using the desc element.

### Required Accessibility Elements for Informal Tables

The following table indicates the elements or attributes that are required to make informal tables accessible

| Element | Purpose |
|---|---|
| `desc` | Includes a short description about the organization of the table (rows and columns) and its contents. Depending on the length of the description you may use a single ph to enclose the contents or different p elements. |

***Figure 29-2 Accessibility Elements in an Informal Table***




# Figures Accessibility Checklist

In DITA, figures comply with accessibility requirements by using the `alt` or `longdesc`tag within their element structure. The element must provide a description in such a way that a user can understand the layout of an image, the flow of a diagram, or the action that is illustrated, without actually viewing the image.

### Required Accessibility Elements for Figures

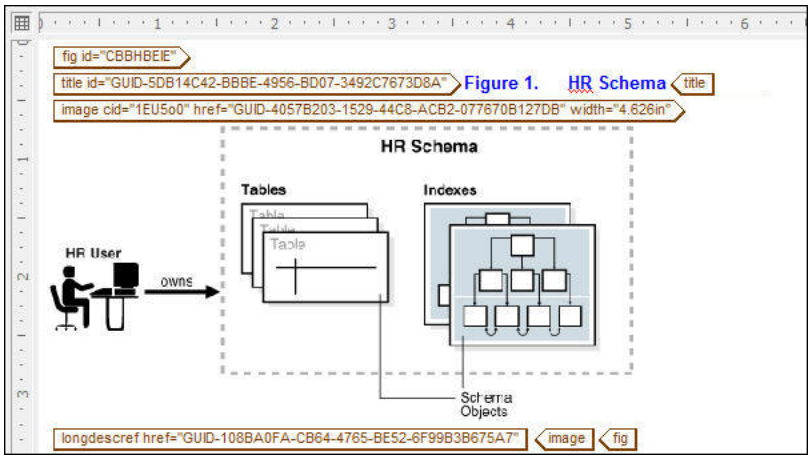
The following table indicates the element or attributes that are required to make formal figures accessible:

| Element | Purpose |
|---|---|
| `alt` | Contains an alternate text description equivalent to the information in the image. Use this element when describing simple graphics such as navigational arrows or buttons. |
| `longdescref` | References a Graphic Description topic that contains the description of the image through its `href` attribute. Use this element when describing complex graphics, such as screenshots or diagrams. |

**Note:** Follow these guidelines as you write your alternate image description:

- Use statements such as “at upper left is a...” or “the navigation tree on the left...” to help readers understand the layout of the picture.
- Do not use the word graphic because it is a keyword for most screen readers. Use image, illustration, or figure instead.
- Avoid including the image file name in the description.. Otherwise, if you change the image file name, then you must update the description.

***Figure 29-3 Accessibility Elements in a Figure***




# Graphic Accessibility Checklist

In DITA, graphics comply with accessibility requirements by using the `alt`or `longdesrev` tag within their element structure. The element must provide a description in a way that users can understand the layout of an image, the flow of a diagram, or the action that is illustrated, without actually viewing the image.

### Required Accessibility Elements for Graphics

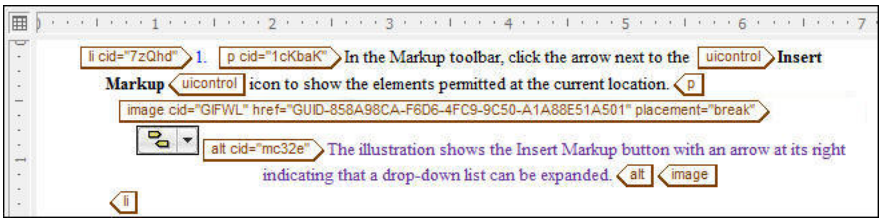
The following table indicates the element or attributes that are required to make formal tables accessible:

| Element | Purpose |
|---|---|
| `alt` | Contains an alternate text description equivalent to the information in the image. Use this element when describing simple graphics such as navigational arrows or buttons. |
| `longdescref` | References a Graphic Description topic, that contains the description of the image, that contains the description of the image, through its`href` attribute. Use this element when describing complex graphics, such as screenshots or diagrams. |

**Note:** Follow these guidelines as you write your alternate image description:

- Use statements such as “at upper left is a...” or “the navigation tree on the left...” to help readers understand the layout of the picture.
- Do not use the word graphic because it is a keyword for most screen readers. Use image, illustration, or figure instead.
- Avoid including the image file name in the description. Otherwise, if you change the image file name, then you must update the description.

***Figure 29-4 Accessibility Elements in a Graphic***



# Glossary

### Arbortext Editor

The selected authoring tool chosen InfoDev to create and edit documents in DITA and other XML languages. Arbortext Editor has an interface that facilitates creating structured documents, and fully supports topic-based writing.

### Architect

(SDL LiveContent Architect), a component content management system that enables to collaborate in the tasks of planning, authoring, reviewing, translating and publishing content. It is specially designed to support DITA. The main interface of the system consists of Publication Manager (also called Publication Explorer), SDL Authoring Bridge, and Condition Manager.

### authoring bridge

A part of the SDL LiveContent Architect interface. The authoring bridge is integrated to Arbortext Editor as a regular menu, it is used to access the SDL repository and use the different features and tools included in Architect.

### baseline

An object in the SDL LiveContent Architect repository that lists the specific versions of the topics, graphics, and variable libraries used in a publication. The baseline object allows you to view and access the versions of the objects used in the publication. The baseline of the publication is automatically frozen when you release an output release candidate, or when you manually freeze a baseline. This means that the baseline and thus the content of the publication can no longer be modified and is preserved as the historical version of a released product.

### batch server

In the SDL Live Content Architect system architecture, a server that performs background tasks such as publishing output.

### branching

An object in the SDL LiveContent Architect repository that lists the specific versions of the topics, graphics, and variable libraries used in a publication. The baseline object allows you to view and access the versions of the objects used in the publication. The baseline of the publication is automatically frozen when you release an output release candidate, or when you manually freeze a baseline. This means that the baseline and thus the content of the publication can no longer be modified and is preserved as the historical version of a released product.

### Browse Repository dialog box

An essential dialog of the SDL LiveContent architect interface. The Browse Repository dialog box is accessed from the Authoring Bridge and Publication Manager. It enables users to navigate the repository using three different views: the repository tab, the search tab, and the inbox tab.

### component

A part of a document or publication. In a component-oriented Content Management System, such as SDL LiveContent Architect, the term is synonymous with an object (topic, maps, graphics, etc.).

### condition expression

Combination of a condition name and a condition value. They can be set as an attribute to an XML element. Which XML elements support this attribute depends on the ddD that is used.

### conref

A reference to another object (topic, map, graphic, etc.), or element within an object, stored in the repository. A conref enables writers to easily reuse chunks of content.

### content team

A group of members (writers, architects, editors, functional area owners, and project managers) that works together to determine the topics that must be written to document a specific product feature.

### DITA

Darwin Information Typing Architecture - Provides a way for authors and architects to create collections of typed topics that can be easily assembled into various delivery contexts. Topic specialization is the process by which authors and architects can define topic types, while maintaining compatibility with existing style sheets, transformations, and processes. New topic types are defined as an extension, or delta, relative to an existing topic type, thereby reducing the work necessary to define and maintain the new type. DITA can be compared with Information Mapping.

### DocBuilder

A tool that validates the XML files associated with a publication and creates XHTML and PDF files from those XML files. It can also send the XHTML files to Oracle Review and create an archivable ZIP file to send to DocArch.

### feature map

On organized list of topics that are required to fully document a product feature. Feature maps are objects in the repository created using the **Custom Map** template.

### functional area

One of a group of a product's functions. For example, these are examples of functional areas for the Oracle Database: administration, application development, performance, and security.

### functional subarea

One of a group of a functional area's functions. A functional subarea is logically dependent on its functional area. For example, "database tuning" is a functional

subarea for the functional area “performance,” and ”SQL Developer” is a functional subarea for the functional area “application development.”

**GUID**

Globally Unique Identifier - Unique 128-bit number that is generated by theWindows OS or aWindows application to identify a particular component, XML element, application, file, database entry, user, and so on.

**inbox**

A Publication Explorer’s tab that contains objects assigned to a user using the defined workflos based on a user’s role.

**library**

A generic topic object used to store a set or related variable definitions.

**map**

DITA uses map to refer to a collection of modules. A map equates to a Master document in SDL LiveContent Architect. The map defines the topics in a collection or assembly, the order and other relationship between the topics. Maps do not contain any content of their own.

**markup**

**master document**

XML file used to combine a number of modular objects (procedures, processes, concepts, structures, references…) together in one document. It defines the modules in a collection the order and other relationship between the modules. It presents the components in a logical way, organizing them into parts, chapters and sections. As such, it is an XML map and can be compared with a table of contents (TOC).

**metadata**

Metadata is data about data such as the creation time-date stamp and name of the creator or author. Inside SDL LiveContent Architect additional information about objects is stored as metadata. SDL LiveContent Architect uses metadata extensively for its core functionality. Other examples are language, version number, status, modification date, and so on.

**module**

XML file that contains textual content and possibly references illustrations. It is an independent, logical information entity which can be created, stored managed, distributed and reused separately. They are referenced in master documents to build complete publications. In Information Mapping (IMAP), a module is a compilation of blocks following the IMAP methodology.

**object**

A single element stored in the repository. In SDL LiveContent Architect object types include: module, graphic, master document, publication, library, template, and folder.

**product**

The main software or tool that is the topic of a publication.

### publication

A publication is a CCMS specific object that defines a document in order to be published. It does so by associating a master document (ditamap), a baseline, the publication context (variables and conditions), and several output format objects.

### publication context

Context that defines which conditional sections should be published. It provides all conditions that are used by the conditional publishing engine for evaluating conditions set inside XML documents. Typically, a product configurator or a CRM module provides the delivery context

### Publication Manager

SDL LiveContent Architect client software application that is used to manager and compose a final publication. It allows editing of the master document associated with a publication, selection of versions of the objects that comprise the publication, and managing the baseline. You can access and check out objects, search, publish, apply conditions, and access the repository from Publication Manager.

### release

The public or private distribution of a new or upgraded version of a product.

### revision

Intermediary version of an object that is stored in the repository without creating a new version number. It is a snapshot underneath an existing version to which an author can return. All revisions under a certain version number have the same metadata. They are used for saving intermediate changes.

### SDL LiveContent Architect web client

A web client that provides users with access to the repository and their inbox using an internet browser. For administrators, it provides access to administration tools for configuring users, workflow, and permissions, and for managing, and troubleshooting the system

### topic

DITA term for a SDL LiveContent Architect module.

### variable

An uniquely identified DITA element defined in a variable library. A variable can be referenced from a topic where the definition of the variable (contents) is inserted when the document output is generated.

### version

A named revision of an object that is maintained and/or distributed in parallel to other versions. The revision was important enough to create a separate version of it.

### workflow

An integrated mechanism that enables authors to send documents from user to user and task to task, for example: for approval, review, translation, and so on. Through a number of configurable inboxes, the content objects are sent to the person in charge of the next step in the production process. Status transitions ensure that the workflow is followed in the correct and approved order. The workflow is fully customizable to comply with existing workflows.

**writer**

The person who is assigned ownership of a particular topic.

**XML**

Extensible Markup Language - Markup language (much like HTML) that was designed to describe data. No tags are predefined. It uses a DTD or Schema to describe the data.

# Index

## A



## B



## C

## D



## E



## F



## G



## I



## L

## M



## N



## O



## P

## R



## S



## T



## V



## W



## X