# TriTeal Enterprise Desktop (TED™) 4.0

# Help System Author's and Programmer's Guide

TriTeal™

Printed in USA

Part No. TED006004000

**Printing History**
July 1995 ... First Edition
Printed in the U.S.A

**Sales Information and Technical Support**

TriTeal Corporation
2011 Palomar Airport Road, Suite 200
Carlsbad, CA 92009
USA

Tel: (619) 930-2077
Fax: (619) 930-2081
Support E-Mail: support@triteal.com
Sales E-Mail: info@triteal.com
World Wide Web: http://www.triteal.com/

Please
Recycle

# Contents

# Figures

# Tables

# Preface

This manual describes how to develop online help for TriTeal Enterprise Desktop application software. It covers how to create help topics and how to integrate online help into an OSF/Motif™ application.

## Who Should Use This Book

The audience for this book includes:

- Authors who design, create, and view online help information.
- Developers who want to create software applications that provide a fully integrated help facility.

## How This Book Is Organized

This book has four parts. Part 1 describes the collaborative role that authors and developers undertake to design application help. Part 2 provides information for authors organizing and writing online help. Part 3 describes the Help System application programmer's toolkit. Part 4 contains information for both authors and programmers about preparing online help for different language environments.

This book includes these chapters:

Part 1— Introduction

**Chapter 1, "Introducing the Help System,"** provides an overview of authors' and developers' collaborative role in producing online help.

Part 2— The Author's Job

**Chapter 2, "Organizing and Writing a Help Volume,"** describes the components that make up a help volume.

**Chapter 3, "Writing a Help Topic,"** introduces the Help System markup language and gives examples of elements used to format different types of information. It describes how to include graphics and create hyperlinks.

**Chapter 4, "Processing and Displaying a Help Volume,"** describes how to process a marked-up file (or files) to generate a single run-time file for online viewing.

**Chapter 5, "HelpTag Markup Reference,"** lists in alphabetical order the HelpTag markup language elements, with an example of each element.

**Chapter 6, "Summary of Special Character Entities,"** provides a list of characters and associated entity names that can be used to insert special characters into help topic text.

**Chapter 7, "Command Summary,"** summarizes how to process and view a help volume by entering commands in a terminal emulator window.

**Chapter 8, "Reading the HelpTag Document Type Definition,"** describes the HelpTag DTD and how to use it to create fully compliant Standard Generalized Markup Language (SGML) help files.

Part 3— The Programmer's Job

**Chapter 9, "Creating and Managing Help Dialog Boxes,"** introduces the Help Dialog widgets and explains how to use them.

**Chapter 10, "Responding to Help Requests,"** explains how an application provides entry points to access different types of help.

**Chapter 11, "Handling Events in Help Dialogs,"** shows how an application can use a callback structure to handle hyperlink events.

**Chapter 12, "Providing Help on Help,"** describes how an application can provide a help module that tells users how to use the Help System.

**Chapter 13, "Preparing an Installation Package,"** covers what to include in an installation package to supply online help with an application.

Part 4— Internationalization

**Chapter 14, "Native Language Support,"** identifies language-dependent files used by the Help System.

**Glossary** is a list of words and phrases found in this book and their definitions.

## Related Books

Related TriTeal Enterprise Desktop books that you may find helpful are:

- *Advanced User's and System Administrator's Guide*
- *Internationalization Programmer's Guide*
- *Style Guide and Certification Checklist*
- *User's Guide*

For a technical description of Standard Generalized Markup Language (SGML), refer to:

- *The SGML Handbook* by Charles F. Goldfarb, Oxford University Press (ISBN 0-19-853737-9).

## What Typographic Changes and Symbols Mean

The following table describes the type changes and symbols used in this book.

*Table P-1* Typographic Conventions

| Typeface or Symbol | Meaning | Example |
|---|---|---|
| `AaBbCc123` | The names of commands, files, and directories; onscreen computer output | Edit your `.login` file.<br>Use `ls -a` to list all files.<br>`system% You have mail.` |
| *AaBbCc123* | Command-line placeholder: replace with a real name or value | To delete a file, type `rm` *filename.* |
| *AaBbCc123* | Book titles, new words or terms, or words to be emphasized | Read Chapter 6 in the *User's Guide*.<br>These are called *class* options.<br>You *must* be root to do this. |

# Part 1 — Introduction

# Introducing the Help System 1

This chapter introduces the Help System and briefly describes the user interface. It shows how help information is organized, outlines how to create and process help modules, and discusses the collaborative role of authors and developers in the design and creation of application help.



## Introduction to the Help System

The Help System provides a complete set of tools to develop online help for application software. It enables authors to write online help that includes graphics and text formatting, hyperlinks, and communication with the application.

The Help System also provides a programmer's toolkit for integrating online help into an application. The Help System application program interface supplies two specialized help dialogs and supporting routines that are used to display, navigate, search, and print online help modules.

### Developer's Toolkit

The Help System Developer's Toolkit contains tools to write, process, and view online help and contains an application programming library.

#### *For Authors*

- *HelpTag markup language*—a set of tags used in text files to mark organization and content of your online help.
- *HelpTag software*—a set of software tools for converting the HelpTag files you write into run-time help files.
- *Helpview application*—a viewer program for displaying your online help so you can read and interact with it just as your audience will.

Refer to Chapter 2, "Organizing and Writing a Help Volume," to learn more about creating and processing online help.

#### *For Application Developers*

- *DtHelp programming library*—an application program interface (API) for integrating help windows into your application.
- *A sample program*—a simple example that shows how to integrate the Help System into an OSF/Motif application (see page 21).

Chapters 9 through 13 discuss the application programming library.

## Overview of Online Help

It's virtually impossible—and certainly impractical—for anyone to learn and remember *everything* there is to know about the computer hardware and software they use to do their job. Nearly every computer user needs help at one time or another.

Online help, unlike a printed manual, has the power of the computer at its disposal. Most importantly, this power makes it possible to adapt the information to the user's current "context." *Context-sensitive* help provides just enough help to get the user back on task. In developing your online help, remember that users need different types of help at different times. By anticipating users' questions, you can design your application help to respond in a logical and intuitive manner.

## Help Information Model

There are two general styles of online help:

- *Application help*, whose primary role is to be an integrated part of an OSF/Motif application.
- *Standalone help*, whose primary role is to provide online access to task, reference, or tutorial information, independent of any application software.

If you are developing online help for an application, you may choose to organize the information exclusively for access within the application. Or, you may design the information such that it can be browsed without the application present, as in standalone help.

### Part of the Application

Help promotes a high degree of integration between the application and its online help. From the user's perspective, the help is part of the application. This approach minimizes the perceived "distance" away from the application that the user must travel to get help.

Staying close to the application makes users more comfortable with online help and gets them back on task as quickly as possible.

### Types of Help

Online help can be divided into three general categories:

- *Automatic help*—The application determines when help is needed and what to present. This is sometimes called system-initiated help.

- *Semiautomatic help*—The user decides when help is needed, but the system determines what to present. Semiautomatic help is initiated by a user's gesture or request for help, such as pressing F1. The system's response is called context-sensitive help because it considers the user's current context in deciding what information to display.
- *Manual help*—The user requests specific information, such as from a Help menu.

## How Users Get Help

A user can request help in several ways. Most applications provide a Help menu and Help key as well as Help buttons in dialog boxes.

### Help Key

Within most applications, the primary way for a user to request help is by pressing the *help key*. In recent years, the F1 function key has become a defacto standard help key for many workstation and personal computer products.

The *Style Guide and Certification Checklist* recommends the use of F1 as the help key, and the OSF/Motif programmer's toolkit even provides some built-in behavior to make it easier to implement the help key in OSF/Motif applications.

Some computers provide a Help key on the keyboard.

### Help Menu

The Help menu is a common way to provide access to help information. OSF/Motif applications provide a Help menu, which is right-justified in the menu bar. The *Style Guide and Certification Checklist* makes recommendations regarding the commands contained in a Help menu.

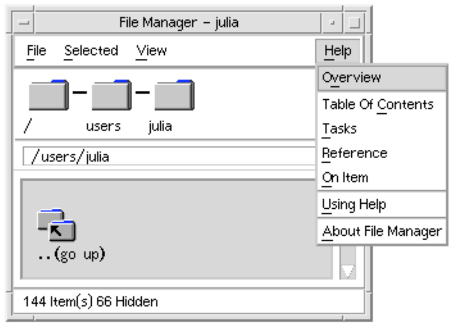



*Figure 1-1* Application Help menu

### Help Buttons

Many dialog boxes also provide a Help button to get help on the dialog. The *Style Guide and Certification Checklist* recommends that choosing the Help button in a dialog box be equivalent to pressing the Help key while using that dialog. Exceptions should be made for complex dialogs, where help on individual controls within the dialog box is appropriate.

## Help User Interface

This section is an overview of the graphical interface provided by the Help System. For a detailed description of Help features and capabilities, refer to the *User's Guide*; or, to view the corresponding online help, you can open the desktop Front Panel Help Viewer (see "To Display the browser Volume" on page 99). Then choose, TriTeal Enterprise Desktop and Desktop Help System.

While using an application, a user can request help by pressing the Help key or by selecting the application's Help menu. In addition, applications integrating the Help System can be installed so that their respective help modules are accessible from the desktop Help Viewer. This enables a user to browse help information supplied by different applications without having to run each application.

## Help Windows

When a user requests help, the Help System displays a help window. There are two types of help windows: general help and quick help. A general help window has a menu bar, topic tree, and a topic display area. The topic tree lists help topics that a user can choose. The lower portion of the window—the topic display area—displays the selected topic.

A quick help window is a streamlined help window. It has only a topic display area and one or more dialog buttons. Quick help windows are often used for short, self-contained information such as a definition.

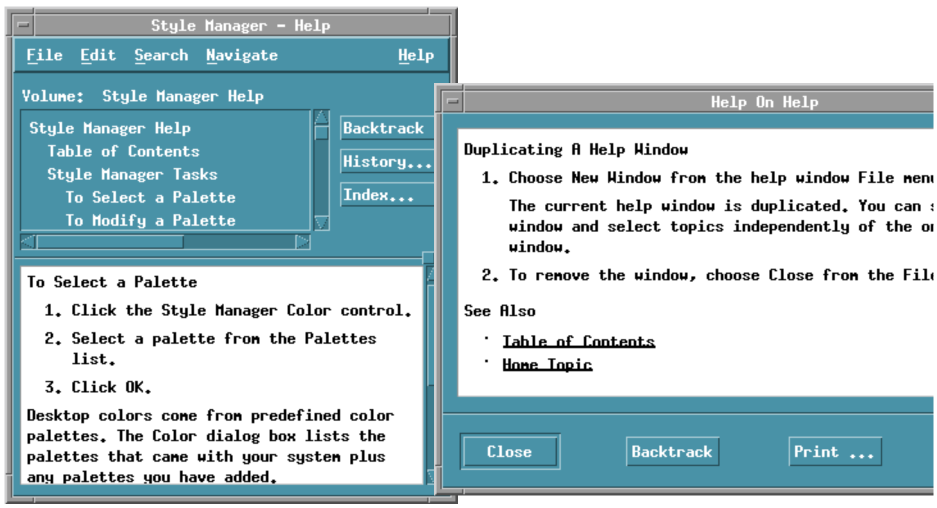



*Figure 1-2* General help and quick help window

## Hyperlinks

Help topics often contain hyperlinks that "jump" to related help information. Both text and graphics can be used as hyperlinks. Figure 1-3 on page 9 shows formatting styles used to identify hyperlinks.

Solid or dashed underscores identify words or phrases that are hyperlinks. The solid underscore, or standard hyperlink, is most common. When the hyperlink is selected, the related topic is displayed. An author designates whether the hyperlink topic is displayed in the current help window or a new window. The dashed underscore represents a definition link. When selected, the related topic is displayed in a quick help window. A gray, open-corner box (dashed or solid line) designates a graphic hyperlink.

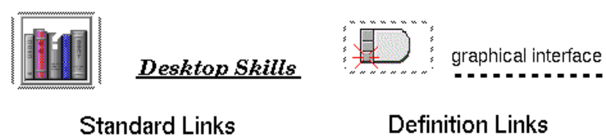



*Figure 1-3* Formats for graphic and text hyperlinks

## Help Navigation

The topic tree shown in Figure 1-4 is an outline of topics in the current help volume. The first topic at the top of the list is the *home topic*, or beginning of the help volume. An arrow (→) points to the current topic and shows the user's location in the help volume.

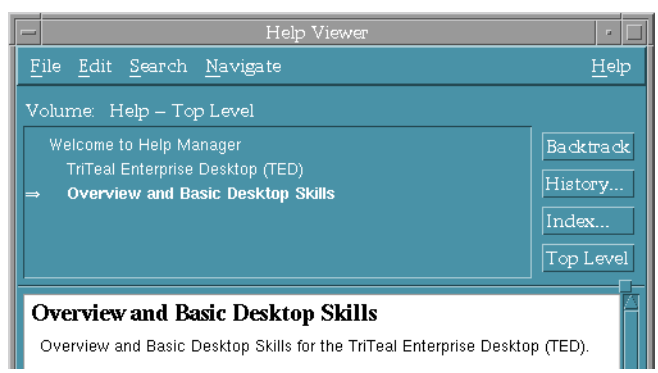



*Figure 1-4* Topic tree in a general help dialog box

To display a help topic, a user selects a title in the topic tree or a hyperlink within the topic display area. The user can browse the outline of topics by scrolling the list and then select any topic. Navigation commands enable the user to return to previous topics or to the beginning of the help volume.

### Help Navigation Buttons

The general help dialog includes three dialog buttons: Backtrack, History, and Index. These features are also available as menu selections.

- *Backtrack* — returns to the previous topic. To retrace topics visited, press Backtrack repeatedly until the desired topic is displayed.
- *History* — displays the History dialog box. This dialog box lists the help volumes and topics that have been visited. To return to any topic in the list, select its title.
- *Index* — displays the Index Search dialog box. This dialog lists all the words and phrases that the author has marked as index entries. Selecting an index entry, then one of the topics where the entry occurs, displays that topic in the general help dialog.

When using the Help Viewer from the desktop Front Panel, the general help dialog includes an additional dialog button called Top Level. After exploring different help volumes, a user can select this button to return to the top-level of the desktop browser help volume.

## Help Menus

A general help dialog menu bar has five menus: File, Edit, Search, Navigate, and Help. The Search and Navigate menus contain commands for the index and navigation buttons described previously. In addition, the Navigate menu has a Home Topic command that returns to the beginning of the help volume. The remaining menus provide these features:

- *File menu* — duplicates a help window, prints a help topic or the current help volume, or closes the help window.
- *Edit menu* — copies text from the help window to another application.
- *Help menu* — provides help information that describes features of the help dialogs and how to use them.

## Help Index

A help volume has an index of important words and phrases that the user can search to find help topics on a subject. A user can browse or search the index of the current volume, selected volumes, or all help volumes available on the system. Regular expressions such as * (asterisk) and ? (question mark) can be used to search for topics. To view the corresponding help topic, the user selects the index entry.

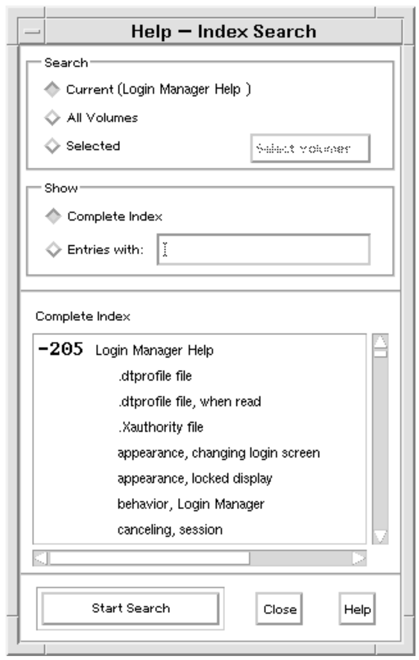



*Figure 1-5* Index search dialog box

Because the help index can be large even for a single volume, index entries can be expanded or contracted. A prefix notation, either a + (plus) or - (minus) sign, is used to show whether an index entry is expanded or contracted. A minus sign indicates that all of the entries are displayed, whereas a plus sign indicates that the entry can be expanded to show additional index entries.

In Figure 1-6 on page 12, the -36 prefix means there are 36 index entries displayed. The +3 notation identifies contracted entries. Selecting a contracted entry causes the list to expand, and the + sign changes to a - sign. The last index entry shown in the figure has been expanded in this manner.

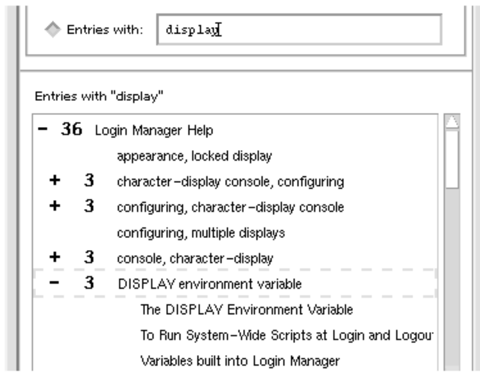



*Figure 1-6* Index entry prefix notation

## Printing from Help

The user can print an individual help topic, a table of contents and index, or the entire help volume. Printed output is text-only. Printing options, such as paper size, number of copies, and destination printer, can also be set in the Print dialog box.

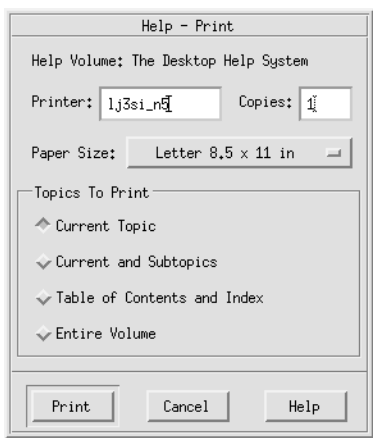



*Figure 1-7* Print dialog box

## Help Topic Organization

An author organizes help information into a logical framework. Most times, but not always, this results in an outline, or a hierarchy of topics. The topic hierarchy in Figure 1-8 consists of a main level, three sections, and subordinate topics. Although Help has been optimized for information that is organized in a hierarchy, you are free to create any kind of organization you want.

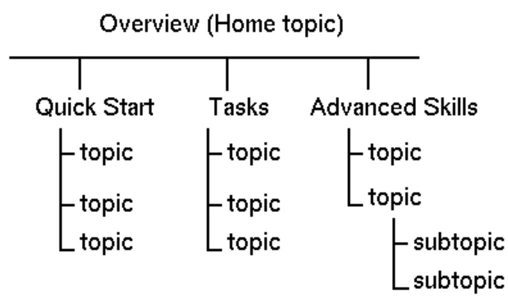



*Figure 1-8* Hierarchy of topics

### Help Topic

A *help topic* is a unit of information identified with a unique ID. A set of tags provided by the Help System is used to mark help topics and create a structural framework. The Help Viewer, which is part of the Help System, is able to directly access and display a help topic.

### Help Volume

A *help volume* is a collection of topics that describe an application or a particular subject. If you are developing application help, typically there's one help volume per application. However, for complex applications, or a collection of related applications, you might develop several help volumes.

### Help Family

Often, software is available as a set of related applications known as a product family. For example, a set of office productivity applications may include a word processor, a spreadsheet application, and a drawing program. Because each application may have its own help volume, it may be desirable to group the related help volumes in a *help family.* A help family can include a single help volume or several volumes.

Assembling your help volumes into a help family is optional. It is required only if you want your help available for browsing within a *help browser* such as the Help Viewer in the Front Panel.

Refer to “To Create a Help Family” on page 97 for a description of help family files and how they are used.

## Help Browser Volume

The desktop provides a special help volume called the *browser* volume that lists help installed on your system. Clicking the Help Viewer control in the Front Panel displays the browser volume shown in Figure 1-9 on page 15.

It lists help families (underlined titles) and any volumes that are members of the help family.

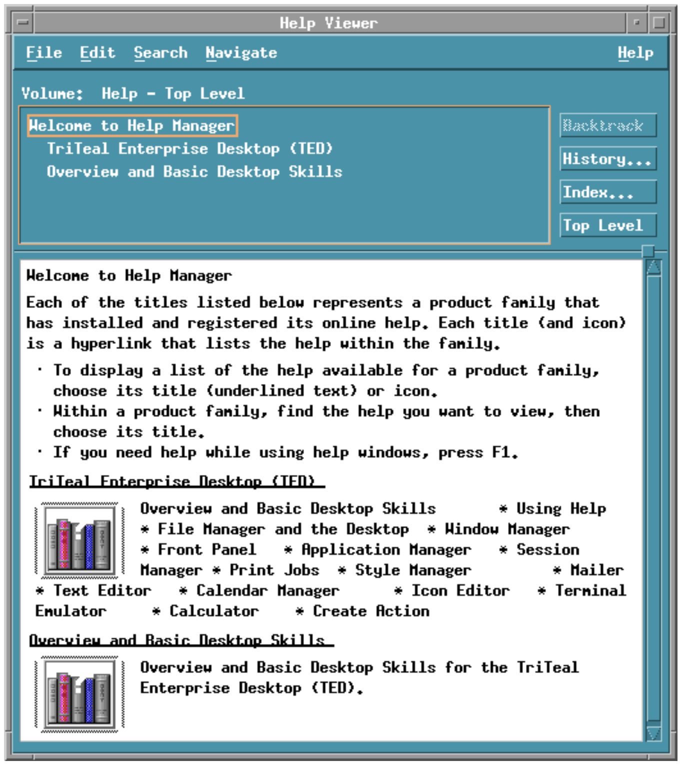



*Figure 1-9* Browser help volume

The browser volume allows access to application-specific help without using the application. Or, if you are writing standalone help, this is the only way for users to get to your help. Even if you have only a single help volume, it must belong to a help family to be browsable using the Help Viewer.

"Adding Your Help to the Browser Volume" on page 96 describes how to create a family file and what you need to do to make your help volume accessible from the browser volume.

# The Author's Job

Writing online help differs from writing printed manuals, so it is important to understand who you are writing for, how the information is accessed, and how the information fits into an application.

## Objectives for Online Help

The two most important objectives for designing quality online help are:

- Get the user back on task as quickly and successfully as possible.
- Educate the user to prevent future need for assistance.

Applying these objectives will help you make decisions regarding what type of help is best and what amount of detail is needed.

## Know Your Audience

Just as with any writing, to do a good job, you must know your audience and understand what they require from the information you are writing. Most importantly, with online help, you need to know the tasks they are attempting and the problems they may encounter.

## Consider How Your Help Is Accessed

It is just as important to understand how users will access your help as it is to identify your audience correctly.

### Application Help

If you are writing help for an application, you need to decide which topics are browsable and which topics are available from the application as context-sensitive help. A topic is browsable if you can navigate to it using the topic tree or hyperlinks. Topics designed exclusively for context-sensitive help might not be browsable because the only way to display the topic may be from within a particular context in the application.

You must also decide if you want your application's help volume to be *registered*. Registered help volumes can be displayed by other applications (such as the Help Viewer), making the information more widely accessible. If another help volume contains hyperlinks to topics in your help volume, your help volume must be registered.

See "Registering Your Application and Its Help" on page 249 for information about installing and registering your application.

### Standalone Help Volumes

If you are writing a standalone help volume (a help volume not associated with an application), you may choose to do things differently.

First, you must provide a path for users to get to all the topics you've written. That is, every topic must be browsable through at least one hyperlink. Also, because there's no application associated with your help, you must rely on a help viewer (such as Help Viewer) to display your help volume.

## Evaluate How to Present Help

An application can incorporate different types of help. It is important to evaluate what kind of help is best suited for your application. For example, the same help information may be presented in a variety of ways. Some choices include key features, a tutorial, examples, task instructions, shortcuts, troubleshooting, reference information, glossary of terms, or referral to hard copy or other online documentation. A help volume often combines different presentations.

## Collaborate with the Application Programmer

If you are writing application help, you should work closely with the application programmer. The degree to which the Help System is integrated into an application is a design decision that you make collectively.

If an application and its help have very loose ties, there may be only a handful of topics that the application is able to display directly. This is easier to implement.

In contrast, the application could provide specific help for nearly every situation in the application. This requires more work, but benefits the user if done well.

It's up to you and your project team to determine the right level of help integration for your project.

## Author's Workflow

After designing your help, you create and process help topics to produce a help volume. Your focus as an author is on these key tasks:

- Write help topics.
- Create run-time help files.
- View the help volume.

### Write Help Topics with HelpTag

Online help is written in ordinary text files. You use special codes, or *tags*, to markup *elements* within the information. The tags form a markup language called HelpTag.

The HelpTag markup language defines a hierarchy of elements that define high-level elements, such as chapters, sections, and subsections, and low-level elements such as paragraphs, lists, and emphasized words.

"General Markup Guidelines" on page 30 gives a brief description of using markup. For a detailed description of each element see "HelpTag Markup Reference" on page 103.

#### Shorthand Markup

The tag set can be used in two different ways to produce run-time help files: shorthand markup or formal markup. The first approach, called *shorthand markup*, is optimized for authors using a standard text editor to "hand-tag" information. That is, the author types the tags in addition to the actual help topic text. To minimize the impact of hand-tagging, shorthand markup incorporates several shortcuts. First, it reduces the number of required start and end tags. It also offers simple character combinations for frequently used markup and stylistic changes.

## Formal Markup

*Formal markup* is a Standard Generalized Markup Language (SGML) that an author can use to create fully compliant SGML help topics. It requires start and end tags for *all* elements. Additionally, the structure of each element must be explicitly tagged. Therefore, the number of tags increases significantly using formal markup. Although an author can enter formal markup using a standard editor, a structured editor is recommended.

### *Structured Editors*

New tools, called structured editors, are becoming available in response to the need to create SGML markup efficiently. Typically, a structured editor provides a context-sensitive menu. That is, the elements that appear in the menu dynamically change based on the location of the cursor in the document.

For example, if you are entering a list, then the menu contains only elements that are valid within the context of a list element. This built-in "intelligence" allows an author to create markup easily.

When an author chooses an element, such as section, head, or list, the editor generates the corresponding start, end, and any intermediate structural tags. For example, when an author selects a chapter element, the editor automatically inserts the intermediate tags required by this element. The author simply types the chapter title. Viewing the generated tags is optional; authors can suppress the tags.

---

**Note –** Either markup approach— shorthand or formal— produces identical online information when compiled and displayed. Choosing which markup approach to use depends on the requirements for your help information and your available authoring tools.

---

### *Using Formal Markup*

If you intend to use formal markup, first read the chapters in *Part 2 - The Author's Job* to become familiar with the set of HelpTag elements. Although shorthand and formal markup share the same tag set, there are several important differences.

Chapter 8, "Reading the HelpTag Document Type Definition," explains key components of the Document Type Definition (DTD) and shows you how to create formal markup. The complete HelpTag Document Type Definition appears in Appendix A.

---

**Note –** The Developer's Kit includes the HelpTag Document Type Definition. The file is located in the `/usr/dt/dthelp/dthelptag/dtd` directory and is named `helptag.dtd`.

---

### See Also

- Chapters 2, 3, and 4 introduce and explain how to use shorthand markup.
- Chapter 5 gives a detailed description of each tag listed in alphabetical order.
- Chapter 8, "Reading the HelpTag Document Type Definition," describes formal markup.
- `dthelptagdtd(4)` man page

## Think Structure, Not Format

If you are familiar with other publishing systems, you may be accustomed to formatting information as you like to see it. Authoring with HelpTag requires you to think about structure and content, not format.

As you write, you use tags to mark certain types of information. When you do so, you are identifying *what* the information is, but not how it should be formatted.

For instance, to refer to a book title, include markup like this:

```
<book>System Administrator's Reference Guide<\book>
```

This abstraction separates structure and content from format, which allows the same information to be used by other systems and perhaps formatted differently. For instance, Help displays book titles using an italic font. However, on another system an italic font may not be available, so the formatter could decide that book titles are underlined.

## Create Run-Time Help Files

The text files you write must be "compiled" using the HelpTag software to create *run-time help files*. It's the run-time help files that are accessed when the user requests help. Run-time files use the Semantic Delivery Language (SDL) format. This delivery language is based on an SGML document type definition designed expressly for online information delivery.

The Help System defines desktop actions and data types for help-specific files. This lets you easily create a run-time file from your desktop by selecting the icon of a help source file and choosing a menu command that processes the file. A `.sdl` extension is used to identify run-time help files. If any errors occurred during processing, they are reported in an error file (*volume*`.err`).

Refer to "Creating Run-Time Help Files" on page 91 for complete instructions to create a run-time help file. For general information about desktop actions and data types, refer to the *Advanced User's and System Administrator's Guide*.

## Review Help as the User Will See It

During the authoring process, you will need to display your help so you can interact with it just as your audience will. To display a help volume from the desktop, double-click the file icon of the run-time help volume (*volume*`.sdl`). Or, you can also display any help topic using the `dthelpview` command. Chapter 4, "Processing and Displaying a Help Volume," describes both methods.

If you are writing application help, and the Help System has been integrated into your application, you can view your help by running the application and making help requests just as the user will.

# Programmer's Job

As a programmer, you add code into your application so that when a user requests context-sensitive help, the application displays help information that is relevant to what the application is doing at that time.

---

**Note –** The`/usr/dt/share/examples/dthelp` directory contains source code for a sample program called `dthelpdemo`. It demonstrates how to add help dialogs to an OSF/Motif application.

---

## Consider How Your Help Is Accessed

Providing useful information to the user requires considering the following:

- What confusing situation commonly arise? Specific help in these situations can save users lots of time.
- Why is the user asking for help now instead of earlier or later? If there are several steps in a process and the user is not at the first step, branch to information that is specific to the step being done. This is more helpful than displaying the same information at each step. If the user is at the first step, make available both detailed information about the first step and an overview of all the steps.
- Is the user requesting context-specific help or just browsing the help information? If it is context-specific, supply information that's relevant to the task now being done.

## Collaborate with the Help Author

Close collaboration with the online help author is needed because the author needs to know how each context-specific topic is reached and the programmer needs to know what is explained in each context-specific topic. Without such coordination, the user may see irrelevant, ambiguous, or misleading information.

Collaboration makes the best use of the programmer's understanding of the application and the author's understanding of how to best communicate relevant information to the user.

## Identify Help Entry Points

An application provides online help by establishing help *entry points*. Entry points are defined in the application and associated with specific help topics. Each of the ways that a user can request help—the Help key, button, or menu—represent entry points. For example, consider an application with a Print dialog box that has a Help button. The author writes a help topic that describes the contents of the dialog box and supplies you with the ID of the topic. You can then associate the ID of the help topic with the Help button using a callback routine.

## Create and Manage Help Dialogs

The Help System application program interface is designed especially for use with OSF/Motif applications. Specifically, Help extends the OSF/Motif widget set by providing two new widget classes (plus convenience functions to manipulate them):

- *General help dialog*, which provides a help window that includes a menu bar and a topic tree, in addition to a help topic display area.
- *Quick help dialog*, which provides a simple help window with a topic display area and a few dialog buttons.

You can use either or both of these types of help windows within your application. Once the application is compiled (with the Help library), the help windows become part of the application.

Chapter 9, "Creating and Managing Help Dialog Boxes," describes the general help and quick help dialog boxes.

## Package and Distribute Help

Your product package includes both the run-time help file (*volume*.`sdl`) and its graphics files. Additionally, you can provide a help family file that enables your volume to be viewed using the Front Panel Help Viewer.

If the help volume uses execution links, you should collaborate with the author to include the appropriate execution link resources in your application's application defaults file. Chapter 13, "Preparing an Installation Package," discusses which help files are delivered with your application.

# Part 2 — The Author's Job

# Organizing and Writing a Help Volume

2

This chapter describes the organization and components of a help source file. It also provides a step-by-step example that shows how to process a help source file to create an online help volume.



## Help Volume Components

A help volume has six major types of components: the home topic, topics, subtopics, entity declarations, meta information, and the glossary.

## Home Topic

The *home topic* is the top-level topic in the topic hierarchy. It is the first topic, or beginning of the help volume. All other topics are subtopics. Your topic hierarchy may be several levels deep. However, to help prevent users from getting lost, you should keep your hierarchy as shallow as possible.

## Topics and Subtopics

Topics and subtopics form a hierarchy below the home topic. Typically, the first level of topics following the hometopic are divided into chapters, using the `<chapter>` element. Within a chapter, topics are organized into sections. Subtopics of an `<s1>` section are entered with `<s2>`, subtopics of `<s2>` entered as `<s3>`, and so on.

Either element, chapter or section, can follow the home topic. There is no visible difference to the user if you start your hierarchy with `<chapter>` or `<s1>`. Figure 2-1 shows a simple hierarchy that includes three chapters. Each chapter contains several first-level sections. The third chapter adds two second-level sections.

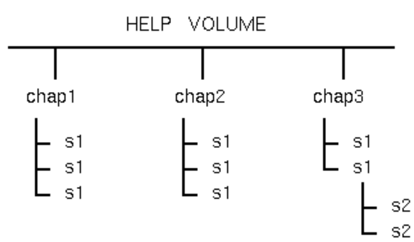



*Figure 2-1* Help volume topic organization

## Entities

An author-defined *entity* can represent a string of characters or a file name. An *entity declaration* defines the entity name and the string or file it represents.

Entities are useful for:

- *Referencing a common string of text.* This is useful if there is some likelihood that the text may change or you simply don't want to type it repeatedly. Each place you want the text inserted, you reference the entity name.
- *Referencing an external file.* Entities are required for accessing graphics files. The `<figure>` and `<graphic>` elements have a required parameter that specifies an entity name, which refers to a graphic image file.

All entity declarations must be entered before any other markup in your help volume. To include an entity that you have defined, you use an *entity reference.* Entity references can be used anywhere within your help volume. When you process your help volume with the HelpTag software, each entity reference is replaced with the text or file that the entity represents. "Using Entities" on page 48 describes how to define and use entities.

## Meta Information

*Meta information* is information about your information. It includes information such as the volume's title, copyright notice, and abstract. The abstract is a brief description of the volume's contents.

The Help System uses the meta information to display the title of a help volume and its copyright information. The abstract description is displayed by the desktop Help Viewer in the Front Panel. Other applications capable of displaying help volumes could also make use of this information.

Meta information can also include help topics that are *not* part of the normal topic hierarchy. Nonhierarchical topics placed in the meta information section are accessed with links.

"Creating Meta Information Topics" on page 43 shows you how to create a meta information section.

## Glossary

The glossary includes definitions for terms that you've used throughout your help volume. If a term is entered using the `<term>` element, then it automatically becomes a *definition link* that, when selected, displays the glossary entry for that term.

# General Markup Guidelines

Online help is written in ordinary text files. You use special codes, or tags, to markup elements within the information. The tags form a markup language called HelpTag. If a standard text editor is used, HelpTag markup is typed. Or, if the editor provides a macro package, tags can be stored and inserted using command keys. HelpTag markup can also be generated using a structured editor (see "Formal Markup" on page 19).

The HelpTag markup language defines a hierarchy of elements that define high-level elements, such as chapters, sections, and subsections, and low-level elements, such as paragraphs, lists, and emphasized words.

## Markup in Your Source Files

The markup for most elements consists of a start tag and an end tag. Start tags are entered with the element name between angle brackets (< and >). End tags are similar, but the element name is preceded by a \ (backslash).

*<element>    ... text ...  <\element>*

For example, to mark the start and end of a book title you use markup like this:

```
<book>Geographical Survey of Northern Wisconsin<\book>
```

Where `<book>` is the start tag, and `<\book>` is the end tag.

### Shorthand Markup

Shorthand markup is a streamlined tag set designed for authors who are using a standard text editor to "hand-tag" information. Shorthand markup provides several shortcuts. First, it minimizes the use of end tags. For example, you do not need to enter end tags for chapters, sections, or paragraphs. In addition, when possible, intermediate tags are automatically assumed. For instance, the chapter and section elements allow you to omit a `<head>` tag; you just type your headline.

Shorthand markup also simplifies the markup for many inline elements as well as stylistic changes. Rather than entering a begin and end tag, vertical bars are used to delimit the text like this:

*<element|  ... text ...  |*

For example, here's the short form of the `<book>` element shown previously:

```
<book| Geographical Survey of Northern Wisconsin|
```

If the element has parameters, they're entered before the first vertical bar like this:

*<element parameters| ... text ... |*

Some elements support an even shorter form where the start and end tags are replaced with a special two-character shortcut. For example, the `<emph>` (*emphasis*) element, whose normal syntax looks like this:

`<emph>` *... text ...* `<\emph>`

can be entered using this shorthand form:

`!!` *... text ...* `!!`

Chapter 3, "Writing a Help Topic," introduces shorthand markup and gives examples of the most frequently used elements. Chapter 5, "HelpTag Markup Reference," provides an alphabetical list of elements and describes each element in detail.

### Formal Markup

If you intend to use formal markup, you still need to become familiar with the information covered in Part 2 of this book. Then refer to Chapter 8, "Reading the HelpTag Document Type Definition," for a description of formal markup.

## Displaying HelpTag Symbols

At times, you may need to use the < (left angle bracket), the \ (backslash), or the `&` (ampersand) as text characters. To do so, precede each with an ampersand (`&<`, `&\`, or `&&`).

# A Help Volume at a Glance

The following markup illustrates important elements of a help volume and the tags used to enter them. This example uses shorthand markup, which omits intermediate SGML structural tags and minimizes the number of required end tags. Indentation is used to highlight the hierarchical relationship of the elements; you don't need to indent the help files that you write.

```
All entity declarations go here (before any other markup).
<helpvolume>
     <metainfo>
          <title>   Volume Title
          <copyright>
          Copyright topic goes here ...
          <abstract>
           The abstract describing your help volume goes here.
             There may be other meta information topics.
              .
              .
              .
     <\metainfo>

     <hometopic>   Home Topic Title
                     Help volume introduction  goes here ...
          <s1>   Title of First Topic Goes Here
                   Body of the first topic goes here ...
          <s1>   Title of Second Topic
                   Body of the second topic goes here ...
                   <s2>   Title of Suptopic
                            Body of the subtopic goes here ...
               .
               .
               .

     <glossary>
     The body of the glossary, which contains term definitions, goes here ...
<\helpvolume>
```

## Help Source Files

Online help is written in ordinary text files. You process, or compile, these files with the HelpTag software to create run-time help files that can be read by the Help System.

### Creating Your volume.htg File

HelpTag expects a primary control file named *volume*`.htg` or *volume*`.ctg`, where *volume* is a name you choose. File extensions are used to distinguish whether the control file references shorthand (`.htg`) or formal (`.ctg`) markup.

Be sure your *volume* name is unique and meaningful. If your *volume* name is too general, it may conflict with another volume that someone else has created. If you are writing application help, one recommended practice is to use the application's class name. For example, the class name for the Icon Editor is Dticon, so its help volume is named `Dticon.htg`.

### Multiple Source Files

The *volume*`.htg` file contains entity declarations and entity references to files that make up the help volume. Although HelpTag expects a single *volume*`.htg` file as input, you can separate your work into multiple source files. Additional files are sourced into the *volume*`.htg` file using *file entities*. A file entity is like a pointer to another file. That file, in effect, is inserted wherever the entity's name appears in the *volume*`.htg` file. The referenced files can also contain entity references to yet other files. (Entities can also be used to reference text strings.)

### Example

Suppose a help volume has six chapters and each chapter is a separate file. The files are: HomeTopic, Metainfo, TOC, Tasks, Reference, and Glossary. The *volume*`.htg` file for the help volume includes file entities for each of the six files and a list of entity references that instruct the HelpTag software to process the files.

```
<!entity HomeTopic                FILE “HomeTopic”>
<!entity MetaInformation          FILE “Metainfo”>
<!entity TableOfContents          FILE “TOC”>
<!entity Tasks                    FILE “Tasks”>
<!entity Reference                FILE “Reference”>
<!entity Glossary                 FILE “Glossary”>

&HomeTopic;
&MetaInformation;
&TableOfContents;
&Tasks;
&Reference;
&Glossary;
```

The details of running HelpTag are covered in “To Create a Run-Time Help Volume” on page 91.

## Help Files in File Manager

File Manager represents help files as file icons with a question mark. In Figure 2-2 on page 34, there are two source files (.ctg and .htg extensions) and one run-time file (.sdl extension). If you double-click a markup file, your standard editor opens the file for editing. Double-clicking a .sdl file displays the run-time file using the Help Viewer.

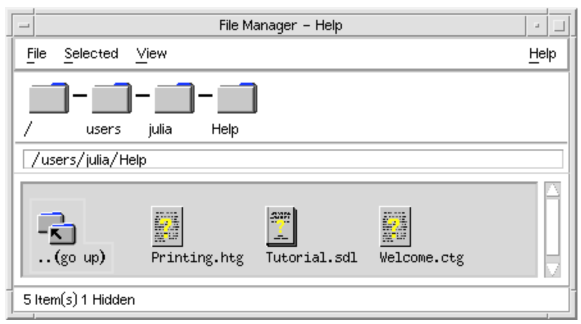



*Figure 2-2* File Manager view of help files

To create a run-time help volume, first select the .htg or .ctg file icon in File Manager. Then, choose Compile from the File Manager Selected menu.

### See Also

- `dthelpaction(5)` man page

## Writing Your First Help Volume: A Step-by-Step Example

Typically you organize your help files in a `help` directory. This step-by-step example demonstrates how to create and view a standalone help volume. (As a standalone volume, it does not involve interaction with an application.)

To create and process a help volume, you follow these steps:

1. Create the source directory for help files.
2. Create the build directory.
3. Create the master HelpTag file.

4. Create the `helptag.opt` file.

5. Create the run-time help files.

6. Display the help volume.

Each task is described in the section that follows. The markup language used in the text files is introduced later in this chapter. HelpTag markup is described more fully in Chapter 3, “Writing a Help Topic,” and Chapter 5, “HelpTag Markup Reference.”

## Create the Source Directory

1. Create a directory named `helpfiles` where you will create and process your help files.

2. Create a text file named `Commands` in the directory just created.

   For this example, all the information is put into a single file. Typically, you will use multiple files to fully explain the system or application you are writing help for.

   The `Commands` file contains text and element tags. The element tags within the **<** and **>** (angle brackets) indicate the structure of the information.

3. Type the following markup text in the `Commands` file.

```
 <hometopic>  Command Summary
                        <idx|commands|

Your &product; is capable of the following operations:

<list bullet>
 * <xref ChannelChange>
 * <xref VolumeUp>
 * <xref VolumeDown>
 * <xref VolumeMute>
<\list>

Choose one of the hyperlinks (underlined phrases)
to find out how to perform that operation.

<s1 id=ChannelChange>  Changing the Channel
                        <idx|channel, changing|
Speak the command:
```

```
<ex> channel<\ex>
followed by a number from one to ninety nine.

<s1 id=VolumeUp>  Turning Up the Volume
                        <idx|volume, changing|
Speak the command:
<ex> volume up<\ex>

For additional volume, speak the command:
<ex> more<\ex>

(See also <xref VolumeDown> )

<s1 id=VolumeDown> Turning Down the Volume
                        <idx|volume, changing|
Speak the command:
<ex> volume down<\ex>

To further reduce the volume, speak the command:
<ex> more<\ex>

(See also <xref VolumeUp>  and <xref VolumeMute> )

<s1 id=VolumeMute> Turning Off the Sound
                        <idx|volume, changing|
                        <idx|sound, on/off|
Speak the command:
<ex> sound off<\ex>

To restore the sound, speak the command:
<ex> sound on<\ex>

(See also <xref VolumeDown>  and <xref VolumeUp> )
```

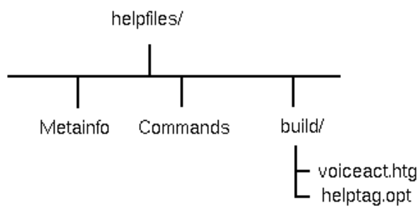
4. Create a text file that gives the information a title, provides copyright information, and provides other information about the online help.

   In this example, the following text is put into a file called `Metainfo` in the same directory as the `Commands` file.

```
<metainfo>
     <title> Using the &product;
     <copyright>
     &copy; 1995 Voice Activation Company.  All rights reserved.
     <abstract> Help for Using the &product;.
<\metainfo>
```

## Create the Build Directory

Create a subdirectory named `build` in the `helpfiles` directory.




## Create the Master HelpTag File

1. In the `build` subdirectory, create a text file whose name is of the form *volume*`.htg`. In this example, the file is named `voiceact.htg`.
2. In the `.htg` file, define *entities* that associate the names of the `Commands` and `Metainfo` files with entity names. Also, define any entities that are used (either directly or indirectly) in the `Commands` and `Metainfo` files. Finally, refer to the `Commands` and `Metainfo` files by their entity names.

In this example, the contents of the `voiceact.htg` file look like this. The text within the `<!--...-->` elements are comments, which are ignored.

```
<!-- Declare an entity for each of the source text files. -->
<!entity MetaInformation     FILE   "Metainfo">
<!entity Commands            FILE   "Commands">

<!-- Define an entity that names the product and includes
     the trademark symbol (&tm;). -->

<!entity product  "VoAc&tm; Voice-Activated Remote Control">

<!-- Include the text files. -->

&MetaInformation;
&Commands;
```

## Create the helptag.opt File

1. In the `build` subdirectory, create a file named `helptag.opt` and put the following text into it. This information selects HelpTag options and indicates where to search for any files defined in FILE entity declarations.

   ```
   onerror=go
   memo
   search=./
   search=../
   ```

   The `onerror=go` option instructs the HelpTag software to continue processing input files even if an error occurred. See "Parser Options" on page 193 for an explanation of parser options. For a complete list and description of parser options, refer to the `dthelptag(1)` man page.

2. Verify that the `/usr/dt/bin` directory is in your search path by typing this command in a terminal window:

   ```
   echo $PATH
   ```

   If the directory is not in your path, add it to your `PATH` environment variable. If you're not sure how to do this, refer to your system documentation or ask your system administrator.

## Create the Run-Time Help Files

1. Open File Manager and change to the `build` subdirectory. Select the `voiceact.htg` file icon and choose Compile from the Selected menu in File Manager.

   This executes the HelpTag software which creates a run-time version of your online help volume (`voiceact.sdl`). Status and error messages are placed in a new file, whose name is of the form *volume*`.err`.

2. Open the `voiceact.err` file to check that your file processed without errors. If errors occurred, fix them by editing or renaming the text files as needed.

---

**Note –** You can run HelpTag manually in a terminal window.

---

To do so, execute the following command:

```
dthelptag -verbose voiceact.htg
```

The -verbose option tells HelpTag to display its progress on your screen.

### Display the Help Volume

From the build subdirectory, double-click the voiceact.sdl file icon.

This displays the help volume using the desktop Help Viewer. You can now scroll the information and jump to related information by choosing hyperlinks.

---

**Note –** You can run the Help Viewer manually in a terminal window.

---

To do so, execute the following command. It displays the new help volume.

```
dthelpview -h voiceact.sdl
```

#### See Also

- Chapter 4, "Processing and Displaying a Help Volume," on page 89.
- Chapter 7, "Command Summary," on page 191.

## Creating a Topic Hierarchy

The topic hierarchy within your help volume begins with the home topic. Each help volume must have one home topic. The first level of subtopics below the home topic may be entered with <chapter> or <s1>.

Additional levels of subtopics are entered with <s2>, <s3>, and so on. The HelpTag markup language supports nine topic levels, <s1> to <s9>. However, information more than three or four levels deep often leads many readers to feel lost.

When a help volume is displayed, the help window displays a list of topics in its topic tree. Any topic entered with a <chapter> or <s1...s9> tag automatically appears in the topic tree. This provides an easy way to browse and view topics.

To enable users to display other related information from within a topic, you create hyperlinks. To do so, you assign a unique ID to each destination topic. Hyperlinks make it possible to reference a specific ID anywhere in your help information.

### Example

Suppose you want to create a hierarchy to match this simple outline:

*Tutorial for New Users*
  *Module 1: Getting Started*
  *Module 2: Creating Your First Report*
  *Module 3: Printing the Report*
  *Module 4: Saving Your Work and Quitting*
*Task Reference*
  *Starting and Stopping*
    *To Start the Program*
    *To Quit the Program*
  *Creating Reports*
    *To Create a Detailed Report*
    *To Create a Summary Report*
*Concepts for Advanced Users*
  *Using Report Hot Links*
  *Sharing Reports within a Workgroup*
*Reference*
  *Command Summary*
  *Report Attributes Summary*

Then the general outline of your help volume would look like this. (The body of each topic and IDs for the topics are not shown.)

```
<hometopic>  Welcome to Report Master
  <chapter>  Tutorial for New Users
    <s1>   Module 1: Getting Started
    <s1>   Module 2: Creating Your First Report
    <s1>   Module 3: Printing the Report
    <s1>   Module 4: Saving Your Work and Quitting
  <chapter>  Task Reference
    <s1>   Starting and Stopping
      <s2>   To Start the Program
      <s2>   To Quit the Program
    <s1>   Creating Reports
      <s2>   To Create a Detailed report
      <s2>   To Create a Summary report
  <chapter>  Concepts for Advanced Users
    <s1>   Using Report Hot Links
    <s1>   Sharing Reports within a Workgroup
```

```
<chapter>   Reference
  <s1>   Command Summary
  <s1>   Report Attributes Summary
```

Indentation is used here to make it easier to see the structure of the help volume. You do not have to indent your files.

### See Also

- “Accessing Topics” on page 46 describes assigning IDs to topics.
- “Creating Hyperlinks” on page 66 describes how to create hyperlinks.

## ▼ To Create a Home Topic

♦ Use the `<hometopic>` element as follows:

```
<hometopic> Title
Body of topic.
```

If you include a meta information section (`<metainfo>`), the home topic must follow it.

### Examples

Here’s a home topic with a title and a single sentence as its body:

```
<hometopic> Welcome to My Application


Congratulations, you’ve entered
the online help for My Application.
Here’s a sample home topic that includes hyperlinks to its four
subtopics:
<hometopic> Welcome to Report Master

Welcome to the online help for Report Master.
Choose one of the following hyperlinks:

<list bullet>
 * <xref Tutorial>
 * <xref Tasks>
 * <xref Concepts>
 * <xref Reference>
<\list>
If you need help, press F1.
```

The preceding markup produces this output:


## ▼ To Add a Topic to the Hierarchy

♦ To add another topic at the same level, repeat the same element.

*Or,* to add a subtopic (a topic one level deeper in the hierarchy), use the element that is one level deeper than the preceding topic.

### Example

If the current topic is an `<s1>`, enter a subtopic using `<s2>`.

```
<s1 id=getting-started>  Getting Started

<s2 id=starting-the-program>  Starting the Program
Here's the body of the first subtopic.

<s2 id=stopping-the-program>  Stopping the Program
Here's the body of the second subtopic.
```

The second `<s2>` is also a subtopic of the `<s1>`.

---

**Note –** Sometimes a parent-child-sibling metaphor is used to describe the relationships between topics in a hierarchy. In the preceding example, the `<s1>` topic is the "parent" of both `<s2>`s (the "children" topics). The two `<s2>`s are "siblings" of one another. All three topics are "descendents" of the home topic.

---

# Creating Meta Information Topics

The meta information section is primarily intended for information about information. Similar to providing a copyright page in a book, this section includes information such as the volume title, copyright, trademark, and other notices.

A secondary use of the meta information section is to enter help topics that are not part of the normal topic hierarchy. These nonhierarchical topics are useful for creating custom definition links that pop-up a topic in a quick help dialog box.

## ▼ To Create a Meta Information Section

1. Enter the `<metainfo>` tag to start the section, and enter the required subelements `<title>` and `<copyright>` as shown:

   <metainfo>

   <title> *Volume Title Here*

   <copyright>
   *Body of copyright topic here.*
   .
   .
   .

2. Enter any of the optional elements as shown:

   <abstract>
   *Body of the abstract topic here.*
   *Do not use any HelpTag markup within the abstract!*

3. Enter the `<\metainfo>` end tag to end the section.

   .
   .
   .
   <\metainfo>

---

**Note –** Some elements in the meta information section require a `<head>` tag before the topic heading.

---

The <abstract> section is recommended. Applications that access help volumes can use this information to present a brief description of the volume. Because the abstract might be displayed in plain text windows (that do not support multiple fonts and graphic formatting), avoid including any HelpTag markup in the abstract.

### Example

Here's a typical meta information section:

```
<metainfo>
  <title>  Report Master, Version 1.0

  <copyright>
    <otherhead> Report Master

    <image>
    Version 1.0
    &copy; Copyright Reports Incorporated 1995
    All rights reserved.
    <\image>

  <abstract>
    This is the online help for the mythical Report Master
    application. This help includes a self-guided tutorial,
    a summary of common tasks, general concepts, and quick
    reference summaries.

<\metainfo>
```

The <image> element is used to preserve the author's line breaks. The © entity inserts the copyright symbol.

### See Also

- "To Link to a Meta Information Topic" on page 73.

## Adding a Nonhierarchical Topic

Topics entered with a <chapter> or <s1...s9> element tag automatically appear in the topic tree. When a title is selected in the topic tree, the corresponding help topic is displayed in a general help dialog box. However, sometimes you may want to create and display a topic independent of the topic hierarchy you have created. For example, you might want to display a topic in a separate, quick help window.

## ▼ To Add a Nonhierarchical Topic

♦ Add the topic just before the end of your meta information section using the `<otherfront>` element as follows:

`<otherfront id=`*id*`><head>` *Topic Title*
*Body of topic.*

The ID parameter and `<head>` tag are required.

You can add as many `<otherfront>` topics as you want. They may be in any order, but they must be the last topics in the `<metainfo>` … `<\metainfo>` section.

### Example

This partial help volume shows how a general topic is added to the meta information section. The topic's title is "Pop-up!" and its ID is `my-popup-topic`.

```
<metainfo>

  <title> My Help
  <copyright>
     This is My Help, Version 1.0.  &copy; 1995.
        .
        .
        .
  <otherfront id=my-popup-topic> <head> Pop-up!

  This is a pop-up topic, displayed via a definition link
  somewhere in my help volume.
<\metainfo>

<hometopic>  Welcome to My Help
        .
        .
        .
```

Presumably, within some other topic in the help volume, there’s a definition link to display this topic.

The link might look like this:

```
Here’s a sample of a pop-up <link my-popup-topic Definition>
definition link<\link>.
```

The words "definition link" become the active hyperlink and will be displayed with a dashed underline. Selecting the link displays the "Pop-up!" topic in a quick help dialog box.

### See Also

- “Creating Hyperlinks” on page 66.
- “<otherfront>” on page 159.

## Accessing Topics

Many elements in the HelpTag language support an ID attribute. An ID is a unique name used internally to identify topics and elements within topics. An ID is defined only once, but multiple hyperlinks and cross-references can refer to the same ID. IDs are not seen by the user.

If you are writing help for an application, IDs are also used by the application to identify particular topics to display when the user requests help. For example, you might write several topics that describe an application’s menus. The IDs that you assign to the topics are used by the application developer. By defining identical IDs within the application code, the developer can integrate specific topics. This allows the application to access and display the correct topic when help is requested for a particular menu.

### Rules for ID Names

- ID strings may contain letters (A - Z and a - z), digits (0 - 9), and the minus (–) sign, and must begin with a letter.
- Author-defined IDs may *not* use the _ (underscore character); it is reserved for IDs that are built into some HelpTag elements.
- Case is *not* significant, but is often used to increase readability.

- ID strings cannot be longer than 64 characters.
- Each ID within a single help volume must be unique.

## ▼ To Add an ID to a Topic

♦ Use the `id` parameter for the element as follows:

<*element* `id=`*id*> . . .

The elements that start a new topic and support an author-defined ID are:

- <chapter id=*id*>
- <otherfront id=*id*>
- <rsect id=*id*>
- <s1 id=*id*>
- <s2 id=*id*> . . .<s9 id=*id*>

### Built-in IDs

A few elements have built-in IDs and, therefore, do not support an author-defined ID. Each of the following elements also starts a new topic, but these elements have predefined IDs (shown in parentheses):

| | |
|---|---|
| <abstract> | (_abstract) |
| <copyright> | (_copyright) |
| <glossary> | (_glossary) |
| <hometopic> | (_hometopic) |
| <title> | (_title) |

## ▼ To Add an ID to an Element within a Topic

♦ If the element supports an author-defined ID, use the `id` parameter for the element as follows:

<*element* `id=`*id*> . . .

The elements (within a topic) that support an ID attribute are:

- <figure id=*id*>
- <graphic id=*id*>
- <image id=*id*>

- <location id=*id*>
- <p id=*id*>

*Or,* use the `<location>` element to set an ID at an arbitrary point within the topic as follows:

```
<location id=id> text <\location>
```

Where *text* is any word or phrase where you want to add an ID. The `<\location>` end tag is required. When you activate a link to a location ID, the Help Viewer displays the topic containing the ID and scrolls the window to the ID position.

### Examples

If you add an ID to a figure, you must have a caption. The caption is needed in case a cross reference is made to the figure's ID. In that case, the caption becomes a hyperlink to the figure.

Here's the markup for a figure with the ID `my-big-picture`.

```
<figure id=my-big-picture entity=big-picture-TIFF>
Here's My Figure
<\figure>
```

Here's a paragraph where the phrase "easier than ever" has been assigned the ID `easy-spot`:

```
Getting help is <location id=easy-spot> easier than ever<\location>.
```

## Using Entities

An *entity* can represent a string of characters or the contents of a file. An *entity declaration* defines the entity by associating the entity name with a specific character string or file name. An *entity reference* is replaced by the string or file contents when you process the help volume with the `dthelptag` command.

Entities are useful for:

- *Referencing a common string of text.* This is useful if there is some likelihood that the text may change or you simply don't want to type it repeatedly. Each place you want the text inserted, you reference the entity name.
- *Referencing an external file.* Entities are required for accessing graphics files. The `<figure>` and `<graphic>` elements have a required parameter that you use to specify an entity name, which refers to a graphic image file.

File entities are also useful for splitting your HelpTag source into multiple files. Use entity references to include other files into your master HelpTag file for processing.

### Rules for Entity Declarations

- Entity names may contain letters (A - Z and a - z), digits (0 - 9), and the minus (–) sign, and must begin with a letter.
- Case is *not* significant in entity names, but is often used to increase readability.
- Entity names cannot be longer than 64 characters.
- Each entity name must be unique within a single volume.

## ▼ To Create a Text Entity

1. Declare an entity as follows:

   `<!entity` *Entityname* `"`*text*`">`

   Where *Entityname* is the name of the entity and *text* is the string that you want substituted for every reference to the entity. Remember, all entity declarations must come before any other markup in your help volume.

2. For each location where you want the *text* string to be inserted, enter the entity reference as follows:

   `&`*Entityname*`;`

   The `&` (ampersand) and `;` (semicolon) characters are required for the HelpTag software to properly recognize the entity reference.

### Example

The following line declares a text entity named Assoc that contains the string "Society of Agricultural Engineers":

```
<!entity Assoc "Society of Agricultural Engineers">
```

The following sentence includes a reference to the entity:

```
Welcome to the &Assoc;.
```

When the help volume is processed with the HelpTag software, the entity reference is replaced with the value of the entity. So, the sentence reads:

Welcome to the Society of Agricultural Engineers.

## ▼ To Create a File Entity

1. Declare an entity as follows:

   `<!entity` *Entityname* `FILE "`*filename*`">`

   Where *Entityname* is the name of the entity and *filename* is the name of the file. The keyword `FILE` is required.

2. Reference the entity as follows:
   - If the file is a text file, enter the following entity reference at each location where you want the contents of the file inserted.

     `&`*Entityname*`;`

     The `&` (ampersand) and `;` (semicolon) characters are required for the HelpTag software to properly recognize the entity reference.

   - If the file is a graphics file, include the name of the entity as a parameter in one of the following markup lines:

     `<figure entity=`*Entityname* `... >`

     *Or:*

     `<graphic entity=`*Entityname* `... >`

     *Or:*

     `<p gentity=`*Entityname* `... >`

---

**Note –** Do *not* include any path in the file name. If the file is not in the current directory when you run the HelpTag software, add the appropriate search path to the `helptag.opt` file. (See “To Create a Run-Time Help Volume” on page 91.)

---

### Example: Text File Entities

Suppose you wrote the text for your help volume in three files named `file1`, `file2`, and `file3`, plus a fourth file containing your `<metainfo>` …`<\metainfo>` section. You could include them in your *volume*`.htg` file like this:

```
<!entity MetaInformation   FILE  "metainfo">
<!entity MyFirstFile       FILE  "file1">
<!entity MySecondFile      FILE  "file2">
<!entity MyThirdFile       FILE  "file3">

&MetaInformation;

<hometopic>  My Home Title

Welcome to my application's help volume.

&MyFirstFile;
&MySecondFile;
&MyThirdFile;
```

### Example: A Graphic File Entity

Suppose a simple help volume has a figure in the home topic and the graphic image for the figure is stored in a file named `picture.tif`. The following example shows how that image would be used in a figure.

```
<!entity MetaInformation  FILE  "metainfo">
<!entity MyPicture        FILE  "picture.tif">

&MetaInformation;

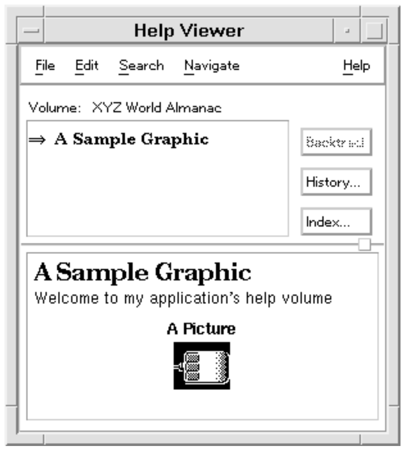
<hometopic> A Sample Graphic

Welcome to my application's help volume

<figure nonumber entity=MyPicture>
A Picture
<\figure>
```

The text "A Picture" is the figure's caption.

The markup produces this output:




### See Also

- "Displaying Graphics" on page 78.

# Writing a Help Topic 3

This chapter describes elements that you can use to format your text. It also explains how to include graphics and how to create hyperlinks to other help topics. Examples shown in this chapter use shorthand markup.



## Creating Help Topics

A help topic is a unit of information identified with a unique ID. Help topics are grouped into a logical framework that best describes the product you are writing online help for.

Each topic you write should have an element (or tag) that marks the start of the topic:

*<element* `id=`*id>* *Help Topic's Title*
*The body of the topic*

Where *element* is one of the following: `chapter, s1, s2, …, s9`. The body of the topic may begin on any line after the title.

The topic's position within the topic hierarchy is determined by the *element* used to start the topic and by the *element* used to start the immediately preceding topic. For example, a topic that starts with `<s2>` and immediately follows a topic that starts with `<s1>` makes the `<s2>` topic a subtopic of the `<s1>` topic.

The *id* is required if the topic is to be accessed either from the application (if you are writing application help) or from a hyperlink.

The help topic title can be any string. If the title string occupies more than one line in your source file, end all but the last line with an `&` (ampersand). To force a line break at a particular place within the title, use a \ (backslash) character.

### Example

The following line marks the start of a topic using the `<s1>` tag:

```
<s1 id=welcome>Welcome to My Application
```

To force the title to be displayed on two lines, you use a \ (backslash) like this:

```
<s1 id=welcome> Welcome to \ My Application
```

### See Also

- Chapter 2, "Organizing and Writing a Help Volume," describes the general structure of a help volume, including how to create a topic hierarchy.

## Creating Structure within a Topic

Within the body of a help topic, you have the following elements to choose from to organize and present your information:

- *Paragraphs* are used for bodies of text.

- *Lists* are used for itemized information. There are several types of lists including bulleted, ordered (numbered), and plain.
- *Subheadings* are used to partition sections within a topic.
- *Graphics* can be included within your text as *inline graphics* or displayed between paragraphs as standalone *figures*.
- *Hyperlinks* provide references to related topics. A hyperlink may lead to a subtopic, deeper in the hierarchy, or branch to a topic in a completely different part of the hierarchy, or even in another help volume.
- *Computer literals* are computer-recognized text, such as file names and variable names, that can be displayed as either separate example listings or inline elements.
- *Notes, cautions, and warnings* call the reader's attention to important information.
- *Emphasis* is used to highlight important words and phrases within paragraph text.

## ▼ To Start a Paragraph

♦ Insert a blank line after the previous paragraph or other element.

*Or,* use the `<p indent>` element and parameter if the paragraph is to be indented.

*Or,* use the `<image>` element if you want the paragraph to maintain the line breaks that you enter in your source file.

An end tag for `<p>` is not required. However, the `<\image>` end tag *is* required with the `<image>` element.

### Examples

Here are two paragraphs, separated by a blank line. Because neither paragraph has any special parameters, the `<p>` tag does not have to be entered (it is assumed when you enter one or more blank lines):

```
The Application Builder provides an interactive, graphical
environment that facilitates the development of desktop
applications.
```

```
The Application Builder is designed to make it easier for developers
to construct applications that integrate well into the desktop. It
provides two basic services: assembles Motif objects into the
desired application user interface, and generates appropriate calls
to the routines that support desktop integration services.
```

If you want a paragraph indented from the left margin, include the optional `indent` parameter:

```
<p indent> An indented paragraph can be used to draw the reader's
attention to an idea.
```

The following paragraph overrides the automatic word wrap in help windows and maintains the line breaks exactly as entered in the source file. The `<image>` element is especially useful for entering addresses.

```
<image>
Brown and Reed Financial Investors
100 Baltic Place  Suite 40 New York, New York
<\image>
```

## ▼ To Enter a List

♦ Use the `<list>` element as shown:

```
<list type spacing>
* item
* item
 .
 .
 .
* item
<\list>
```

Where *type* indicates the type of list you want: `bullet` (default), `order`, or `plain`; and *spacing* is `loose` (default) or `tight`. Each *item* in the list is marked with an `*` (asterisk).

### Examples

Here's a simple list. Because the *type* isn't specified, it defaults to a bulleted list. Because *spacing* isn't specified, it defaults to loose, which leaves a blank line between each item.

```
<list>
* Creating a Mail Message
* Sending a Message
* Reading Your Mail
<\list>
```

The online format of the preceding markup is:


To format the same list with numbers and reduced spacing between items, use:

```
<list order tight>
* Creating a Mail Message
* Sending a Message
* Reading Your Mail
<\list>
```

The output is:


## ▼ To Enter a Lablist

A lablist is a two column list with optional column headings.

♦ To create a labeled list without headings, use the `<lablist>` element as shown:

```
<lablist spacing>
    \ label 1\  item 1  text
    \ label 2\  item 2  text
```

```
      .
      .
      .
    \ label N\  item N  text
<\lablist>
```

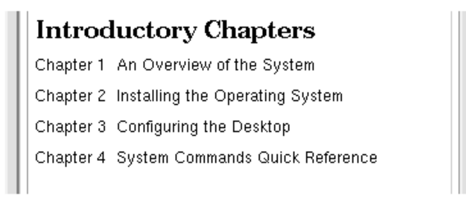
Where *spacing* is `loose` (default) or `tight`.

### Example

Here's a list of labeled chapter descriptions. The optional label headings are not provided.

```
<lablist tight>
\Chapter 1\ An Overview of the System
\Chapter 2\ Installing the Operating System
\Chapter 3\ Configuring the Desktop
\Appendix A\ System Commands Quick Reference
<\lablist>
```

The output is:




## ▼ To Enter a Lablist with Headings

♦ Use the `<lablist>` and `<labheads>` elements as shown:

```
<lablist spacing>
    <labheads>   \ heading for labels  \ heading for items
    \ label 1\   item 1 text
    \ label 2\   item 2 text
```

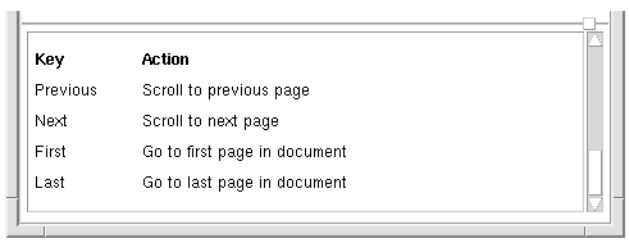
```
        .
        .
        .
    \  label N\  item N text
<\lablist>
```

### Example

This markup:

```
<lablist>
<labheads>\Key \Action
\Previous\ Scroll to previous page
\Next\ Scroll to next page
\First\ Go to first page in document
\Last\ Go to last page in document
<\lablist>
```

produces this output:




### See Also

- “<list>” on page 149 summarizes the use of the `<list>` element.
- “<lablist>” on page 141 summarizes the use of the `<lablist>`.

## ▼ To Provide Subheadings within a Topic

♦ For medium headings (slightly smaller than the topic title), use the following markup:

`<otherhead>` *Heading*

*Or,* for small headings, use the following markup:

`<procedure>` *Heading*

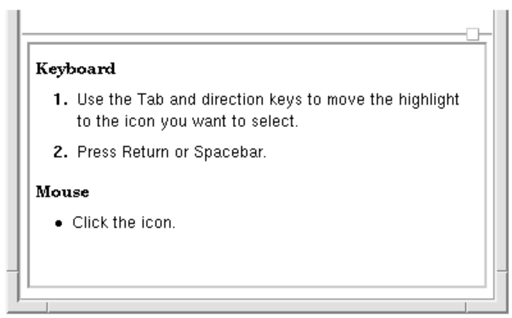
Subheadings add structure *within* a topic, but they do not appear in the list of topics in the topic tree.

### Example

Here, the `<procedure>` element is used to add a small heading before each list.

```
<procedure>Keyboard
<list order>
* Use the Tab and direction keys to move the highlight to the icon
  you want to select.
* Press Return or Spacebar.
<\list>
<procedure>Mouse
<list bullet>
* Click the icon.
<\list>
```

This markup creates this output:




## ▼ To Show a Computer Listing

For computer listings that do *not* contain any special character sequences that will be interpreted as HelpTag markup, use the `<ex>` (*example*) element as shown:

`<ex` *size*`>`
*Computer text here.*
`<\ex>`

For computer listings that contain special character sequences used by HelpTag, use the <vex> (*verbatim example*) element as shown:

```
<vex size>
Computer text here.
<\vex>
```

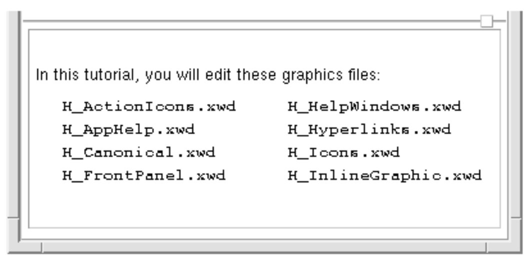
The optional *size* attribute, which determines the size of the font used to display the example, can be specified as `smaller` or `smallest`.

### Example

Here the <ex> element is used to represent a directory listing in a terminal window.

```
In this tutorial, you will edit these graphics files:
<ex>
H_ActionIcons.xwd      H_HelpWindows.xwd
H_AppHelp.xwd          H_Hyperlinks.xwd
H_Canonical.xwd        H_Icons.xwd
H_FrontPanel.xwd       H_InlineGraphic.xwd
<\ex>
```

The markup produces this output:




Line breaks appear where you enter them in your source file. If the example is too wide for the help window, a horizontal scroll bar appears so the user can scroll to see all the example text.

### See Also

- “To Display a Computer Literal” on page 65.
- “<ex>” on page 123.
- “<vex>” on page 177.

## ▼ To Add a Note, Caution, or Warning

♦ Include the `<note>`, `<caution>`, or `<warning>` element as follows:

```
<note>
Body of note here.
<\note>

<caution>
Body of caution here.
<\caution>

<warning>
Body of warning here.
<\warning>
```

To associate an icon with the note, caution, or warning element, define a file entity that identifies the graphics file containing the icon. Use one of the predefined entity names:

- <!ENTITY NoteElementDefaultIconFile FILE "*filename*">
- <!ENTITY CautionElementDefaultIconFile FILE "*filename*">
- <!ENTITY WarningElementDefaultIconFile FILE "*filename*">

If you do not want icons with notes, cautions, or warnings, don't declare the corresponding entities. (Remember, all entity declarations must come before any other markup at the beginning of your help volume.) If you include such an entity reference, be sure the graphics file is in your HelpTag search path (`helptag.opt`).

Names of the default icons used by the Help System for note, caution, and warning elements are specified in the following entities.

- <!ENTITY NoteElementDefaultIconFile FILE "noteicon.pm">
- <!ENTITY CautionElementDefaultIconFile FILE "cauticon.pm">
- <!ENTITY WarningElementDefaultIconFile FILE "warnicon.pm">

These default icons are located in the `/usr/dt/dthelp/dthelptag/icons` directory.

If you create your own icon images for notes, cautions, and warnings, be sure to keep them small so they will fit into the area allotted. Also, the graphic images must be in your HelpTag search path, which is specified in your `helptag.opt` file.

## Example

The following markup for a note, warning, and caution produces the output shown in Figure 3-1.

```
<note>
Before installing your application, complete the options checklist
to determine the amount of disk space required.
<\note>

<warning>
This product is highly acidic and can cause skin irritation. Wearing
protective gloves is mandatory when applying this product.
<\warning>

<caution>
    Do not place your fingers near the parrot cage!
<\caution>
```

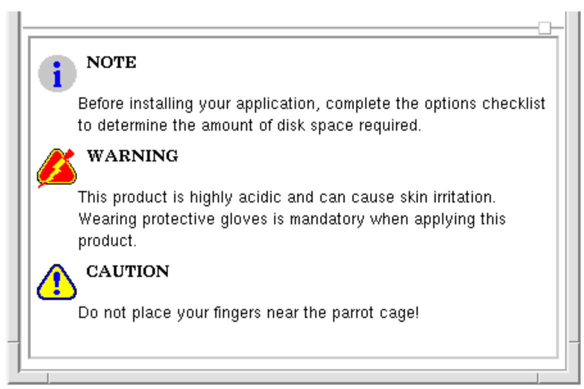



*Figure 3-1* Note, warning, and caution help icons

## See Also

- "To Create a Run-Time Help Volume" on page 91 describes using a `helptag.opt` file.
- "Using Entities" on page 48.

## Entering Inline Elements

Inline elements are used to mark words or phrases within a paragraph of text. These elements affect the font used to format particular items.

### ▼ To Emphasize a Word or Phrase

♦ Use the `<emph>` element (*emphasis*) as shown:

`<emph>` *text* `<\emph>`

*Or,* use the shorthand form:

`!!` *text* `!!`

Emphasized text is displayed using an italic font.

#### Example

Here’s how you might emphasize an important word:

```
A thousand times <emph>no<\emph>
```

Or, using the shorthand form:

```
A thousand times !!no!!
```

In both cases, the word "no" is displayed in italics.

### ▼ To Enter a Book Title

♦ Use the `<book>` element as shown:

`<book>` *title* `<\book>`

*Or,* use the short form:

`book|` *title* `|`

Book titles are displayed using an italic font.

#### Example

Here’s how you would enter the title of this guide:

```
<book|The Help System Author’s and Programmer’s Guide|
```

## To Emphasize Using a Bold Font

- Use the `<term>` element as shown:

  `<term nogloss>` *bold text* `<\term>`

  *Or*, use the shorthand form:

  `<term nogloss |`*bold text* `|`

  The `<term>` element is used to create a glossary entry. However, by adding the `nogloss` parameter, the text is displayed in a bold font without being added to the glossary.

## To Display a Computer Literal

- Use the `<computer>` element as shown:

  `<computer>` *text* `<\computer>`

  *Or*, use the shorthand form:

  *" text "*

### Example

Computer text is useful for identifying a file name. Here the `helptag.opt` file name is tagged using shorthand markup. The file name will be displayed in computer text.

This markup:

```
Add the search path to your "helptag.opt" file.
```

produces this output:

Add the search path to your `helptag.opt` file.

## To Display a Variable

- Use the `<var>` element (*variable*) as shown:

  `<var>` *text* `<\var>`

  *Or*, use the short form:

  `<var |`*text* `|`

*Or,* use the shorthand form:

`%%` *text* `%%`

Variables are displayed using an italic font.

### Example

This command-line syntax uses a variable to show that the user supplies a file name.

```
dtpad %%filename%%
```

It produces this output:

`dtpad` *`filename`*

Variables can appear within computer text or computer example listings. This example specifies *volume* as a variable part of a file name:

```
The HelpTag software takes your “%%volume%%.htg” file as input.
```

It produces:

The HelpTag software takes your *`volume`*`.htg` file as input.

In both of these examples, the `%%` pairs could have been entered with the long form (`<var>`…`<\var>`) or the short form (`<var`|…|).

## Creating Hyperlinks

A hyperlink references a specific topic or location in a help volume. This requires that the element you want to reference is given a unique ID. These HelpTag elements can be assigned IDs: `<chapter>`, `<s1...s9>`, `<location>`, `<p>`, `<image>`, `<figure>`, and `<graphic>`.

Help supports five types of hyperlinks:

- *Hypertext links* "jump" to another help topic. By default the new topic is displayed in the same window, but you may request that the new topic be displayed in a new window.
- *Definition links* display a topic in a simple pop-up help window. Most frequently, definition links are used to access the definition of a new term or phrase used within a sentence.
- *Man page links* display any man page installed on the system.

- *Execution links* execute a shell command or program. This greatly expands the possibilities for what happens when the user activates a hyperlink.
- *Application-defined links* create custom links that the application interprets. This provides facilities for communication between the Help System and the application.

To create a hyperlink to an element, you include its ID in a hyperlink command. HelpTag provides two elements, `<xref>` and `<link>`, that can be used to create hyperlinks to an ID. In addition, the `<p>`, `<image>`, and `<figure>` elements can be used to create hyperlinks using a graphic image.

## Using the <xref> Element

If you are linking to an element with a title, such as a chapter or section, the simplest way to do so is with the `<xref>` element. When you use `<xref>` to create a link, you specify the ID of the topic that you want to link to. The title of the topic is inserted in place of the `<xref>` element and becomes the active hyperlink.

Hypertext links created with `<xref>` display the new topic in the same window. If you desire different behavior by using the other link types, then you must use the `<link>` element.

Also, you *cannot* use `<xref>` to jump to topics that have built-in IDs (such as `<hometopic>` or `<glossary>`). To create a hyperlink to any of those elements, you must use the `<link>` element.

## ▼ To Create a Link Using <xref>

♦ Use the `<xref>` element as shown:

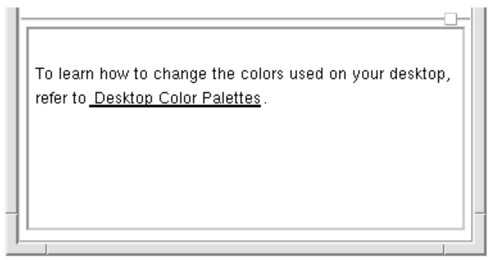
```
<xref id>
```

where *id* is the ID of the chapter or section that you want to create a link to. Notice that capitalization of the ID is not significant.

Here’s an example that creates a link to a section title.

```
<s1 id=colorpalettes>Desktop Color Palettes
.
.
.
To learn how to change the colors used on your desktop,
refer to <xref colorpalettes>.
```

The section title replaces the `<xref>` element. The title is a hyperlink, designated by an underline. This is how the sentence would appear in a help volume.




## Using the Link Element

You can use either the `<xref>` or `<link>` element to create standard hypertext links. However, to use the other types of links listed on page 66, you must use the `<link>` element.

## ▼ To Create a Link Using <link>

♦ To jump to a topic within the same volume, use the `<link>` element as shown:

```
<link id>text<\link>
```

Where *id* is an ID declared somewhere in the help volume, and *text* is the portion of your help text that is underlined to indicate it is an active hyperlink.

### Example

Here is the previous example using the <link> element instead of the <xref> element.

```
<s1 id=colorpalettes>Desktop Color Palettes
.
.
.
To learn how to change the colors used on your desktop,
refer to <link colorpalettes>Desktop Color Palettes<\link>.
```

## To Create a Link to a Predefined ID

- To jump to a topic (within the same volume) that has a predefined ID, use the <link> element as shown:

  `<link hyperlink="`*id*`">`*text*`<\link>`

  All the predefined IDs start with a _ (underscore) character. So this makes it necessary to use the `hyperlink= "`*id*`"` form.

### Example

This link jumps to the home topic of the current volume:

```
Return to <link hyperlink="_hometopic">Introduction<\link>.
```

## To Create a Link to a Topic in a Different Volume

- To jump to a topic in another help volume:

  `<link hyperlink="`*volume id*`" JumpNewView>`*text*`<\link>`

  If the other volume is registered, the *volume* parameter is just the base name of the volume file. If the volume is not registered, you must include a complete path name to the volume.

  The `JumpNewView` parameter is recommended for links to other volumes so that users realize they have jumped into another volume. The previous view remains displayed so they can see where they came from.

### Examples

This link jumps to the home topic of a help volume called `GeoMap`:

```
To view a map of the United States, see <link hyperlink="GeoMap
_hometopic"> Geography Maps <\link>.
```

Here’s the same link, but it displays the topic in a new window:

```
To view a map of the United States, see <link hyperlink=”GeoMap
_hometopic” type=JumpNewView> Geography Maps <\link>.
```

This link jumps to the topic, `DesktopKeyboardNav`, in the help volume named `Intromgr`.

```
For more information, see <link hyperlink=”Intromgr
DesktopKeyboardNav”>Keyboard Shortcuts for the Desktop<\link>.
```

If the help volume you are targeting is not registered on the desktop, then you must provide a complete path name to the volume or specify the appropriate search path in your `helptag.opt` file.

### See Also


- “Registering Your Application and Its Help” on page 249.
- “<figure>” on page 125.
- “<image>” on page 137.
- “<link>” on page 146.
- “<p>” on page 162.
- “<xref>” on page 180.


## ▼ To Create a Definition Link

♦ If you are linking to a term in the glossary, use the `<term>` element as shown:

<term>*text*<\term>

*Or,* use the shorthand form:

++*text*++

Whenever you use the `<term>` element, be sure you include the corresponding definition in the Glossary.

If you are linking to a topic within the same help volume, use the `<link>` element as shown:

```
<link id Definition>text<\link>
```

Where *id* is a topic ID (or the ID of an element within the topic) and *text* is the portion of your help text that you want to be the active hyperlink. The `Definition` keyword specifies that the link should pop-up a quick help dialog box.

*Or,* if you are linking to a topic in another help volume, use the `<link>` element as shown:

```
<link hyperlink="volume id" Definition>text<\link>
```

If the other volume is registered, the *volume* parameter is just the base name of the volume file. If the volume is not registered, you must include a complete path name to the volume.

### Example

The following link creates a definition link that displays the copyright topic in the meta information:

```
<link hyperlink="_copyright" type=Definition>Version
Information<\link>
```

The phrase "Version Information" becomes the hyperlink text (dashed underline).

### See Also


- “Creating a Glossary” on page 85.
- “<term>” on page 172.
- “<link>” on page 146.


## ▼ To Create a Man Page Link

♦ Use the `<link>` element as shown:

```
<link manpage Man>text<\link>
```

To request a man page from a particular section, use the hyperlink parameter like this:

```
<link hyperlink="section manpage" Man>text<\link>
```

For man page links, the hyperlink parameter is the same string you would enter if executing the man command in a terminal emulator window.

---

**Note –** If you are writing help for an application and you include any man page links, your application must include special support for man pages. See "To Display a Man Page" on page 222. (The desktop Help Viewer includes support for man page links.)

---

### Example

Here's a link that displays the man page for the grep command:

```
Refer to the <link grep Man> grep(1)<\link> command.
```

"Man" is a keyword for the <link> element, so if you want to create a link that displays the man page for the man command, you must use the hyperlink parameter:

```
Refer to the <link hyperlink="man" Man>man(1)<\link> command.
```

To display a man page in a particular section, precede the man page name with the section number. The following link displays the "mkdir" man page from section 2 (which is different from the man page of the same name in section 1):

```
Refer to the <link hyperlink= "2 mkdir" Man>mkdir(2)<\link> command.
```

### See Also

- "<link>" on page 146.

## To Create an Application-Defined Link

♦ Use the <link> element with the AppDefined parameter as shown:

<link hyperlink="*data*" AppDefined>*text*<\link>

Where *data* is a text string passed to the application when the link is invoked and *text* is the hyperlink.

### Example

Suppose you are writing help for an application that prints three styles of reports. You might create three hyperlinks like this:

```
Choose a report type:
<list plain tight>
* <link hyperlink="Report-Daily" AppDefined>Daily Report<\link>
* <link hyperlink="Report-Month-To-Date" AppDefined>MTD
Report<\link>
* <link hyperlink="Report-Year-To-Date" AppDefined>YTD
Report<\link>
<\list>
```

If your application is set up to handle these special links and to interpret the hyperlink strings, it could generate the appropriate report based on the hyperlink chosen by the user.

For a complete example, refer to the sample application code located in the /usr/dt/share/examples/dthelp directory.

## To Link to a Meta Information Topic

♦ Use the <link> element as shown:

<link hyperlink="*_id*">*text*<\link>

Where *id* is the predefined ID associated with the element you want to link to and *text* is the word or phrase that you want to be the active hyperlink.

Most topics within the meta information section have predefined IDs, so they do *not* allow author-defined IDs. The predefined IDs consist of the element name preceded by an underscore character. For example, the ID for the <copyright> topic is _copyright. (Case is not significant.)

The predefined IDs for the meta information topics are listed below:

| | |
|---|---|
| `<abstract>` | (_abstract) |
| `<copyright>` | (_copyright) |
| `<title>` | (_title) |

Topics entered with the `<otherfront>` element can be linked to just like any normal topic in the topic hierarchy.

### See Also

- "Built-in IDs" on page 47 lists the Help System predefined IDs.

## Execution Link Control

Most hyperlinks display a related help topic, but hyperlinks can also execute shell commands and scripts. Links of this type are called execution links. Because execution links interact with a user's system, the Help System provides an *execution policy* to control how execution links are handled.

The Help System uses resources to define the behavior of execution links. The `DtNexecutionPolicy` resource is set in an application's application defaults file to modify how execution links are handled by the Help System. In addition the Help System uses a set of resources called *execution aliases*. An execution alias is a resource that assigns a name (or label) to the command string or script that an execution link executes.

### Execution Policy Default Behavior

When an execution link is selected, if the link has an execution alias, the Help System retrieves the value of the alias and executes the command. If an execution alias has not been defined, the Help System displays a confirmation dialog box that shows the command to be executed and asks the user whether to execute the command or to cancel the operation.

### Execution Aliases

When using execution links in a help volume, it is recommended to create execution aliases. That is, in the application's application defaults file you define an alias (a name) that represents the actual command to be executed. One advantage of this method is that it isolates the actual commands from the help volume source files. This makes it possible to edit the commands in the application defaults file without changing the hyperlinks in the help volume. Each hyperlink references an alias name, which remains unchanged even though its content may have been edited. For instance, a tutorial help volume that uses scripts could be easily customized to accommodate a particular shell environment by modifying the shell script commands in the application defaults file.

## ▼ *To Create an Execution Alias*

To create an execution alias in an application's application defaults file, use this resource specification syntax:

`application_name.executionAlias.alias_name: command`

Where:

| | |
|---|---|
| *application_name* | Name or class name of the application that owns the help volume. |
| executionAlias | Keyword that identifies the resource is an alias. |
| *alias_name* | Name assigned to the command. |
| *command* | Shell command or script to be executed for this link. |

There is no restriction on the length of the *command* string. To enter commands with multiple lines, end each line (except the last) with a `\` (backslash).

### Examples

This resource entry creates an execution alias named, `StartDtterm`, which starts a terminal emulator. The `&` (ampersand) starts the command in the background.

```
Dtterm.executionAlias.StartDtterm: dtterm &
```

This entry creates an alias named, `xclockAlias`, that executes the `xclock` application in an application named NightAlert.

```
NightAlert.executionAlias.xclockAlias: xclock &
```

## Using Execution Aliases in Hyperlinks

An execution alias can be referenced using the `<link>` element or used in conjunction with elements that have a hyperlink parameter, such as `<p>` or `<figure>`.

### ▼ *To Create an Execution Link Using an Execution Alias*

♦ Use the `<link>` element as shown:

```
<link "DtHelpExecAlias alias_name [default_command]" Execute
>text<\link>
```

Where:

| | |
|---|---|
| `DtHelpExecAlias` | Keyword that identifies this link has an execution alias. |
| *alias_name* | Name defined as an alias in the execution alias resource specification. |
| *default_command* | *Optional.* If provided, this command is executed when an execution alias has not been loaded from an application's application defaults file. For example, application resources are not loaded when a help volume is displayed from an information viewer, such as Help View. |
| *text* | The portion of your help text that you want to designate as the hyperlink text (underlined). |

---

**Note –** If the command you are executing doesn't finish immediately, run it in the background by appending an `&` (ampersand) to the command. If you don't, the help window will not operate until the command finishes.

---

#### Examples

This hyperlink references the execution alias named, `xclockAlias`. The resource definition for the alias is shown in the section "Execution Aliases" on page 74.

The link starts the `xclock` program running in the background. The phrase "Start the Clock" becomes the hyperlink. Clicking the hyperlink runs the `xclock` program in a separate window. To end the program, close the window.

```
<link "DtHelpExecAlias xclockAlias" Execute>Start the Clock<\link>
```

Here is the same hyperlink including an optional default command.

```
<link "DtHelpExecAlias xclockAlias xclock &" Execute>Start the
Clock<\link>
```

## DtNexecutionPolicy Resource

The `DtNexecutionPolicy` resource enables a system administrator or user to select an appropriate level of security for a given application.

The resource values that can be set are:

| | |
|---|---|
| `DtHELP_EXECUTE_QUERY_ALL` | Query all execution links. |
| `DtHELP_EXECUTE_QUERY_UNALIASED` | Query only link commands that do not have execution aliases defined. |
| `DtHELP_EXECUTE_NONE` | Do not execute any execution links. |
| `DtHELP_EXECUTE_ALL` | Execute all execution links. |

The default value is `DtHELP_EXECUTE_QUERY_UNALIASED`. Any execution links defined as execution aliases will be automatically executed, whereas the Help System will display a confirmation dialog box for any other execution links.

It is not recommended for the application developer to set the `DtNexecutionPolicy` because this prevents a system administrator or user from altering the value.

### See Also


- "<link>" on page 146.
- "<figure>" on page 125.
- "<p>" on page 162.

- `DtHelpDialog(3)`.
- `DtHelpQuickDialog(3)`.

## Displaying Graphics

Help supports four graphics formats:

- *Tagged Image File Format (TIFF)*—Color, grayscale, and black-and-white images created by many standard drawing and scanning applications (*filename*`.tif`).
- *X Window dump*—Screen dumps from the X Window System™ created with the `xwd` utility (*filename*`.xwd`).
- *X pixmap*—Color icon images (*filename*`.pm`).
- *X bitmap*—Two-color icon images (*filename*`.bm`).

Each graphic is maintained as a separate file. The file format is determined using the file name extensions listed.

### ▼ To Create a Figure

1. Declare a file entity to identify the image file to be included in the figure.

   ```
   <!entity graphic-entity FILE "filename.ext">
   ```

   Remember, all entity declarations must come before any other markup at the top of your help volume.

2. Use the `<figure>`element as shown:

   ```
   <figure entity=graphic-entity>
   caption string
   <\figure>
   ```

   Where *graphic-entity* is the entity name for the graphic file you want to display, and *caption string* is an optional string. Caption text is displayed above the graphic.

By default, figures are numbered and the number is prepended to your *caption string*. To create a nonnumbered figure, include the `nonumber` parameter (as shown in one of the following examples).

If you want the figure to be a hyperlink, use the `ghyperlink` (*graphic hyperlink*) and `glinktype` (*graphic link type*) parameters as shown:

```
<figure entity=graphic-entity ghyperlink="id" glinktype=type>
caption string
<\figure>
```

The ghyperlink and glinktype parameters work just like the hyperlink and type parameters for the <link> element.

### Examples

For these examples, assume that you've declared these two file entities at the top of your help volume:

```
<!entity FirstPicture  FILE "first.tif">
<!entity SecondPicture FILE "second.pm">
```

- The following figure displays the graphic in the first.tif file and displays a number (by default) and caption:

  ```
  <figure entity=FirstPicture>
  Here's the First Picture
  <\figure>
  ```

- Here's a figure that displays the second.pm file without a number or a caption:

  ```
  <figure nonumber entity=SecondPicture>
  <\figure>
  ```

  If you add an ID to a figure, you must have a caption. The caption is needed in case an <xref> uses the figure's ID; if so, the caption is inserted in place of the <xref> and becomes a hyperlink to the figure.

- The following figure is an execution hyperlink that runs the xclock program:

  ```
  <figure entity=SecondPicture ghyperlink="xclock &"
  glinktype=execute>
  Choose This Figure to Start the Clock
  <\figure>
  ```

### See Also

- "<figure>" on page 125.
- "<link>" on page 146.

## ▼ To Display an Inline Graphic

1. Declare a file entity to identify the image file to be used in the figure.

   <!entity *graphic-entity* FILE "*filename.ext*">

Remember, all entity declarations must come before any other markup at the top of your help volume.

2. Use the `<graphic>` element as shown:

   ```
   ... text <graphic entity=graphic-entity>   text ...
   ```

   Where *graphic-entity* is the entity name for the graphic file you want to display.

To use a graphic as a hyperlink, place it inside a `<link>` element:

```
<link parameters><graphic entity=graphic-entity><\link>
```

---

**Note –** The `<graphic>` element is intended for small graphics, although larger images can be used. Additional white space is added between lines to accommodate the height of the graphic.

---

### Example

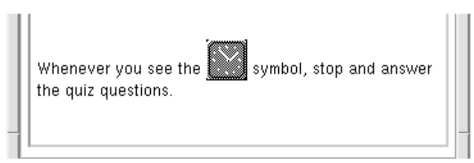
Here's an example that uses a small X bitmap image in the middle of a sentence. First, at the top of the volume, the bitmap file must be declared as a file entity:

```
<!entity StopWatch FILE  "stopwatch.bm">
```

Within the help text, the image is inserted using the `<graphic>` element:

```
Whenever you see the <graphic entity=StopWatch> symbol, stop and
answer the quiz questions.
```

This markup produces this output.




## ▼ To Wrap Text around a Graphic

1. Declare a file entity to identify the image file to be included with the paragraph.

```
<!entity graphic-entity FILE "filename.ext">
```

2. Use the <p> element (*paragraph*) with the `gentity` parameter as shown:

```
<p gentity=graphic-entity> Paragraph text here ...
```

Where *graphic-entity* is an entity name that refers to the graphic file you want inserted.

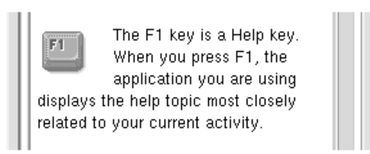
### Example

Suppose you want to display an icon named `sample.pm` and wrap paragraph text around it. First, declare the file entity:

```
<!entity HelpKeyIcon FILE "helpkey.xwd">
```

Then, enter the paragraph:

```
<p gentity=HelpKeyIcon gposition=left> The F1 key is a Help key. When
you press F1, the application you are using displays the help topic
most closely related to your current activity.
```




To right-justify the graphic, add the `gposition` parameter like this:

```
<p gentity=HelpKeyIcon gposition=right>Many desktop components
support multicolor icons, in addition to two-color images.
```

Here’s the markup for a paragraph wrapped around an icon, where the icon is a hyperlink that displays a topic with the ID `icon-editor` in a new window:

```
<p gentity=my-icon ghyperlink="icon-editor" glinktype=JumpNewView>
Many desktop components support multicolor icons, in addition to the
two-color images.
```

### See Also

- “<p>” on page 162

## Including Special Characters

Many special characters and symbols are available within HelpTag. You display a particular character by entering the appropriate entity reference.

Some special character entities are declared in the file `helpchar.ent`. The `helpchar.ent` file is located in the `/usr/dt/dthelp/dthelptag` directory. To access these characters, either copy the particular entity declaration into your own volume, or include the entire `helpchar.ent` file. Unused entity declarations are ignored.

Refer to Chapter 6, "Summary of Special Character Entities," for a complete list of the available characters.

### ▼ To Include a Special Character

1. Refer to Chapter 6, "Summary of Special Character Entities," to determine the entity name for the character you want to display. Also, note whether it is a built-in special character.
2. If the character is *not* a built-in special character, add the following two lines among your other entity declarations (where *entity-name* is a meaningful name to you):

   ```
   <!entity entity-name FILE "helpchar.ent">   &entity-name;
   &entity-name;
   ```

   Also, add this line to your `helptag.opt` file:

   ```
   search=/usr/dt/dthelp/dthelptag
   ```

   If the character is built into HelpTag, you can skip step 2.
3. Wherever you want to display the special character, enter its entity reference:

   ```
   &entity-name;
   ```

#### Examples

The entity for the copyright symbol (©) is a built-in special character, so all you have to do to display it is use this entity:

```
&copy;
```

To display the uppercase greek letter sigma (Σ), you must first include the `helpchar.ent` file (at the top of your help volume with your other entity declarations) as shown here:

```
<!entity  SpecialCharacterEntities FILE "helpchar.ent">
&SpecialCharacterEntities;
```

Then you can place the following entity reference where the sigma character is to appear:

```
&Usigma;
```

As with any entity, case is not significant in the entity names for special characters.

### See Also

- Chapter 6, “Summary of Special Character Entities,” on page 183.

## Including Comments and Writer’s Memos

Frequently it is useful to include within your source files comments that are not intended to be part of the help text. Text marked with the *comment* element is always ignored by the HelpTag software. Comments can be used to make notes to yourself or to another author, or to exclude some markup without taking it out of the file.

In addition to standard comments, HelpTag also provides a `<memo>` element for entering writer’s memos. Memo notes appear in your help topics during reviews, but not when you make your final help files. Authors commonly use the `<memo>` element to write questions or make notes to reviewers.

### ▼ To Insert a Comment

♦ Use the comment begin marker (`<!--`) and end marker (`-->`) as shown:

`<!--` *text here is completely ignored* `-->`

The HelpTag software ignores all markup between the `<!--` and `-->`. A comment *cannot* be nested within another comment.

#### Example

Here's an example that has two comments, a line before the paragraph, and a single word within the paragraph.

```
<!-- Here is my rough draft of the introduction: -->

Welcome to my application. This software
is <!-- perhaps --> the fastest and most
efficient software you'll ever own.
```

### ▼ To Insert a Writer's Memo

♦ Use the `<memo>` element as shown:

`<memo>` *text* `<\memo>`

By default, the text within the `<memo>` element is ignored by the HelpTag software (just like a comment). However, if you add the `memo` option to your `helptag.opt` file (or specify the `memo` option with the `dthelptag` command), all memos within your help volume appear in a bold font.

#### Example

Suppose you are writing about your application and have a question for the project team. You can include the question within the text using the `<memo>` element like this:

```
<memo>Team: Will the product also
support 32-bit characters?<\memo>
```

If you process the help volume with the following command (or include `memo` in your `helptag.opt` file), the memo appears in the help text in a bold font.

`dthelptag` *volume* `memo`

If the memo option is not used (or the `nomemo` option is used), the text within the memo is ignored and does not appear in the help text.

## Creating an Index

The index for a help volume is similar to the index for a book. As an author, it is important to create index entries for your topics that will allow users to search for keywords or concepts. Creating a thorough index ensures that users will be able to find topics quickly and accurately.

## To Mark an Index Entry

♦ Within the topic you want to index, use the <idx> element as shown:

<idx>*keyword*<\idx>

*Or,* the short form:

<idx|*keyword*|

*Or,* to control how the entry is sorted, use the <sort> subelement as shown:

<idx>*keyword*<sort>*sortkey*<\idx>

Where *keyword* is the text you want to display in the index and *sortkey* is the text used during sorting.

The <idx> element can be used anywhere within the topic. Neither the *keyword* nor the optional *sortkey* are displayed in the topic.

### Examples

Here's the start of a topic with two keyword index entries:

```
<s1 id=getting-started>Getting Started with Helpview
<idx>starting Helpview<\idx>
<idx> Helpview, starting<\idx>

Welcome ...
.
.
.
```

The following example indexes the + (plus character), putting it in the keyword index where the word "plus" would appear:

```
<idx>+<sort>plus<\idx>
```

## Creating a Glossary

Like a glossary in a book, your help volume can contain a glossary that defines important terms. The glossary, which is marked using the <glossary> element, is the last topic in your help volume.

Throughout your help volume, each key word or phrase that you enter with the `<term>` element automatically becomes a definition hyperlink to the term's definition in the glossary.

### See Also

- "**<dterm>**" on page 118.
- "**<glossary>**" on page 128.
- "**<term>**" on page 172.

## ▼ To Mark a Glossary Term

♦ Use the `<term>` element as shown:

`<term>`*word or phrase*`<\term>`

*Or,* use the short form:

`<term|`*word or phrase*`|`

*Or,* use the shorthand form:

++*word or phrase*++

If the term within the help text isn't spelled exactly the same as the definition in the glossary, you can specify the "glossary form" of the term like this:

`<term` *"glossary form"*`>`*word or phrase*`<\term>`

Where *glossary form* is the term exactly as it appears in the glossary. This is useful if the term must be plural in a help topic (because of its context), but must be singular in the glossary.

Terms are displayed using a bold font and automatically become a definition hyperlink. When the term is chosen, its glossary definition appears in a quick help dialog.

---

**Note –** If you mark a term that you intentionally do not define in the glossary, add the `nogloss` attribute to the `<term>` element. This allows the term to be displayed in the bold font used for terms, but without creating a link to the glossary.

---

### Examples

If your glossary has a definition for the term "widget", you can enter it as a term like this:

```
A ++widget++ is the fundamental building block of OSF/Motif user
interfaces.
```

If the glossary entry is "widget", but you need to use the plural form within the sentence, you could enter the term like this:

```
<term "widget">Widgets<\term> are the fundamental building blocks of
OSF/Motif user interfaces.
```

If you want to enter the same term, but you either don't want to include it in the glossary or you don't want it to be a hyperlink, use the `nogloss` parameter like this:

```
<term nogloss> Widgets<\term>are the fundamental building blocks of
OSF/Motif user interfaces.
```

The equivalent short form is:

```
<term nogloss|Widgets| are the fundamental building blocks of
OSF/Motif user interfaces.
```

## ▼ To Define a Term in the Glossary

♦ Enter the `<dterm>` element into the glossary as shown:

```
<glossary>
.
.
.
<dterm>word or phrase
Definition of the term
.
.
.
```

Be sure to keep the `<dterm>`words and phrases sorted within the glossary.

### Example

Here's part of a glossary that includes the definition of the term SGML:

```
<glossary>
```

```
.
.
.
<dterm>SGML
Standard Generalized Markup Language. An
international standard [ISO 8859: 1986] that
establishes a method for information interchange.
SGML describes constructs for marking the
structure of information separate from its
intended presentation or format.
```

# Processing and Displaying a Help Volume

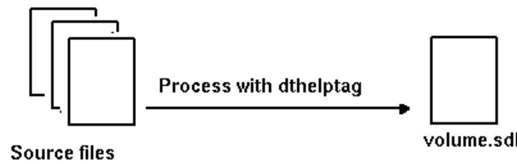
4

This chapter shows you how to process your marked-up help files to create an online format that you view using the Help System. It also describes how to make your help volume accessible from the desktop Front Panel Help Viewer.



## Overview

Before a help volume can be displayed, you must create a run-time help file by processing your files with the HelpTag software. Run-time files use an online presentation format called *Semantic Delivery Language.* A `.sdl` file extension identifies a run-time help file.




The Help System defines desktop actions and data types for help-specific files. This lets you easily process and view a run-time help file from the desktop.

## HelpTag Software

The HelpTag software can be invoked automatically by double-clicking a help source file in File Manager or by running the `dthelptag` command manually in a terminal window.

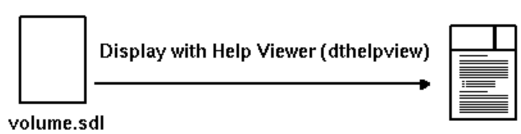
Helptag does two significant tasks:

1. The HelpTag *parser* converts your marked-up files into an internal format (Semantic Delivery Language) understood by the Help System. If you've made any markup errors, the errors are reported in a file named *volume*`.err`.

2. If there are no parser errors, the master help volume file (*volume*`.sdl`) is created.

## Viewing Your Volume

After processing your source files with HelpTag, your help volume is ready to be displayed. You can display it by double-clicking the *volume*`.sdl` file icon in File Manager, or use the `dthelpview` command in a terminal window.




If you have written help for an application and the application is ready to use, you can display your help by running the application and asking for help.

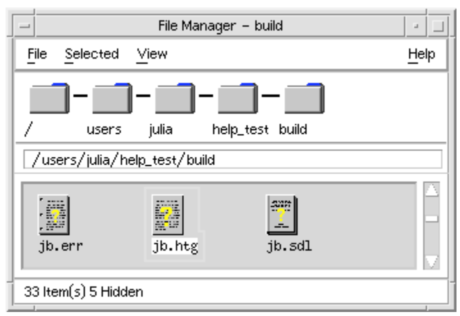
## Creating Run-Time Help Files

When you run HelpTag, it reads your *volume*.htg or *volume*.ctg file and any additional source files that are included using entities. Also, graphics file names are validated.

Be sure the /usr/dt/bin/dthelptag command is in your search path. (If you're not sure how to do this, ask your system administrator.)

### ▼ To Create a Run-Time Help Volume

1. Open File Manager and change to the directory where your *volume*.htg file is located.




2. Select the file icon.
3. Choose Compile from the File Manager Selected menu.

   The *volume*.htg file is processed and creates a *volume*.sdl file and *volume*.err file.

### HelpTag Output

The output from HelpTag is a run-time help volume, named *volume*.sdl. If any errors occurred during processing, they are reported in an error file (*volume*.err). If no errors were encountered, the *volume*.err file contains copyright information and several status lines.

Setting the `onerror=go` option in your `helptag.opt` file allows the parser to continue processing (if possible) after encountering an error. Without the `onerror=go` option, the parser halts when the first error is detected. The *volume*`.sdl` file is not created until the source file is without errors.

The *volume*`.sdl` file, plus your graphics files, are read by the Help System to display help topics. The run-time help file has the same base name as your *volume*`.htg` file. For example, if your *volume*`.htg` is named `Librarian.htg`, then the help volume name is `Librarian.sdl`.

---

**Caution –** Never rename a run-time help file or graphics file after running HelpTag. The information stored in the *volume*`.sdl` file depends on the original names. If you rename your *volume*`.htg` file or any of your graphic files, be sure to rerun HelpTag.

---

## ▼ To Run the dthelptag Command Manually

♦ Run the `dthelptag` command as follows:

`dthelptag` *command-options* *volume* *parser-options*

Where *command-options* are options entered before the *volume* name and *parser-options* are options entered after the *volume* name. "Processing HelpTag Files (dthelptag)" on page 192 lists all available options.

### Example: Commands

The following command processes a help volume named `MyVolume`:

```
dthelptag MyVolume
```

Using the `-verbose` option causes the progress of the processing to be displayed on your screen:

```
dthelptag -verbose MyVolume
```

Adding a search path enables HelpTag to find files stored in a subdirectory (of the current directory) named `graphics`:

```
dthelptag -verbose MyVolume search=graphics
```

### Example: A helptag.opt File

Here's a sample `helptag.opt` file showing that each option is on a separate line. It would be appropriate for creating a draft version of the volume.

```
memo
onerror=go
search=graphics/
search=entityFiles/
```

Before producing the final version of the help volume, you would remove the `memo` and `onerror=go` lines.

### See Also

- Chapter 13, "Preparing an Installation Package," explains which help files are included in your application installation package.

## ▼ To Review and Correct Parser Errors

♦ Look at the contents of the *volume*`.err` file after running HelpTag (where *volume* is the base name of your *volume*`.htg` file).

Each error listed in the *volume*`.err` file begins with a string of asterisks (`*****`). For example, the following error was detected at line 54 of the file `actions`:

```
*****
Line 54 of actions,
Missing end tag for LIST:
...the execution host becomes the current working directory.

<s2 id=EverythingYouNeedToKnow> E...
Current element is LIST begun on Line 28 of actions.
```

A few lines of the file are shown to give you some context for the error. Also, there is a hint that the current element is a LIST started on line 28 of the same file. An `<s2>` is not allowed within a list, so it appears that the author forgot to enter the `<\list>` end tag.

It's possible for a single, simple error to produce several error messages. This is because the first error may cause the parser to lose track of the intended context, making it impossible to interpret subsequent markup properly.

### Common Errors

Most processing errors result from these common mistakes:

- Omitting an end tag
- Using an incorrect entity name
- Referring to an invalid element ID

Omitting an end tag for an element is a common mistake. When creating elements, such as a list, figure, note, caution, or warning, be sure to include the end tag. Check your markup carefully especially if you have nested one element within another, such as a figure within a list,

Errors can also be introduced by using an incorrect entity name. In most instances, it is simply a misspelled word. In other cases, an entity name may have been changed, but cross-references to the original name were overlooked. When you change an entity name, remember to search your source file (or files) for all instances of the entity name.

Similarly, changing the ID assigned to an element affects any cross-reference or link to that topic.

## Viewing a Help Volume

The Help Viewer can be used to display any help volume. It supports all types of hyperlinks except application-defined links (because it cannot know how your links are to be interpreted).

If you are writing application help and your application is ready to use, you can also view your help by running your application, then requesting help just as a user would.

### ▼ To Display a Help Volume

1. Open File Manager and change to the directory where the *volume*`.sdl` file is located.
2. Double-click its icon.

   The default action displays the file using the Help Viewer.

### ▼ *To Run the dthelpview Command Manually*

♦ If the `MyVolume.sdl` file for the volume you want to display is either in the current directory or has been registered, execute this command:

```
dthelpview -helpVolume MyVolume.sdl
```

*Or,* if the `MyVolume.sdl` is in another directory (and hasn't been registered), execute this command:

```
dthelpview -helpVolume /full-path/MyVolume.sdl
```

The `-helpVolume` parameter can be shortened to `-h` in any of these commands.

#### Example

Suppose you just edited your help volume. First, process it with the HelpTag software:

```
dthelptag MyVolume
```

If no errors occurred, you could then display it with this command:

```
dthelpview -h MyVolume.sdl
```

#### See Also

- "Registering Your Application and Its Help" on page 249.

#### Example: A Personal Help Directory

During a project, you may want to access the help volume you are developing, but not expose it to all users on your system. For example, suppose your working directory is `/projects/help` and your help volume is named `MyVolume`.

First, create the personal help directory in your home directory where you can register the volume:

```
mkdir -p $HOME/.dt/help/C
```

Now create a symbolic link to the `MyVolume.sdl` file (which is created by the HelpTag software):

```
ln -s /projects/help/Myvolume.sdl $HOME/.dt/help/C/MyVolume.sdl
```

You can now display the volume with the following command (regardless of your current directory) because the `.dt/help/C` directory within your home directory is one of the first places the Help System looks for help volumes.

```
dthelpview -helpVolume MyVolume
```

## Adding Your Help to the Browser Volume

The desktop provides a special help volume called the browser volume that lists help volumes available on your system. The browser volume is displayed by clicking the Help Viewer control in the Front Panel.

You can view assorted help volumes directly from the browser volume. This allows access to application-specific help without starting the application. Or, if you are writing standalone help, this is the only way for users to get to your help.

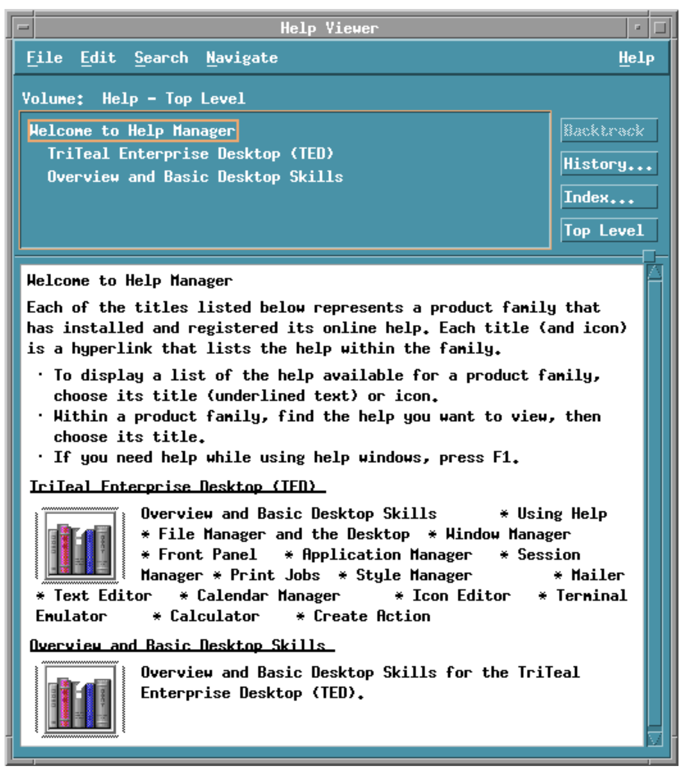



*Figure 4-1* Browser help volume displaying help families

To make your help volume available in the browser volume, you create a help family file. When your application is registered on the desktop, the presence of a family file causes the help volume to be included in the browser volume.

### Browser Volume

A desktop utility creates and updates the browser volume. When a user clicks on the Front Panel Help Viewer for the first time, the utility is automatically run. It identifies help volumes and help family files that are located in the help search path directories. It creates a file called `browser.hv` in the user's *HomeDirectory*`/.dt/help/$DTUSERSESSION` directory. After initial creation, the volume is updated only if changes have occurred.

To manually update the browser volume, refer to "Generating a Browser Help Volume (dthelpgen)" on page 195.

Any help volume listed in the browser volume can be viewed by selecting the volume title. Because you can display and navigate through different volumes, the browser help window includes an additional button, called Top Level. You can use this button to return to the browser list after displaying one or more volumes.

### Help Family File

The desktop utility examines help family files to identify which help volumes are gathered into the browser volume. Figure 4-1 on page 96 shows two help families, TriTeal Enterprise Desktop and Overview and Basic Desktop Skills, listed in the browser volume. Each family file consists of one or more related help volumes. For example, the TriTeal Enterprise Desktop family includes different volumes that describe the desktop.

Refer to the *Advanced User's and System Administrator's Guide* for a detailed explanation of how an application and its help files are installed on the desktop.

## ▼ To Create a Help Family

1. Pick a file name that is unique to your product. Use the `.hf` extension to identify the file as a help family.

   *family*`.hf`

2. Enter the following lines into the file:

   ```
   *.charSet:      character-set
   *.title:        family title
   *.bitmap:       icon file
   *.abstract:     family abstract
   *.volumes:      volume volume volume ...
   ```

   Where *character-set* specifies the character set used by the *family title* and *family abstract* strings. See “Understanding Font Schemes” on page 262 for a list of supported character sets. The *family title* and *family abstract* should not contain any HelpTag markup; this file is *not* processed with the HelpTag software.

   The *icon file* is optional. If you provide one, the path you use to specify the location of the file should be a complete path name. If you do not provide an icon, do not include the `*.bitmap` resource in your family file.

   The list of *volume* names identifies which volumes belong to the family. The volumes will be listed in the order they appear on this line. A volume may be listed in more than one family.

   If any of the values occupy more than one line, end each line — except the last — with a backslash (\).

   Any line in the file that begins with an `!` (exclamation mark) is a comment line and is ignored.

3. When you prepare your final product, you should install your *family*`.hf` file with the rest of your help files. When the desktop integration script, (`dtappintegrate`) is run, it creates the symbolic links to your family file.

   The *Advanced User’s and System Administrator’s Guide* describes how to run the `dtappintegrate` script.

### Example

Here’s a family file for the desktop’s online help. Comments at the top of the file identify the family and release version.

```
!##############################################
!#                                            #
!# TriTeal Enterprise Desktop Help Family #
!#                                            #
!#                Version 4.0                 #
!#                                            #
!##############################################
*.charSet:      ISO-8859-1
*.title: TriTeal Enterprise Desktop Version 4.0
*.bitmap:       /usr/dt/appconfig/help/C/tedlogo.pm
*.abstract:     Overview and Basic Desktop Skills \
                * File Manager and the Desktop \
                * Front Panel \
                * Application Manager \
                * Style Manager \
                * Text Editor \
                * Mailer

*.volumes: Intromgr.sdl Filemgr.sdl FPanel.sdl
           Appmanager.sdl Stylemgr.sdl
           Textedit.sdl Mailer.sdl
```

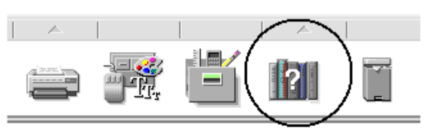
The help family file actually included with the desktop software may not exactly match this example.

### See Also

- "Character Sets and Multibyte Characters" on page 256 for a list of supported character set names.

## ▼ To Display the browser Volume

1. Choose the Help Viewer control from the desktop's Front Panel.



2. Scroll the help window to view the help families available on your system.
3. If desired, display a volume by selecting the help family title.

---
**Note –** To view help information about the Help System, choose the title TriTeal Enterprise Desktop and then Desktop Help System.

---

### ▼ *To Display the browser Volume Manually*

♦ Run the `dthelpview` command as follows:

```
dthelpview -helpVolume browser
```

#### See Also

- "Displaying Help Topics (dthelpview)" on page 194 lists `dthelpview` command line.
- `dthelpgen (1)` man page

## Printing Help Topics

After displaying your help volume, you can print help topics. Using the Print dialog box shown in Figure 4-2 you can print an individual topic, a table of contents and index information, or the entire help volume. Printed output omits graphics.

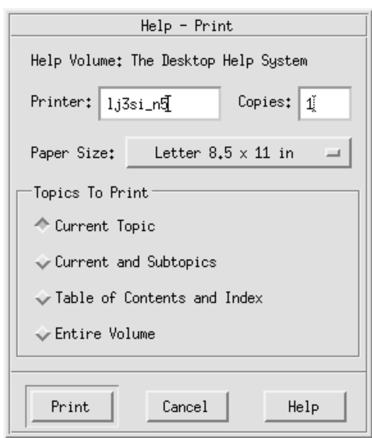



*Figure 4-2* Help print dialog box

# Testing Your Help

Testing your help volume is as important as testing any software product. Here are some tips to help you plan your testing.

## Validating Hyperlinks

- Display your help volume and try every hyperlink. Any underlined text (solid or dashed underlines) is a hyperlink. Also, test any graphics that are hyperlinks. Graphic hyperlinks use an open-cornered border (dashed or solid) around the image as a hyperlink cue.
- If you are writing application-specific help and you have included any `JumpNewView`, `Man`, or `AppDefined` links, you must test these links from your application. Testing such links using `dthelpview` does not ensure that the links will operate correctly from within your application.

## Verifying Entry Points

If you are writing application-specific help that uses IDs to access particular help topics, there are two ways to verify that the IDs have been properly established within the help volume:

- Run your application and request help just as a user will, trying each of the entry points. This also verifies that the application is using the correct IDs.
- If your application is not ready to use (still under development), you can test each ID by running `dthelpview` for each ID:

  `dthelpview -helpVolume` *volume*`.sdl -locationId` *id*

  Where *id* is the location ID that you want to test. If `dthelpview` displays the correct topic, then the ID is okay.

## Checking Index Entries

Users search or browse a help volume index to find help topics. Examine your index entries carefully to eliminate any vague terms or duplicate entries. Also select each index entry to verify that the topic displayed is the most appropriate information.

## Testing Graphics

- Physically run your application on various displays to verify that the graphics are acceptable on color, grayscale, and monochrome displays.
- You can also simulate other displays by changing the number of colors used by the desktop. To do so, open Style Manager, choose Number Of Colors, and select a different color option.

## Checking for Parser Errors

When developing a help volume, it is often convenient to set the `onerror=go` option in the `helptag.opt` file. If you have done this, you should remove the option and process your source files a final time to ensure that no errors are encountered.

### See Also

- "Generating a Browser Help Volume (dthelpgen)" on page 195.

# HelpTag Markup Reference 5

This chapter describes all of the HelpTag markup elements (and their associated tags) in alphabetical order. To help determine the name of a tag based on how it is used, the elements are grouped below according to use. (A few elements appear in more than one group.)

*Meta information (information about your volume):*

`<metainfo>`
`<title>`
`<copyright>`
`<abstract>`
`<otherfront>` (nonhierarchical topic)

*Structure of a help volume:*

`<!entity>`
`<helpvolume>`
`<hometopic>`
`<chapter>`
`<s1>` …`<s9>` (heading)
`<rsect>` (reference section)
`<otherhead>`
`<procedure>`
`<p>` (paragraph)

*Inline elements:*

<book>
<computer> (shorthand: "*text*")
<emph> (emphasis) (shorthand: !!*text*!!)
<ex> (example) and <vex> (verbatim example)
<image>
<keycap> (shorthand: [[ *text* ]])
<lineno> (line number)
<newline>
<p> (paragraph)
<quote> (directional quotes)
<sub> (subscript) (shorthand: _ _ *text* _ _)
<super> (superscript) (shorthand: ^^*text*^^)
<term> (shorthand: ++*text*++)
<user> (user input)
<var> (variable) (shorthand: %%*text*%%)
&...; (see <!entity> )

*Important information:*

<note>
<caution>
<warning>
<emph> (emphasis) (shorthand: !!*text*!!)

*Lists:*

<list>
<lablist> (labeled list)
<item> (shorthand: *)

*Graphics:*

<figure>
<graphic>

*Glossary and index:*

<glossary>
<dterm> (definition of term)
<term> (shorthand: ++*text*++)
<idx> (index)

*Cross-references and hyperlinks:*

`<xref>` (cross-reference)
`<link>`
`<location>`
`<term>`

*Hidden text:*

`<!--` … `-->` (comment)
`<memo>`

*Titles and headings:*

`<abbrev>`
`<head>`
`<otherhead>`
`<procedure>`
`<title>` (title of help volume)

*Override meaning of HelpTag markup:*

`<vex>` (verbatim example)

## <!-- ... -->

*Comment*

Identifies text you want the HelpTag software to ignore. Comments cannot be nested.

### Syntax

<!-- *comment text here* -->

The comment text can contain any text except two dashes (--).

### Example

The following markup hides both a comment and a figure:

```
<!-- Let's leave out this figure for now:
<figure entity=ProcessFlowChart>
Before and After Processing
<\figure>
-->
```

### See Also

- "<memo>" on page 154.

## <abbrev>

*Abbreviated title*

Indicates an alternate, typically shorter, heading for a topic that has a long title. When an abbreviated title is provided, it is used in the Index and History dialog boxes rather than the full title.

If a heading contains a graphical element, you must provide an `<abbrev>` that contains only the text of the heading. Although the graphic image can be displayed in the topic tree, the Index and History dialog boxes cannot display graphic elements.

An `<abbrev>` should not contain any markup.

### Syntax

```
<topic-element>  title
<abbrev>  short title
```

Where *topic-element* is `<hometopic>`, `<chapter>`, `<s1>`, or any other element that begins a new topic.

The `<abbrev>` tag must appear on the line immediately following the heading.

An end tag is not required.

### Examples

Here is a simple example:

```
<chapter> Ways of Treating Headings that are Too Long
<abbrev> Long Headings
```

Suppose you want to have a topic that doesn't have its title displayed in the help topic display area, but you do want a title to appear in the topic tree. The following markup shows how this can be done:

```
<chapter> &empty;
<abbrev>  chapter title
```

### See Also

- "<chapter>" on page 115.

## <abstract>

*Abstract*

Provides a short description of the help volume.

### Syntax

```
<metainfo>
    .
    .
    .
  <abstract>
  abstract text here ...
  <\abstract>
    .
    .
    .
<\metainfo>
```

The abstract text should not contain HelpTag markup because the abstract may be read and displayed by applications that don't recognize markup.

The `<abstract>` element is automatically assigned the ID string `_abstract`. An author-defined ID cannot be assigned. The `_abstract` ID can be used with the `<link>` element, but not with the `<xref>` element.

Abstract text may contain an optional `<head>`.

### Example

This markup briefly describes the contents of a help volume:

```
<abstract>
Online help for the Application Manager Version 1.0.
<\abstract>
```

### Note

When creating a link to an element within the `<metainfo>` element, be sure it is a `type=Definition` link.

The following markup shows how to create a link to the abstract:

```
<link hyperlink= "_abstract" type=Definition>
Choose this link for an abstract.<\link>
```

### See Also


- “<metainfo>” on page 155.
- “<head>” on page 131.

## <<*annotation text*>>

*Annotation*

Provides an explanatory note or comment within an example (`<ex>` tag).

### Syntax

`<ex [side | stack]>`
*text of the example ...*<<*annotation text* >>

<\ex>

*Where:*

| | |
|---|---|
| `side` | Default. Places the annotation to the right of the example text and on the same line as the first line of the example. |
| `stack` | Places the annotation below the example text. |

Enclose the text of an annotation in double angle brackets, as follows: << *this is the annotation text*>>. An annotation can only be used within an `<ex>` tag. The side and stack parameters of the `<ex>` tag can be used to position the annotation in relation to the example text.

To insert a blank line in an annotation, use a space followed by an empty annotation, *wordspace* <<>>.

### Example

The following markup uses the default side placement for the annotation:

```
<ex>
Login: <<Enter your name>>
<\ex>
```

It produces:

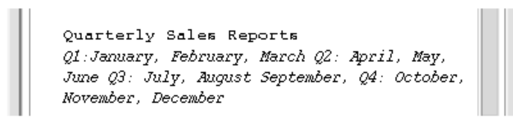
Login: *Enter your name*

The following markup uses the stack parameter to accommodate a long annotation:

```
<ex stack>
Quarterly Sales Reports

<<Q1: January, February, March Q2: April, May, June Q3: July, August,
September Q4: October, November, December>>
<\ex>
```

It produces:



## <book>

*Book title*

Identifies the title of a book.

### Syntax

`<book>`*book title*`<\book>`

*Or:*

`<book|`*book title*`|`

HelpTag formats book titles using an italic font.

### Example

Either of the following two variations:

```
Refer to <book>The Elements of Style<\book>
for further details.
```

*Or:*

```
Refer to <book|The Elements of Style|
for further details.
```

produce:

Refer to *The Elements of Style* for further details.

## <caution>

*Caution notice*

Specifies information that warns the user about a potential loss of data or hazard.

### Syntax

```
<caution>
text of caution
<\caution>
```

The default heading is "Caution". To specify a different heading, use the `<head>` tag as shown here:

```
<caution><head>alternate heading
text of caution
<\caution>
```

The `<\caution>` end tag is required.

To specify that an icon be displayed with the caution, define a file entity at the top of your help volume as follows:

```
<!entity CautionElementDefaultIconFile FILE "filename">
```

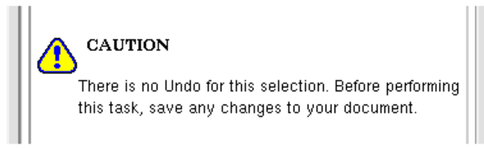
Where *filename* is the name of the icon graphic. A sample caution icon named `cauticon.pm` is provided in the `/usr/dt/dthelp/dthelptag/icons` directory.

### Example

Here is a caution message:

```
<caution>
There is no Undo for this selection. Before performing this task,
save any changes to your document.
<\caution>
```

The markup produces this output:




### See Also

- “<note>” on page 158 includes an example of changing a heading.
- “<warning>” on page 179.
- “<figure>” on page 125.
- “<head>” on page 131.

# <chapter>

*Chapter*

Indicates the start of a new topic with a new title.

## Syntax

```
<chapter id=id>title
topic text ...
```

An end tag is not required.

If the topic title is long, you may want to provide an alternate abbreviated title using `<abbrev>`. The short title is used in the Index and History dialog boxes. If the title contains a graphical element, create an `<abbrev>` with the title text only.

## Example

Here are two markups that begin a new topic:

```
<chapter>A Manual of Style
<chapter id=DesktopTools>Desktop Tools
```

## See Also


- “<abbrev>” on page 107.
- “<link>” on page 146.
- “<rsect>” on page 166.
- “<s1>…<s9>” on page 168.
- “<xref>” on page 180.

## <computer>

*Computer literal*

Displays text that represents computer input or output.

### Syntax

`<computer>`*text*`<\computer>`

*Or:*

*"text"*

The shorthand form uses two " (left apostrophes) and two "(right apostrophes).

### Examples

- The following markup:

  ```
  <computer>Enter the correct numerical value.<\computer>
  ```

  produces the following output:

  `Enter the correct numerical value.`

- The following markup uses the shorthand form:

  ```
  Everything in "computer" comes out looking "like this."
  ```

  and it produces:

  Everything in `computer` comes out looking `like this`.

- Variables can be nested within computer text. For example, this markup:

  ```
  "void DisplayTopic (%%topic%%);"
  ```

  produces:

  `void DisplayTopic (`*topic*`);`

### See Also


- "<ex>" on page 123.
- "<user>" on page 175.
- "<var>" on page 176.

# <copyright>

*Copyright notice*

Identifies text for the copyright notice.

## Syntax

```
<metainfo>
   <title>Title (always before copyright)
   <copyright>
   &copy;  Copyright notice here ...
```

This element is optional within the `<metainfo>` section. If used, it must follow the `<title>` element.

The end tag is not required.

The predefined entity `©` produces the copyright symbol (©).

## Example

The following markup assigns a title to the help volume and provides copyright information:

```
<metainfo>
<title>XYZ World Almanac
<copyright>
&copy; Copyright 1995 XYZ Company. All rights reserved.
```

It produces:

© Copyright 1995 XYZ Company. All rights reserved.

## See Also


- "<metainfo>" on page 155.
- "<title>" on page 174.

## <dterm>

*Defined term*

Identifies a term and the term's definition within the glossary.

### Syntax

```
<glossary>
  <dterm>first term
  definition of first term
    .
    .
    .
  <dterm>Nth term
  definition of Nth term
```

This element is used within the `<glossary>` section.

The name of the term follows the `<dterm>` tag and appears on the same line. The term's definition begins on the line following the `<dterm>` tag.

An end tag is not required.

### Example

The following markup defines the first two words in a glossary:

```
<glossary>

<dterm>algorithm
A mathematical rule or procedure for solving a problem.

<dterm>click
To press and release a mouse button.
```

### See Also


- "<glossary>" on page 128.
- "<term>" on page 172.

## <emph>

*Emphasized text*

Formats the text in a font that draws attention to the text.

### Syntax

`<emph>`*text*`<\emph>`

*Or:*

`!!`*text*`!!`

The shorthand form for the `<emph>` element is a set of double exclamation marks (`!!`) before and after the text.

If you use the `<emph>` start tag, the `<\emph>` end tag is required.

### Example

Either of the following two markups:

```
A thousand times <emph>no<\emph>.
A thousand times !!no!!.
```

produces:

A thousand times *no*.

### See Also


- “<book>” on page 112.
- “<var>” on page 176.

## <!entity>

*Entity declaration*

Assigns an entity name to a string of characters or to an external file.

### Syntax

`<!entity` *entityname* `"`*string*`">`

*Or:*

`<!entity` *entityname* `FILE "`*filename*`">`

An entity name can contain up to 64 letters, digits, and hyphens. Case is not significant in entity names, but is often used to improve readability for the author. The first character must be a letter. No space is permitted between the < ( left angle bracket), `!` (exclamation mark), and `entity` in an `<!entity>` declaration.

Entity declarations must always precede any other markup or text in the help volume.

Where you want the defined entity to appear, insert an entity reference using this syntax:

`&`*entityname*`;`

The entity reference consists of an `&` (ampersand), followed by the entity name (as defined in the entity declaration), and ends with a `;` (semicolon).

### Purposes for Entities

There are four common reasons for defining an entity:

- Text that is associated with an entity name appears only once so that changing the text requires making a change in only one place. All references to the entity automatically change when HelpTag reprocesses the files.
- The inefficiency of typing the same long or complex text string many times can be avoided (along with typing mistakes) by typing just a short entity reference wherever that text string will appear. The full text string needs to be typed only once.

- The <figure> and <graphic> elements do not accept a file name. The name of the file that contains the figure must be specified in an entity declaration.
- It is convenient to put the help text into multiple files, yet HelpTag accepts only one source file. These needs can be balanced by creating one file that contains entity declarations and entity references that refer to the files that contain the actual help text.

### Examples

- The *volume*.htg source file can contain the following entity declarations and entity references so that the actual text can be put into the named files:

```
<!entity topic1 FILE "topic1">
<!entity topic2 FILE "topic2">
<!entity topic3 FILE "topic3">

&topic1;
&topic2;
&topic3;
```

- The following entity declaration causes the words "Architectural Analysis of Aircraft Precision Components" to be displayed wherever the &apc; entity reference appears in the marked-up files.

```
<!entity apc "Architectural Analysis of Aircraft Precision
Components">
```

- The following entity declaration for a *figure* is placed at the beginning of the source file:

```
<!entity CloseUpFig FILE "figname.tif">
```

  and the figure would be inserted where the following markup appears:

```
<figure entity=CloseUpFig>
Close Up View
<\figure>
```

### See Also


- "Using Entities" on page 48.
- "<figure>" on page 125.
- "<xref>" on page 180.
- Chapter 6, "Summary of Special Character Entities".

## <esc>

*Escape*

Causes text to be passed directly to the run-time help file without being interpreted by HelpTag. In a customized application for example, an author could embed Semantic Delivery Language (SDL) markup in the help source file. The `<esc>` element prevents the SDL markup from being read by the HelpTag parser. When the help volume is displayed with the Help Viewer, the authored SDL markup is processed.

Do not use the `<esc>` tag to escape individual HelpTag symbols or markup examples. To display HelpTag symbols, such as < (left angle bracket), \ (backslash), or `&` (ampersand), precede each symbol with an ampersand. Use the `<vex>` element to provide HelpTag markup examples in a help volume.

### Syntax

`<esc>`*text*`<\esc>`

*Or:*

`<esc|`*text*`|`

---

**Note –** If the long form is used, the text cannot contain the three-character sequence `<\`*x* (the less-than symbol followed by a backslash followed by a letter). If the short form is used, the text cannot contain the | (vertical bar) character.

---

If you use the first syntax, the `<\esc>` end tag is required.

### See Also

- "Displaying HelpTag Symbols" on page 31.
- "<vex>" on page 177.

## <ex>

*Computer example*

Shows computer text without changing the spacing or line breaks.

### Syntax

`<ex [nonumber | number] [smaller | smallest] [side | stack]>`
*example text here ...*
`<\ex>`

Where:

| | |
|---|---|
| `nonumber` | (Default.) Omits the adding of line numbers to the beginning of each line. |
| `number` | Puts a line number at the beginning of each line. |
| `smaller` | Displays the example using smaller fonts. |
| `smallest` | Displays the example using smallest fonts. This makes long lines fit within a narrower width. |
| `side` | Applicable only when using an annotation within the example. Specifies the position of the annotation text in relation to the example text. The default position is `side`, which places the annotation to the right of the example text and on the same line as the first line of the example. |
| `stack` | Places the annotation below the example text. |

Examples are printed in `computer` font, and they are indented from the left text margin.

If you include the `number` attribute, the line numbers of the example will be numbered. This is useful for referring to specific lines.

The following character pairs, which have special meanings in other contexts, are treated as ordinary text within an example:

```
!!  double exclamation
--  double minus sign
++  double plus sign
"   double quote
```

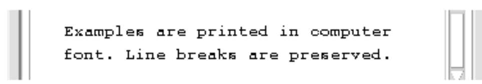
The `<\ex>` end tag is required.

### Example

The following markup:

```
<ex>
Examples are printed in computer
font. Line breaks are preserved.
<\ex>
```

produces:




### See Also

- “<computer>” on page 116.
- “<user>” on page 175.
- “<vex>” on page 177.

## <figure>

*Figure*

Inserts a graphical image.

### Syntax

<figure entity=*entity* [id=*id* [nonumber | number=*n*]
[left |center | right] [cappos=[capleft | capcenter | capright]]
[ghyperlink=*id* [glinktype=*type*] [gdescription=*text* ]]] >
*caption string*
<\figure>

| | |
|---|---|
| entity=*name* | Specifies a file entity which identifies the file that contains the graphic image to be inserted. |
| id=*name* | Optional. Defines an ID name that can be used in cross-references to this figure. |
| nonumber | Optional. Suppresses the word "Figure" and the automatically generated figure number. |
| number=*n* | Optional. Used to override the automatically generated figure number. |
| left, center, or right | Specifies horizontal alignment of the image within the current page width. |
| cappos=*position* | Specifies the horizontal alignment of the caption using the values capleft, capcenter or capright. A caption is optional. |
| ghyperlink="*id*" | Optional. Specifies that the graphic portion of the figure be a hyperlink. Follows the same usage as the *hyperlink* attribute in the <link> element. References to this location would use the specified *id* identifier. |
| glinktype=*type* | Optional. Specifies the type of hyperlink. The default type is Jump. Other type values include JumpNewView, Definition, Man, Execute, and AppDefined. The ghyperlink parameter and *id* value are required when using parameter. Follows the same usage as the *type* attribute in the <link> element. |

`gdescription="`*text*`"`
Optional. Provides a description of the hyperlink. The `ghyperlink` parameter and *id* value are required when using this parameter.

The `<\figure>` end tag is required.

To integrate an external graphics file into a help topic, you must have an entity declaration (`<!entity` *entityname* `FILE "`*filename*`">`) that associates the entity name with the graphic's file name.

### Examples

- The following markup inserts a graphic with the specified caption and an automatically generated figure number:

```
<!entity MapFigure FILE "worldmap.xwd">
   .
   .
   .
<figure entity=MapFigure>
Caption for Figure
<\figure>
```

- The following markup inserts a figure that is numbered but does not have a caption:

```
<!entity StateMap FILE "oregon.xwd">

   .
   .
   .
<figure entity=StateMap>
<\figure>
   .
   .
   .
```

- The following markup inserts a figure using a specific figure number and a caption. The caption is split into two lines where the \ (backslash) character appears:

```
<figure number=99 entity=SchemDiag>
Schematic that Illustrates\the Overall System Design
<\figure>
```

### See Also

- “<!entity>” on page 120.
- “<graphic>” on page 129.
- “<link>” on page 146.
- “<xref>” on page 180.
- “Execution Aliases” on page 74 provides information about using execution links.

## <glossary>

*Glossary*

Starts the glossary section which contains the definitions for all the terms that are marked with the `<term>` element.

### Syntax

```
<glossary>
<dterm>first term
definition of first term can continue over multiple lines or paragraphs

<dterm>second term
definition of second term ...
   .
   .
   .
```

"Glossary" is automatically used as the heading for the glossary section.

A `<dterm>` element identifies each term and its definition.

All terms marked with `<term>` without the `nogloss` parameter are required to be in the glossary. If the term is not in the glossary, omitted terms are listed in the *volume*`.err` file, which is created when you run HelpTag.

An end tag for `<glossary>` is not required.

### Example

Here is a simple glossary with two definitions:

```
<glossary>

<dterm>oxymoron
A combination of contradictory words.
<dterm>veritable
Being in fact the thing named. Authentic.
```

### See Also


- “<term>” on page 172.
- “<dterm>” on page 118.

## <graphic>

*Inline graphic*

Inserts a graphical element within a line of text.

### Syntax

`<graphic entity=`*name* [`id=`*id*]>

Where:

| | |
|---|---|
| *name* | An entity name that is defined in an entity declaration. The entity declaration associates the entity name with the name of the file that contains the graphic to be inserted. |
| `id=`*name* | Optional. Defines an ID name that can be used in cross-references to this figure. |

The `<graphic>` element is similar to `<figure>` except that the `<graphic>` element is intended for embedding *small* graphics within text, whereas the `<figure>` element inserts figures between paragraphs.

### Examples:

- The following markup first defines an entity (`mini-icon`) as being associated with the contents of a graphics file (named `mini.pm`). Then the `<graphic>` element indicates the location of the graphic within a line of text.

```
<!entity mini-icon FILE "mini.pm">
    .
    .
    .
The <graphic entity=mini-icon> icon is used to represent very
small images.
```

- The following markup first defines a topic whose ID is `mini-icon-topic`. It then shows how to use the inline graphic as a hyperlink to this topic.

```
<s1 id=mini-icon-topic>When you click on the inline graphic, it
will bring you to this topic.
.
.
.
The <link mini-icon-topic> <graphic entity=mini-icon> <\link>
icon is to represent very small things.
```

### See Also


- “<!entity>” on page 120.
- “<figure>” on page 125.
- “<link>” on page 146.
- “<p>” on page 162.

## <head>

*Heading*

Indicates the title for elements that normally do not have a title (such as `<abstract>`, `<paragraph>`, `<list>`, or `<otherfront>`) or have a default title (such as `<note>`, `<caution>`, and `<warning>`).

### Syntax

*<element>*`<head>`*title text*

A heading starts with the first nonblank character after the `<head>` tag. The `<head>` tag can appear on the same line as the element to which a heading is being added, or on the following line.

The `<head>` element can be used with elements that expect a title, but it is not required in those cases.

Headings that are wider than the heading area are automatically wrapped onto successive lines. To force a specific line break, put a \ (backslash) where you want the line to break.

A heading ends at the end of the line in the source file unless the line ends with an `&` (ampersand). If a heading spans multiple lines in your source file, put an ampersand after all the lines except the last.

The `<\head>` end tag is not required.

### Examples

- The following markup adds a title to a list and specifies the start of a new line where the \ (backslash) appears:

```
<list><head>Printing Options\for the QRZ Hardware
```

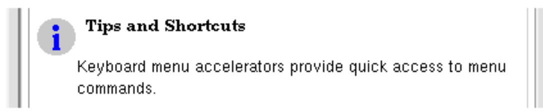
  It produces this output:

- The following markup overrides the default "Note" heading:

```
<note><head>Tips and Shortcuts

Keyboard menu accelerators provide quick access to menu commands.
<\note>
```

It produces this output:




### See Also


- “<abstract>” on page 108.
- “<caution>” on page 113.
- “<image>” on page 137.
- “<lablist>” on page 141.
- “<location>” on page 152.
- “<note>” on page 158.
- “<otherfront>” on page 159.
- “<p>” on page 162.
- “<warning>” on page 179.

## <helpvolume>

*Application help volume*

This is the "root" structural element; it contains all the markup for an entire help volume.

### Syntax

```
all entity declarations
    .
    .
    .
<helpvolume>
    .
    .
    .
   all of your help is included here, either
   literally or using file entity references
    .
    .
    .
<\helpvolume>
```

If you do not enter this tag, its presence is automatically assumed by the HelpTag software.

All entity declarations must appear before the `<helpvolume>` start tag.

### See Also


- “<abstract>” on page 108.
- “A Help Volume at a Glance” on page 31.
- “<!entity>” on page 120.
- “<hometopic>” on page 134.
- “<metainfo>” on page 155.

## <hometopic>

*"Home" or top-level help topic*

Identifies the start of the top-level help topic.

### Syntax

`<hometopic>`*heading*
*topic text begins here ...*

There is only one home topic for a help volume. It comes after the meta information (`<metainfo>`) and before the first `<chapter>` or `<s1>`.

The `<hometopic>` element does not support an author-defined ID. The HelpTag software assigns the predefined ID `_hometopic`.

To create a hyperlink to the home topic, use this syntax:

```
<link hyperlink= "_hometopic">…<\link>.
```

### Example

```
<hometopic>Welcome to Online Help
This is the home topic for the online help ...

<chapter>First Subtopic
This is the first subtopic ...

<chapter>Second Subtopic
This is the second subtopic ...
    .
    .
    .
```

### See Also


- "A Help Volume at a Glance" on page 31.
- "<link>" on page 146.
- "To Create a Home Topic" on page 41.
- "<metainfo>" on page 155.

## <idx>

*Index entry*

Defines an entry to appear in the help volume index.

### Syntax

`<idx>`*text*`<\idx>`

*Or:*

`<idx|`*text*`|`

*Or:*

`<idx>`*text*`<sort>`*sort key*`<\idx>`

Where:

*text* — The text string that appears in the keyword index.

*sort key* — An optional text string used when sorting the index. The *sort key* influences where the *text* appears in the keyword index. The *sort key* string does not appear in the keyword index.

Choosing the Index button in a general help dialog box displays a help index. Adding index entries to help topics is important because a user can search the index for a word or phrase to find help on a subject.

Either the `<idx>` start and end tags or the short form can be used.

The `<sort>` element changes the sort order for an index entry. Specifically, the `<sort>` element is used within the `<idx>` element to request that the keyword appear at the location indicated by the *sort key* string. No end tag for `<sort>` is required.

### Examples

- The following markup shows the definition of some simple index entries using the shortform. The index entries are indented to make the source text easier to read.

```
A portable personal computer has a full-sized keyboard, built-in
disk drives and a detachable LCD screen.
     <idx|keyboard|
     <idx|disk drive|
     <idx|screen, LCD|
     <idx|personal computer, portable|
     <idx|portable, personal computer|
```

- The following example displays "+" in the index, but it appears where "plus" would appear in the alphabetical list of entries.

```
<idx>+<sort>plus<\idx>
```

## <image>

*As-is image*

Shows text with the same line breaks as the source text.

### Syntax

```
<image [indent][id=id][gentity=graphic-ent [gposition=pos]
[ghyperlink=gid [glinktype=type]]]>
text
<\image>
```

Where:

| | |
|---|---|
| `indent` | Optional. Specifies that the paragraph be indented 6 spaces from the current left margin. |
| `id=`*id* | Optional. Defines an ID name that can be used in cross-references to this location. |
| `gentity=`*graphic-ent* | Optional. The name of a graphic entity around which the text is to be wrapped. The `gentity` parameter and *graphic-ent* value are required if the `gposition`, `ghyperlink`, or `glinktype` parameter is used. |
| `gposition=`*pos* | Optional. Either `left` or `right` to indicate whether the optional graphic is to be left-justified or right-justified. |
| `ghyperlink=`*gid* | Optional. Specifies that the graphic be a hyperlink and specifies the destination of the hyperlink. The `ghyperlink` parameter and *gid* value are required if the `glinktype` parameter is used. Follows the same usage as the *hyperlink* attribute in the `<link>` element. (The *id* value, not the *gid* value, would be used to reference the location of the image text.) |
| `glinktype=`*type* | Optional. Specifies the type of hyperlink. The default type is `Jump`. Other *type* values include `JumpNewView`, `Definition`, `Man`, `Execute`, and `AppDefined`. Follows the same usage as the *type* attribute in the `<link>` element. |
| *text* | The text of the paragraph that wraps around the graphic. |

Text between the `<image>` and `<\image>` tags is shown with the same spacing, indentation, and line breaks that appear in the actual text. No justification, word wrapping, or removal of empty lines is done. However, a proportional font is used, so columns of text that are lined up on a computer screen may not line up in the displayed help information. If the displayed text is too wide to fit within the display area, a horizontal scroll bar automatically appears.

All inline text elements and special characters are recognized.

An optional `<head>` can be used with `<image>`. If you intend to create a cross-reference to the element using `<xref>`, the `<head>` tag is required.

The `indent` parameter causes the displayed *text* to be indented from the left margin.

Either the start and end tags (`<image>` and `<\image>`) or the short form (`<image`|...|) can be used.

### See Also

- "<ex>" on page 123.
- "<vex>" on page 177.
- "<p>" on page 162.
- "Execution Aliases" on page 74 provides information about using execution links.

## <item>

*List item*

Identifies an item in a list.

### Syntax

```
<list [id=id]>
 * List item
 * List item
<\list>
Or:
<list order>
  <item id=name1>List item
  <item id=name2>List item
  <item id=name3>List item
    .
    .
    .
<\list>
```

The shorthand form, which is an `*` (asterisk), is almost always used.

The long form allows you to cross-reference an item in a list. You can only cross-reference items in an ordered (numbered) list. The automatically assigned item numbers are used in the cross-reference text (which HelpTag substitutes for the `<xref>` element). Unlike a number, a bullet character is not a meaningful substitution for the cross-reference text.

### See Also


- "<list>" on page 149.
- "<head>" on page 131.
- "<xref>" on page 180.

## <keycap>

*Keyboard keys*

Represents keyboard keys.

### Syntax

<keycap>*keycap characters*<\keycap>

*Or:*

```
[[keycap characters]]
```

The shorthand form is [[ (two left square brackets) and ]] (two right square brackets) before and after the keycap characters.

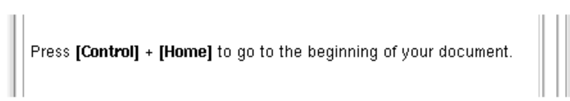
Entity references for special symbol characters, such as arrows, can be used. Multiline keycaps are not available.

### Example

The following markup:

```
Press [[Control]] + [[Home]] to go to the beginning of your document.
```

produces this output:




### See Also

- “<list>” on page 149.
- “<head>” on page 131.
- “<xref>” on page 180.

## <lablist>

*Labeled list*

Starts a labeled list in which the labels appear in the left column and the items (to which the labels refer) appear in the right column.

### Syntax

```
<lablist [loose | tight][wrap | nowrap]>
[ <labheads>  \Heading 1 \  Heading 2 ]
\label\  text for the first item
\label\  text for the second item
   .
   .
   .
<\lablist>
```

Where:

| | |
|---|---|
| `loose` | Default. Requests a vertical gap between the items in the list. |
| `tight` | Requests no extra vertical space between items in the list. |
| `wrap` | Default. Allows long labels to wrap to multiple lines. |
| `nowrap` | Prevents labels from wrapping to multiple lines. |

Backslashes (\) indicate the start and end of a label; leading and trailing spaces are ignored. Long labels are broken into multiple lines unless `nowrap` is used. The predefined character entity, (`&sigspace;`), can be used to insert a nonbreaking space into a label.

The text of the labeled item follows the second backslash, either on the same line or on the following line. The end of the item is indicated by one of the following:

- An empty line.
- Start of another labeled item.
- `<\lablist>` end tag.

If a labeled item consists of more than one paragraph, leave an empty line between the paragraphs. The end of the labeled list is indicated by the required `<\lablist>` end tag.

The optional column headings, one for each column, immediately follow the `<labheads>` tag (on the same line). The column headings are separated from one another by the \ (backslash). The `<\labheads>` end tag is not required. However, the `<\lablist>` end tag is required.

### Example

The following markup:

```
<lablist tight>
<labheads>  \ Unit \ Meaning
 \in\ inches
 \mm\ millimeters
 \cm\ centimeters
<\lablist>
```

produces this output:

| **Unit** | **Meaning** |
|---|---|
| in | inches |
| mm | millimeters |
| cm | centimeters |

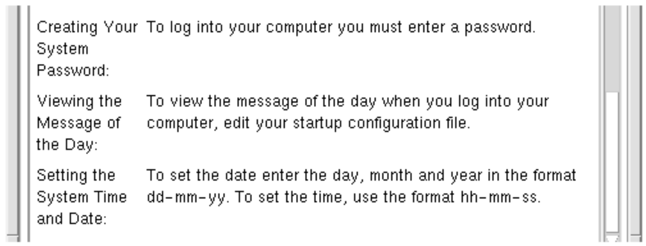
The following markup allows long labels to break into multiple lines.

```
<lablist>
\Creating Your System Password:\
To log into your computer, you must enter a password.

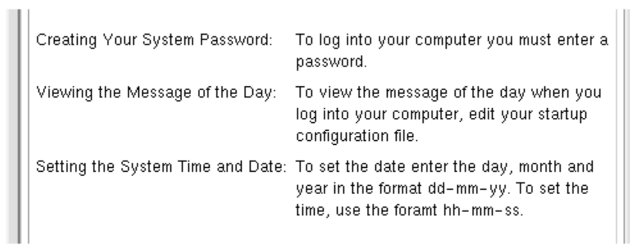
\Viewing the Message of the Day:\
To view the message of the day when you log into your computer, edit
your startup configuration file.

\Setting the System Time and Date:\
To set the date enter the day, month, and year in the format dd-mm-
yy. To set the time, use the format hh-mm-ss.
<\lablist>
```

It produces the following output:




Adding the `nowrap` parameter in the same markup produces this output:




### See Also

- “<head>” on page 131.
- “<list>” on page 149

## <lineno>

*Line number*

Provides a cross-reference to a specified line in an example.

### Syntax

```
<ex number>
example text <lineno id=name>
.
.
.
<\ex>
```

This element is used only in a numbered example. Enter the `<lineno>` tag at the end of the line you want to refer to. The `id` parameter assigns an ID that can be used to create a cross-reference to the line number.

### Example

This markup creates a numbered example that includes a cross-reference to the third line.

```
<ex number>
Enter Daily Account Total
Run Invoice Summary Report
Go to Monthly Ledger <lineno id=ledger>
Run Daily Update
<\ex>
.
.
.
To run closing reports, return to <xref ledger> and run the Past Due
Accounts Report.
```

The line number where the ID is located is substituted for the `<xref ledger>` cross-reference. It produces this sentence:

To run closing reports, return to 3 and run the Past Due Accounts Report.

The end tag is not required for `<lineno>`.

### See Also

- “<ex>” on page 123.

## <link>

*Hyperlink*

Delimits text or an inline `<graphic>` to be used as a hyperlink.

### Syntax

`<link` *hyperlink* `[`*type*`] ["`*description*`"]>`*text*`<\link>`

*Or*:

`<link hyperlink="`*hyperlink*`" [type=`*type*`] [description="`*description*`"]>`

The *hyperlink* attribute, which is required, is a value that identifies the destination or the behavior for the link. For a standard "jump" link, *hyperlink* is the ID of the element you want to jump to.

The *type* parameter can have the following values:

| | |
|---|---|
| `Jump` | Default. Jumps to the topic that contains the ID *hyperlink*. |
| `JumpNewView` | Jumps to the topic that contains the ID *hyperlink*, but requests that the hosting application display the topic in a new window. |
| `Definition` | Displays, in a temporary pop-up window, the topic that contains the ID *hyperlink*. |
| `Execute` | Executes the *hyperlink* string as a command. |
| `Man` | Displays a man page using the *hyperlink* string as the parameter to the `man` command. |
| `AppDefined` | Sends the *hyperlink* string to the hosting application for special processing. |

The *text* between the start and end tag becomes the "hot spot" that the user will choose to invoke the link. Any word or phrase used as a hyperlink is underlined when displayed. Capitalization is not significant for the *hyperlink* and *type* values.

A hyperlink that executes a command is called an execution link. The command to be executed can be included in the `<link>` command or defined as an execution alias, which is a type of resource. For information about using execution links, see “Execution Link Control” on page 74.

### Notes

- Avoid using the *type* keywords (listed above) as values for *hyperlink*. If you must do so, explicitly identify the parameters as shown in the second syntax line.
- The `<link>` element is not needed in a cross-reference that uses the `<xref>` element because a hyperlink is automatically created where the `<xref>` element is used.

### Examples

- The following markup defines a simple hyperlink to a topic with the ID `Intro`. Notice that capitalization of the ID is not significant.

```
<s1 id=Intro>Introducing the Desktop
.
.
.
Refer to the <link intro>Introduction<\link>.
```

- The following markup defines the same hyperlink jump as in the previous example but the `<link>` element is not used because a cross-reference (`<xref…>`) is automatically a hyperlink. In this case, the title of the Intro topic is automatically supplied by HelpTag.

```
Refer to <xref intro>.
```

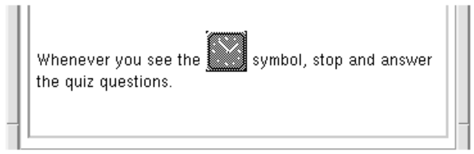
  This markup produces this output:

```
Refer to Introducing the Desktop.
```

- The following markup defines a hyperlink that is activated when the inline graphic is chosen. A new window is opened to display the "clockfeatures" topic.

```
Whenever you see the <link clockfeatures JumpNewView>
<graphic entity=StopWatchIcon><\link> symbol, stop and answer the
quiz questions.
```

It produces this output:




- The following markup creates a link that displays the man page for the `grep` command:

```
For more details, refer to the <link grep Man>grep man
page<\link>.
```

- The following markup creates an execution link using an execution alias named `startDtterm`. The alias name and the command it executes are defined in the application's application defaults file.

```
To open a terminal window, click <link hyperlink=”DtHelpExecAlias
startDtterm” Execute>Start Terminal Emulator.<\link>
```

For information about execution aliases and how to define them, see “Execution Aliases” on page 74.

### See Also

- “<figure>” on page 125.
- “<hometopic>” on page 134.
- “<idx>” on page 135.
- “<image>” on page 137.
- “<location>” on page 152.
- “<xref>” on page 180.
- “Execution Link Control” on page 74.

## <list>

*List*

Starts a list consisting of items that are optionally marked with bullets or automatically generated numbers or letters.

### Syntax

```
<list [bullet | order | plain] [loose | tight][continue]
[lalpha |ualpha | lroman | uroman | arabic] >

  * first item
  * second item
    .
    .
    .
<\list>
```

Where:

| | |
|---|---|
| `bullet` | Default. Displays a bullet before each item. |
| `order` | Displays a number in front of each item. The numbers are automatically generated and begin with the number one. The default is Arabic numbers. Ordered lists can also use alphabetical sequences or Roman numerals. |
| `plain` | Does not put a bullet, number, or letter in front of each item. |
| `continue` | Requests that the numbering of items continue from the previous list. |
| `loose` | Default. Requests a vertical gap between the items. |
| `tight` | Requests no extra vertical spacing between the items. |
| `lalpha` | Lowercase alphabet. |
| `ualpha` | Uppercase alphabet. |
| `lroman` | Lowercase Roman numeral. |
| `uroman` | Uppercase Roman numeral. |
| `arabic` | Default for `order` list type. |

Each item must start on a new line preceded by either an asterisk (*) or the <item> tag. The asterisk is the shorthand form of the <item> tag. Spaces and tabs may appear on either side of the asterisk. Items may continue over multiple lines. An item can consist of multiple paragraphs, in which case an empty line must separate the paragraphs. The nesting of lists is allowed, so a list can appear within a list.

The <\list> end tag is required.

### Examples

The following markup examples:

```
<list>
* chocolate
* raspberry
* vanilla
<\list>
```

```
<list plain tight>
* Word Processing
* Graphics
* Printing
<\list>
```

```
<list order lalpha>
* Word Processing
* Graphics
* Printing
<\list>
```

produce:


## See Also


- “<item>” on page 139.
- “<lablist>” on page 141.
- “<head>” on page 131.

## <location>

*Location*

Defines an ID as referring to the location of the `<location>` element. The `<location>` element enables a portion of a topic to serve as a destination for a hyperlink using the `<link>` or `<xref>` element.

### Syntax

`<location id=`*id*`>`*text*`<\location>`

*Or:*

`<location id=`*id*|*text*|

Where:

*id* The identifier for the current location, which can be used as a destination for hyperlinks.

*text* The block of text where you want to assign the ID.

The `<location>` element is not needed at locations where there is already an element (such as `<hometopic>` or `<figure>`) that has a built-in ID or accommodates an author-defined `id` parameter.

Cross-references created with the `<xref>` element substitute the text between the `<location>` start and end tag for the `<xref>` element.

### Examples

The following markup names a location and elsewhere creates a hyperlink to the location.

```
<s1 id=ConfigTopic> Configuration
    ...
<location id=ConfigTopicBody>some text<\location>
    ...
<s1 id=UseTopic> Usage
    ...
See <link ConfigTopicBody>Configuration<\link>
for additional information.
```

The advantage of linking to the ID in the `<location>` element is that the help window automatically scrolls to the point where the `<location>` tag is entered. In contrast, a link to the topic's ID ("ConfigTopic" in this case), always goes to the top of the topic.

The `<location>` element can also reference a position in your file using the predefined entity, (`∅`), as a placeholder.

Adding this markup at a key position in your file, allows you to create a link to that specific location:

*paragraph text*

```
.
.
.
<location id=pointA>&empty;<\location>
.
.
.
```

### See Also

- “<link>” on page 146.
- “<xref>” on page 180.

## <memo>

*Memo*

Identifies a writer's comments or questions, which do not appear in the final help volume.

### Syntax

```
<memo>
memo text
<\memo>
```

*Or:*

```
<memo|memo text|
```

Memo text is printed in drafts of your help volume *if* you specify `memo` in the `helptag.opt` file. Otherwise, memo text is not printed, especially when you create the final version of the help volume. Memo text, when it appears, is printed in a different typeface. Do not use markup within memo text.

### Examples

Here is an example of a memo:

```
<memo>
Patti: We need a drawing to illustrate this.
<\memo>
```

The following markup uses the short form of the `<memo>` element:

```
<memo|Mike: Please explain how the following
command is supposed to work|
```

### See Also

- "To Insert a Writer's Memo" on page 84.
- Sample `helptag.opt` file on page 93.

## <metainfo>

*Meta information*

Starts the meta information section, which contains information about the information contained in the help volume. Meta information includes the volume's title and a copyright notice.

### Syntax

```
<helpvolume>
  <metainfo>
    <title>volume title
    <copyright>
    &copy;  Copyright XYZ Company 1995...
    <abstract>
    brief description of help volume
   .
   .
   .
  <\metainfo>
<hometopic> ...
   .
   .
   .
```

The meta information section is optional, but it is typically included in a help volume. Although optional, the title, copyright, and abstract subsections provide useful information about your help volume and are recommended.

If you include any of these subsections, the meta information section is required.

The `<otherfront>` element can be used to define subsections other than the predefined title, copyright, and abstract subsections.

The `<\metainfo>` end tag is required.

### Example

```
<metainfo>

<title>Inventory Tracking Software

<copyright>
```

```
&copy; Copyright 1995 XYZ Company.
All rights reserved.

<abstract>
Explains how to use the Inventory Tracking Software

<\metainfo>
```

### See Also


- “<title>” on page 174.
- “<copyright>” on page 117.
- “<abstract>” on page 108.
- “<otherfront>” on page 159.

## <newline>

*New line*

Starts a new line within a paragraph or annotation.

### Syntax

*text*`<newline>`*text on next line*

Text that follows the `<newline>` element begins on a new line.

### Example

The following markup ensures that the path name begins on a new line:

```
Put your files for the manual in the special directory
<newline>/projects/userguide/draftdoc.
```

### See Also


- "<vex>" on page 177.
- "<ex>" on page 123.
- "<image>" on page 137.

## <note>

*Note*

Creates a special format which attracts attention to text that makes an important point.

### Syntax

```
<note>
text of note
<\note>
```

The default heading for the note is "Note". To specify a different heading, use the `<head>` element.

If you want an icon to appear with the note, define `NoteElementDefaultIconFile` in an `<!entity` ...> declaration.

The default note icon named note `icon.pm` is located in the `/usr/dt/dthelp/dthelptag/icons` directory.

The `<\note>` end tag is required.

### Examples

- The following markup uses the default heading:

```
<note>
Warranty information is in your installation manual.
<\note>
```

- The following markup specifies a different heading:

```
<note><head>Read This First
Warranty information is in your installation manual.
<\note>
```

### See Also


- “<caution>” on page 113.
- “<warning>” on page 179.
- “<head>” on page 131.

# <otherfront>

*Other meta information (front matter)*

Used for meta information (front matter) that does not fit within one of the predefined categories such as title, copyright, and abstract. The `<otherfront>` element can also be used to create a nonhierarchical topic. Because a nonhierarchical topic does not appear in the topic tree, a hyperlink must be added to display the topic. The `<link>` or `<xref>` element can be used to create a hyperlink to the `<otherfront>` element.

## Syntax

```
<metainfo>
    .
    .
    .
<otherfront [id=id]><head>title of section
  text
```

If a heading is needed, use the `<head>` element.

`<otherfront>` must follow all other subsections of `<metainfo>`.

## See Also


- "<metainfo>" on page 155.
- "<head>" on page 131.

## <otherhead>

*Other heading*

Creates a subheading within a topic.

### Syntax

`<otherhead>`*heading*

Headings may occur anywhere within the text of a topic. The `<otherhead>` element does not appear in the list of help topics displayed in the topic tree.

The `<\otherhead>` end tag is not required.

### Example

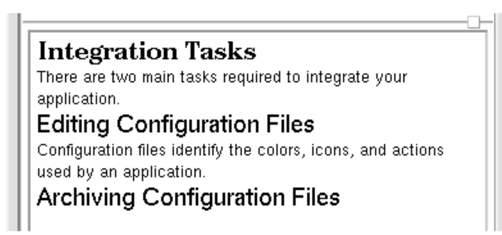
Here is an example in which `<otherhead>` elements identify two subsections within an `<s1>` topic:

```
<s1>Integration Tasks
There are two main tasks required to integrate your application.

<otherhead> Editing Configuration Files
Configuration files identify the colors, icons, and actions used by
an application.

<otherhead> Archiving Configuration Files
.
.
.
```

This markup produces:



### See Also


- “<head>” on page 131.
- “<procedure>” on page 164.
- “<rsect>” on page 166.
- “<s1>…<s9>” on page 168.

## <p>

*New paragraph*

Starts a paragraph that is indented or wrapped around a graphic.

### Syntax

<p [indent] [gentity=*graphic-ent* [gposition=*pos*]
[ghyperlink=*gid* [glinktype=*type*]]] [id=*id*] >*text...*

Where:

| | |
|---|---|
| `indent` | Optional. Specifies that the paragraph be indented 6 spaces from the current left margin. |
| `gentity=`*graphic-ent* | Optional. The name of a graphic entity around which the paragraph is to be wrapped. The `gentity` parameter and *graphic-ent* value are required if the `gposition`, `ghyperlink`, or `glinktype` parameter is used. |
| `gposition=`*pos* | Optional. Either `left` or `right` to indicate whether the optional graphic is to be left-justified or right-justified. |
| `ghyperlink=`*gid* | Optional. Specifies that the graphic be a hyperlink and specifies the destination of the hyperlink. The `ghyperlink` parameter and *gid* value are required if the `glinktype` parameter is used. Follows the same usage as the *hyperlink* attribute in the `<link>` element. (The *id* value, not the *gid* value, would be used to reference this paragraph's location.) |
| `glinktype=`*type* | Optional. Specifies the type of hyperlink. The default type is `Jump`. Other *type* values include `JumpNewView`, `Definition`, `Man`, `Execute`, and `AppDefined`. Follows the same usage as the *type* attribute in the `<link>` element. |
| `id=`*id* | Optional. Defines an ID name that can be used in cross-references to this location. |
| *text* | The text of the paragraph that wraps around the graphic. |

Use the `<p>` element if you need to indent a paragraph, wrap the paragraph around a graphic, or use a run-in head style paragraph.

An optional <head> can be used with <p>. If you intend to create a cross-reference to the element using <xref>, a <head> tag is required. Use the <head> and <\head> tags to delimit the heading text.

A <\p> end tag is not required.

### Examples

- Here are two paragraphs, the second of which is indented:

```
Some people do not like to read instruction manuals.
<p indent>This is not always a good idea.
```

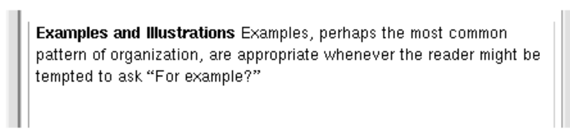
produces:

Some people do not like to read instruction manuals.
    This is not always a good idea.

- This markup creates a paragraph style with a run-in head.

```
<p><head>Examples and Illustrations <\head>
Examples, perhaps the most common pattern of organization, are
appropriate whenever the reader might be tempted to ask <quote>
For example?<\quote>
```

It produces this output:



**Examples and Illustrations** Examples, perhaps the most common pattern of organization, are appropriate whenever the reader might be tempted to ask “For example?”

### See Also

- “<head>” on page 131
- “<procedure>” on page 164
- “<s1>…<s9>” on page 168
- “To Wrap Text around a Graphic” on page 80
- “Execution Aliases” on page 74 provides information about using execution links

## <procedure>

*Procedure*

Starts a section within a topic.

### Syntax

`<procedure>`*heading*
*procedure text...*

Procedures may occur anywhere within the text of a topic. They are not included in the list of topics displayed in the topic tree.

The end tag is not needed.

### Example

This markup:

```
<procedure> Entering Special Characters

To enter Greek or mathematical characters in your document, use the
Symbols font.
```

produces this output:


### See Also


- “<head>” on page 131.
- “<otherhead>” on page 160.
- “<s1>…<s9>” on page 168.

## <quote>

*Quote*

Puts text within double quotation marks using open and close quote characters.

### Syntax

`<quote>`*text*`<\quote>`

*Or:*

*"text"*

Use the start and end tags (`<quote>`…`<\quote>`) or a pair of double quotation marks (`"…"`) to delimit the text.

### Example

The following markup:

```
... referred to in this manual as "the Standard" ...
```

produces:

…referred to in this manual as “the Standard”…

### See Also


- “<book>” on page 112.
- “<computer>” on page 116.
- “<var>” on page 176.

## <rsect>

*Reference section*

Identifies an entry in the reference section.

### Syntax

```
<rsect [id=id]>reference section heading
   .
   .
   .
<rsub>reference subsection heading
```

The `<rsect>` element can be used to identify a reference section. It is useful to identify reference material that is presented in a series of similar sections. For example, each reference section could describe one software command.

An `<rsect>` consists of:

- Required heading
- Optional introductory text
- Optional reference subsections (`<rsub>`)

Each `<rsect>` section can have multiple `<rsub>` sections. Each `<rsub>` element must have a heading. A cross-reference to a reference subsection is not allowed.

The topic tree includes `<rsect>` headings but excludes `<rsub>` headings.

End tags (for either `<rsect>` or `<rsub>`) are not required.

### Example

The following markup illustrates the use of this element:

```
<rsect>purge
   .
   .
   .
<rsub>Syntax
purge filename

<rsub>Example
purge file01
```

```
<rsub>Related Commands
delete
```

## See Also

- “<chapter>” on page 115.
- “<s1>…<s9>” on page 168.

## <s1>…<s9>

*Subsection (*`<s1>`, `<s2>`, ... , `<s9>`*)*

Starts a topic in the hierarchy.

### Syntax

`<sn [id=name]>`*heading*
*topic text*...

Where *n* is the level number (1, 2,…, or 9).

Topics entered with `<chapter>` can have subtopics entered with `<s1>`, `<s1>` topics can have `<s2>` subtopics, and so on. You *cannot* skip a level.

The heading for a section can be on the same line as the `<sn>` tag or on the next line; a heading is required. Text within a section is optional.

The end tag is usually omitted, but in some instances the end tag may be necessary. For example, when a section is followed by an `<rsect>` element that is on the same level, an end tag for the section is required. Without the end tag, the `<rsect>` element would be considered a subsection of the section preceding it.

### Examples

- The following illustrates a three-level hierarchy within a topic.

```
<chapter>Running the Processor
topic text...

<s1>Getting Started
To run the program, type in the usercode and your password.

<s1>Customizing
You may now set up this conversion program to change your computer from beige to red.

<s2>Configuration
Use either the disk drive or the tape drive to archive your files.

<s3>Disk Drive Advantages
See data sheet for specifications.

<s3>Tape Drive Advantages
```

*See data sheet for specifications.*

```
<s2>Support
```
*If you really need help, call technical support.*

- In the following markup, a section end tag (`<\s1>`) is used to make the `<rsect>` section be at the same level in the hierarchy.

  `<s1>`*first-level heading*
  *text*

  `<s1>`*first-level heading*
  *text*
  `<\s1>`
  `<rsect>`*first-level heading*
  *text*

  In contrast, leaving out the end tag causes the `<rsect>` section to become a subtopic of the second `<s1>` section:

  `<s1>`*first-level heading*
  *text*

  `<s1>`*first-level heading*
  *text*

  `<rsect>`*second- level heading*
  *text*

### See Also


- “<chapter>” on page 115.
- “<head>” on page 131.
- “<rsect>” on page 166.

## <sub>

*Subscript*

Creates a subscript character.

### Syntax

`<sub>`*character to subscript*`<\sub>`

*Or:*

`__text__`

The shorthand form uses two __ (underscore) characters before and after the characters to subscript.

The `<\sub>` end tag is required.

### Example

The following markup:

```
<p>The chemical element H<sub>2<\sub>O contains
two hydrogen molecules.
```

produces the following output:

The chemical element $H_2O$ contains two hydrogen molecules.

### See Also

- “<super>” on page 171.

## <super>

*Superscript*

Creates a superscript character.

### Syntax

<super>*character to superscript*<\super>

*Or:*

```
^^text^^
```

The shorthand form uses two ^^ (caret) characters before and after the characters to superscript.

The <\super> end tag is required.

### Example

The following markup:

```
<p>The answer to the problem is 2<super>8<\super>.
```

produces this output:

The answer to the problem is $2^{8}$.

### See Also

- "<sub>" on page 170.

## <term>

*Glossary term*

Writes a newly introduced term in a special font and establishes a hyperlink to its definition in the glossary.

### Syntax

<term *baseform* [gloss | nogloss]>*text*<\term>

*Or:*

<term *baseform* [gloss | nogloss]|*text*|

*Or:*

++*text*++

Where:

| | |
|---|---|
| *baseform* | The form of the term as it appears in the glossary if it is not the same as used in the text. This difference can occur, for example, when the term is used in the text in its plural form but appears in the glossary in its singular form. If the term includes spaces or special characters, put the *baseform* string in quotes. |
| `gloss` | Default. Requests that HelpTag verify that the term is in the glossary. |
| `nogloss` | Omits the term from the glossary; however, the term is formatted in a bold font. |

The shorthand form uses two ++ (plus signs) before and after the glossary term.

---

**Note –** If your help volume does not include a glossary, use the `nogloss` parameter.

---

When HelpTag processes the help volume, warning messages are issued to indicate glossary terms that were not marked with the `nogloss` parameter and do not have corresponding definitions in the glossary.

Tagging a term with the `<term>` element automatically creates a hyperlink to the glossary. If there is no glossary, the link will not work.

A <\term> end tag is required if the long form is used.

### Example

The following markup puts "structural elements" in a special font (boldface with a dotted underscore) to indicate it is a glossary term and creates a hyperlink to the glossary. Because the glossary entry contains a space, the text is in quotes. The plural form appears in the text. HelpTag checks for the singular form in the glossary and reports an error if it is not found.

```
SGML views a document as a hierarchy
of <term "structural element"|structural elements|.
```

### See Also

- “<glossary>” on page 128.
- “<dterm>” on page 118.

## <title>

*Help volume title*

Specifies the title of the help volume.

### Syntax

```
<metainfo>
<title>help volume title
```

The `<title>` element is an optional element in the `<metainfo>` (meta information) section. It is recommended, however, because the title provides the volume name displayed in the help dialog boxes.

The `<title>` follows immediately after the `<metainfo>` tag. Because the title of the volume may be displayed by other applications (information viewers, for example) that may not be able to format the title, you should use only plain text within the title.

The `<\title>` end tag is not required.

### Example

Here is a sample volume title:

```
<metainfo>
<title>The Super Hyperlink User's Guide
```

### See Also

- "<metainfo>" on page 155.

## <user>

*User's response*

Indicates the user's response to a computer prompt.

### Syntax

`<user>`*response*`<\user>`

*Or:*

`<user`|*response*|

This element is used to distinguish user input from computer output in a computer dialog. It is typically used within the `<ex>` element, where spaces and line breaks between the `<user>` start tag and the `<\user>` end tag are significant.

If used within a paragraph, `<user>` text must not break across lines in your source file.

The `<\user>` end tag is required if the long form is used.

### Example

The following markup produces two different fonts, one to indicate what the computer displays and another to indicate what the user types:

```
<ex>
Do you wish to continue? (Yes or No) <user>Yes<\user>
<\ex>
```

The output looks like this:

Do you wish to continue? (Yes or No) **Yes**

### See Also

- "<computer>" on page 116.
- "<ex>" on page 123.
- "<vex>" on page 177.

## <var>

*Variable*

Indicates a user-supplied variable in a command.

### Syntax

```
<var>
text
<\var>
```

*Or:*

```
%%text%%
```

The `<\var>` end tag is required if the long form is used.

The shorthand form uses two `%%` (percent signs) before and after the text.

### Example

These markups:

```
INPUT <var>filename<\var>
```

*Or:*

```
INPUT %%filename%%
```

produce:

INPUT *filename*

### See Also

- "<ex>" on page 123.
- "<computer>" on page 116.

## <vex>

*Verbatim example*

Indicates a verbatim example in which HelpTag elements are not interpreted as elements.

### Syntax

`<vex [number | nonumber][smaller | smallest]>` *text*`<\vex>`

Where:

| | |
|---|---|
| `nonumber` | Default. Omits line numbers. |
| `number` | Puts a line number at the beginning of each line. |
| `smaller` or `smallest` | Displays the example using smaller fonts.This makes long lines fit within a narrower width. |

Within a verbatim example, no HelpTag elements are recognized except `<\`, which is assumed to be an end tag.

Use this element when you need to display markup tags or other characters that could otherwise be interpreted as markup. Line breaks and spacing are preserved as they appear in the source file.

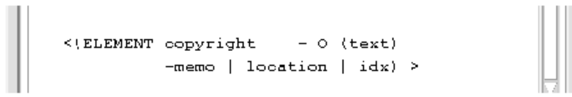
The `smaller` and `smallest` fonts enable wide examples to fit within the margins.

### Example

The following markup:

```
<vex>
<!ELEMENT copyright    - O (text)
         -memo | location | idx) >
<\vex>
```

produces this output:




### See Also


- "<ex>" on page 123.
- "<image>" on page 137.

## <warning>

*Warning*

Calls the user's attention to a situation that could be dangerous to the user.

### Syntax

`<warning>`

*text*

`<\warning>`

The text of the warning message is printed in boldface.

The default heading for the warning is "Warning". To specify a different heading, use the `<head>` element.

To display a graphic with the warning, define `WarningElementDefaultIconFile` in an `<!entity>` declaration. The default warning icon named `warnicon.pm` is located in the `/usr/dt/dthelp/dthelptag/icons` directory.

The `<\warning>` end tag is required.

### Example

- The following markup creates a warning message:

  ```
  <warning>
  Failure to follow these guidelines could result
  in serious consequences.
  <\warning>
  ```

- The following markup specifies a different heading for the warning message:

  ```
  <warning><head>Danger!
  Do not open the high-voltage compartment.
  <\warning>
  ```

### See Also


- "<note>" on page 158.
- "<caution>" on page 113.
- "<head>" on page 131.

## <xref>

*Cross-reference*

Inserts text that identifies another location in the help volume and creates a hyperlink to that location.

### Syntax

`<xref` *id*`>`

Where:

*id* is the identifier of the topic or location that is being cross-referenced.

Cross-references are translated into chapter or section titles, heads, figure captions, list items, or line numbers. The cross-reference text becomes a hyperlink that, when chosen by a user, jumps to the cross-referenced location.

To create a cross-reference, an `id` must be defined in the element that you intend to refer to. You use the `id` of the destination element as the *id* parameter in the `<xref>` tag. This creates a hyperlink from the `<xref>` element to the destination element. The `id` must be spelled exactly the same. Capitalization, however, is not significant.

The `id` parameter can appear with:

```
<chapter>
<s1>, <s2>,…<s9>
<otherfront>
<p>
<image>
<item>
<figure>
<location>
<rsect>
```

A cross-reference to an *id* that contains an underscore (such as "_abstract" or "_hometopic") is not allowed. Instead, use the `<link>` element.

## Examples

To refer a reader to a chapter for more information, use the chapter's id to create a reference. For instance, to refer to a chapter titled "Window Management" whose id is *windowmgr*, you would write, "Refer to <xref windowmgr> for details."

In your online help volume, the result would be: "Refer to Window Management for details." The chapter title, "Window Management", is substituted for the id and is a hyperlink.

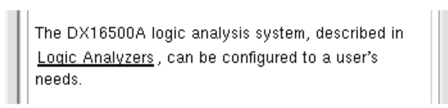
The following markup assigns an id named "analyzer" to a section element:

```
<s1 id=analyzer>Logic Analyzers
```

Here is markup that contains a cross-reference to this topic:

```
The DX16500A logic analysis system, described in
<xref analyzer>, can be configured to a user's needs.
```

After processing, the <xref> element would be replaced by "Logic Analyzers" as shown here:




The text "Logic Analyzers" is a hyperlink that, when chosen by a user, jumps to the cross-referenced help topic.

## See Also


- "<chapter>" on page 115.
- "<!entity>" on page 120.
- "<figure>" on page 125.
- "<graphic>" on page 129.
- "<image>" on page 137.
- "<link>" on page 146.
- "<location>" on page 152.
- "<otherfront>" on page 159.
- "<p>" on page 162.
- "<rsect>" on page 166.
- "<s1>…<s9>" on page 168.

# Summary of Special Character Entities 6

This chapter provides a list of special characters that can be used when writing a help topic. The following special characters can be inserted into text by typing the associated entity name in the position where the special character is to appear.

To use any of the entities whose description is marked with an * (asterisk), you must use the `helpchar.ent` file (see “Including Special Characters” on page 82).

*Table 6-1* Typographical Symbols

| Symbol | Entity Name | Description |
|---|---|---|
| © | `©` | Copyright symbol |
| ® | `®` | Registered symbol |
| ™ | `&tm;` | Trademark symbol |
| – | `&endash;` | En dash (short dash) |
| — | `&emdash;` | Em dash (long dash) |
| • | `•` | * Bullet |
| ↵ | `&cr;` | Carriage return |
| … | `&ellipsis;` | Ellipsis (horizontal) |
| …. | `&pellipsis;` | Ellipsis (end-of-sentence) |
| ⋮ | `&vellipsis;` | Vertical ellipsis |

*Table 6-1* Typographical Symbols *(Continued)*

| Symbol | Entity Name | Description |
|---|---|---|
| ' | `&squote;` | Single quote |
| " | `&dquote;` | Double quote |
| ⊔ | `&vblank;` | Vertical blank |
| ( ) | `∅` | Empty (no text) |
| ( ) | `&sigspace;` | Significant space |
| - | `&sigdash;` | Non line-breaking hyphen |
| § | `&S;` | * Section |
| ¶ | `&P;` | * Paragraph |

* To use any of the entities whose description is marked with an * (asterisk), you must use the `helpchar.ent` file (see "Including Special Characters" on page 82).

*Table 6-2* Greek Characters

| Symbol | Entity Name | Description |
|---|---|---|
| **Lowercase Greek Letters** | | |
| α | `α` | * Lowercase Greek Alpha |
| β | `β` | * Lowercase Greek Beta |
| χ | `χ` | * Lowercase Greek Chi |
| δ | `δ` | * Lowercase Greek Delta |
| ε | `ϵ` | * Alternate lowercase Greek Epsilon |
| ϕ | `φ` | * Lowercase Greek Phi |
| φ | `ϕ` | * Open lowercase Greek Phi |
| γ | `γ` | * Lowercase Greek Gamma |
| η | `η` | * Lowercase Greek Eta |
| ι | `ι` | * Lowercase Greek Iota |
| κ | `κ` | * Lowercase Greek Kappa |
| λ | `λ` | * Lowercase Greek Lambda |
| μ | `μ` | * Lowercase Greek Mu |
| ν | `ν` | * Lowercase Greek Nu |

*Table 6-2* Greek Characters *(Continued)*

| Symbol | Entity Name | Description |
|---|---|---|
| π | `π` | * Lowercase Greek Pi |
| ϖ | `ϖ` | * Alternate lowercase Greek Pi (or Omega) |
| θ | `θ` | * Lowercase Greek Theta |
| ϑ | `ϑ` | * Open lowercase Greek Theta |
| ρ | `ρ` | * Lowercase Greek Rho |
| σ | `σ` | * Lowercase Greek Sigma |
| ς | `&tsigma;` | * Lowercase Greek Sigma1 |
| τ | `τ` | * Lowercase Greek Tau |
| υ | `υ` | * Lowercase Greek Upsilon |
| ω | `ω` | * Lowercase Greek Omega |
| ξ | `ξ` | * Lowercase Greek Xi |
| ψ | `ψ` | * Lowercase Greek Psi |
| ζ | `ζ` | * Lowercase Greek Zeta |
| **Uppercase Greek Letters** | | |
| Δ | `&Udelta;` | * Uppercase Greek Delta |
| Φ | `&Uphi;` | * Uppercase Greek Phi |
| Γ | `&Ugamma;` | * Uppercase Greek Gamma |
| Λ | `&Ulambda;` | * Uppercase Greek Lambda |
| Π | `&Upi;` | * Uppercase Greek Pi |
| Θ | `&Utheta;` | * Uppercase Greek Theta |
| Σ | `&Usigma;` | * Uppercase Greek Sigma |
| ϒ | `&Uupsilon;` | * Uppercase Greek Upsilon |
| Ω | `&Uomega;` | * Uppercase Greek Omega |
| Ξ | `&Uxi;` | * Uppercase Greek Xi |
| Ψ | `ϒ` | * Uppercase Greek Psi |

* To use any of the entities whose description is marked with an * (asterisk), you must use the `helpchar.ent` file (see “Including Special Characters” on page 82).

*Table 6-3* Math Symbols

| Symbol | Entity Name | Description |
|---|---|---|
| **Basic Math Symbols** | | |
| − | `−` | Minus |
| ± | `±` | Plus over minus |
| ÷ | `÷` | Divide |
| × | `×` | Multiply |
| ≤ | `≤` | Less than or equal to |
| ≥ | `≥` | Greater than or equal to |
| ≠ | `&neq;` | Not equal to |
| **Advanced Math Symbols** | | |
| $^2$ | `&squared;` | * Squared |
| $^3$ | `&cubed;` | * Cubed |
| 1/4 | `&one-fourth;` | * One-fourth |
| 1/2 | `&one-half;` | * One-half |
| 3/4 | `&three-fourths;` | * Three-fourths |
| ∞ | `&infty;` | * Infinity |
| ≡ | `≡` | * Exactly equals |
| ≠ | `&not-eq;` | * Not equal to |
| ≈ | `≈` | * Approximate sign |
| ¬ | `&neg;` | * Not |
| ∩ | `∩` | * Cap (Set intersection) |
| ∪ | `∪` | * Cup (Set union) |
| ∨ | `∨` | * Vee (Logical OR) |
| Λ | `∧` | * Wedge (Logical AND) |
| ∈ | `∈` | * In |
| ⊂ | `⊂` | * Proper subset |
| ⊆ | `⊆` | * Subset |
| ⊃ | `⊃` | * Proper superset |

*Table 6-3* Math Symbols *(Continued)*

| Symbol | Entity Name | Description |
|---|---|---|
| ⊇ | `⊇` | * Superset |
| ∀ | `∀` | * For all (Universal symbol) |
| ∃ | `&exists;` | * There exists (Existential symbol) |
| ∉ | `&not-in;` | Not element |
| *f* | `&function;` | * Function symbol (or florin sign) |
| ∠ | `∠` | * Angle |
| ≅ | `≅` | * Congruent |
| ∝ | `∝` | * Proportional to |
| ⊥ | `⊥` | * Perpendicular to |
| · | `ċ` | * Centered dot |
| ⊕ | `⊕` | * Plus in circle |
| ⊗ | `⊗` | * Times in circle |
| ∅ | `ø` | * Slash in circle (Empty set) |
| ∂ | `&partial;` | * Partial differential delta |
| ∑ | `∑` | * Summation (Uppercase Greek Sigma) |
| × | `∏` | * Product (Uppercase Greek Pi) |

* To use any of the entities whose description is marked with an * (asterisk), you must use the `helpchar.ent` file (see “Including Special Characters” on page 82).

*Table 6-4* Arrow Symbols

| Symbol | Entity Name | Description |
|---|---|---|
| ← | `←` | * Left arrow |
| → | `→` | * Right arrow |
| ↑ | `↑` | * Up arrow |
| ↓ | `↓` | * Down arrow |
| ↔ | `↔` | * Left/right arrow |
| ⇐ | `&bigleftarrow;` | * Big left arrow |
| ⇒ | `&bigrightarrow;` | * Big right arrow |

*Table 6-4* Arrow Symbols *(Continued)*

| Symbol | Entity Name | Description |
|---|---|---|
| ⇑ | `&biguparrow;` | * Big up arrow |
| ⇓ | `&bigdownarrow;` | * Big down arrow |
| ⇔ | `&bigleftrightarrow;` | * Big left/right arrow |

* To use any of the entities whose description is marked with an * (asterisk), you must use the `helpchar.ent` file (see "Including Special Characters" on page 82).

*Table 6-5* Miscellaneous Symbols

| Symbol | Entity Name | Description |
|---|---|---|
| **Current Date and Time** | | |
| `8/18/95` | `&date;` | Today's date (when HelpTag is run) |
| `09:50` | `&time;` | Current time (when HelpTag is run) |
| **Currency Symbols** | | |
| ¢ | `&cents;` | Cents |
| £ | `&sterling;` | Sterling |
| ¥ | `¥` | Yen |
| **Units** | | |
| ° | `°` | Degrees |
| ′ | `&minutes;` | Minutes, prime, or feet |
| ″ | `&seconds;` | Seconds, double prime, or inches |
| `AM` | `&a.m.;` | AM |
| `PM` | `&p.m.;` | PM |
| **Card Suits** | | |
| ◊ | `♦` | * Diamond suit |
| ♥ | `♥` | * Heart suit |
| ♠ | `♠` | * Spade suit |
| ♣ | `♣` | * Club suit |
| **Other Symbols** | | |
| ◊ | `⋄` | * Diamond |

*Table 6-5* Miscellaneous Symbols *(Continued)*

| Symbol | Entity Name | Description |
|---|---|---|
| ¿ | `&invert-question;` | Inverted question mark |
| ¡ | `&invert exclamation;` | Inverted exclamation mark |
| ¤ | `&currency;` | Currency |
| ∴ | `∴` | Therefore |
| « | `&openanglequote;` | Open angle quotes |
| » | `&closeanglequote;` | Close angle quotes |
| ℵ | `ℵ` | * Hebrew Aleph |
| ∇ | `∇` | * Nabla (Inverted uppercase Greek Delta) |
| √ | `&surd;` | Radical segment, diagonal |
| ℘ | `℘` | * Weierstraussain symbol |
| ℜ | `ℜ` | * Fraktur R |
| ℑ | `&im;` | * Fraktur I |

* To use any of the entities whose description is marked with an * (asterisk), you must use the `helpchar.ent` file (see "Including Special Characters" on page 82).

# Command Summary 7

This chapter summarizes the command-line options available when the help commands are run manually in a terminal window.



## Help System Commands

Desktop actions and data types provided by the Help System enable you to compile and view run-time help files by clicking a help file icon or choosing a menu item. However, if you want to select particular command options, you must enter the command manually in a terminal window or create new actions.

Help actions and data types are defined in two files, `dthelp.dt` and `dthelptag.dt`, located in the `/usr/dt/appconfig/types/`*lang* directory.

The commands summarized here are:

`dthelptag` Compiles HelpTag source files into a run-time file.

`dthelpview` Displays a help volume, help topic, text file, or man page.

| | |
|---|---|
| `dthelpgen` | Collects help family files into a new help volume, `browser.hv`, which contains an entry for each family file. |

## Processing HelpTag Files (dthelptag)

The HelpTag software, invoked with the `dthelptag` command, compiles your HelpTag source files into a run-time help file. You run `dthelptag` in the directory where your *volume*`.htg` file is located.

### Command Syntax

`dthelptag [`*command-options*`]` *volume* `[`*parser-options*`]`

Where *command-options* are options entered before the *volume* name and *parser-options* are options entered after the *volume* name.

### Command Options

| | |
|---|---|
| `-clean` | Removes all files generated from any previous run of HelpTag for the given *volume*. |
| `-shortnames` | Causes the names of all generated files to be limited to a maximum of eight characters for the base name and three characters for the extension. This allows run-time help files to be moved to systems where longer names may not be supported. |
| `-verbose` | Displays the progress of the `dthelptag` command and displays any parser errors that occur. Parser errors are also saved in a file named *volume*`.err`. |
| `-formal` | Uses the formal parser to interpret help files tagged with SGML-compliant markup. If not specified, `dthelptag` assumes the input file contains shorthand markup. |

Because there are two types of markup—shorthand and formal—it is recommended to distinguish the types by using a file extension. Use `.htg` for shorthand markup and use `.ctg` for formal markup.

## Parser Options

Parser options, which are entered after the *volume* name, are passed directly to the *parser*, which is the part of the HelpTag software that converts your marked-up files into a run-time file.

These options can be applied in the following ways:

- Entered on the command line after the *volume* name.
- Listed in a file named `helptag.opt` located in the current directory.
- Listed in a file named *volume*`.opt` in the current directory.
- Set using the `DTTAGOPT` environment variable.

Options entered on the command line override those options that may have also been set using a different method.

| | |
|---|---|
| `onerror` | Specifies whether the `dthelptag` command should continue if a parser error is encountered. The default is `onerror=stop`, which causes the command to stop even if one parser error is encountered. If you specify `onerror=go`, processing will continue, but the created run-time help file may not work properly. |
| `charset` | Specifies which character set was used to author the text files. The correct character set name is needed to ensure that the help topics are displayed in the proper font. The default is charset=ISO-8859-1. You can also specify a character set within your help volume by declaring an entity named `LanguageElementDefaultCharset`. The `/usr/dt/dthelp/dthelptag/helplang.ent` file includes this entity declaration. See Chapter 14, “Native Language Support,” for a list of supported character sets. |
| `search` | Adds another directory to the list of directories that are searched to find referenced file entities. To specify multiple directories, use multiple `search=`*directory* options. If no search options are used, only the current directory is searched. |
| `clearsearch` | Ignores the list of search directories. This option is useful in the command line to override search options specified in the `helptag.opt` file. |

| | |
|---|---|
| `memo` | Causes author's memos (which are entered using the `<memo>` element) to be included. The default is `nomemo`, which causes HelpTag to ignore memos. |
| `nomemo` | Causes HelpTag to ignore author's memos (which are entered with the `<memo>` element). This is the default. |

#### See Also

- "Creating Run-Time Help Files" on page 91.
- "Creating an Installation Package" on page 246.
- "Viewing a Help Volume" on page 94.
- `dthelptag (1)` man page.

## Displaying Help Topics (dthelpview)

The `dthelpview` command can be used to display a help volume, individual help topic, text file, or man page.

### Command Syntax

The various ways to invoke Helpview are:

- `dthelpview -helpVolume` *volume* [ -locationId *id* ].
- `dthelpview -man`.
- `dthelpview -manPage` *man*.
- `dthelpview -file` *filename*.

Where:

`-helpVolume` *volume*

Specifies the name of the *volume*`.sdl` file you want to view. A path name is not required unless the volume is not in the current directory *and* the volume has not been registered.

`-locationId` *id* — Specifies an ID. `dthelpview` displays the topic that contains *id*. If you do not specify an ID, Helpview uses `_hometopic` by default.

`-man` — Displays a dialog that prompts for a man page to view, then displays the requested man page.

`-manPage` *man* Specifies that a particular man page be displayed.

`-file` *filename* Specifies that a particular text file be displayed.

The default *volume* and *id* can be set in `dthelpview's` app-defaults file, `/usr/dt/app-defaults/C/Dthelpview`.

### See Also

- "Registering Your Application and Its Help" on page 249.
- "Viewing a Help Volume" on page 94.
- `dthelpview (1)` man page.

## Generating a Browser Help Volume (dthelpgen)

The `dthelpgen` utility creates a special help volume that enables users to display help volumes registered on their system using the Front Panel Help Viewer. When a user initially clicks the Help Viewer control in the Front Panel, `dthelpgen` is run automatically. It locates help family files by searching the help search path directories (local or networked), and then creates a browser volume (`browser.hv`) in the user's *HomeDirectory*`/.dt/help/$DTUSERSESSION` directory. Once built, the volume is updated in response to any of these actions:

- Add, remove, or modify family files or help volumes.
- Change the `LANG` environment variable.
- Invoke the ReloadApps action.
- Run `dthelpgen` manually in a terminal window.

The browser volume is displayed by clicking the Help Viewer control in the Front Panel. Or, you can manually run `dthelpview` and supply the browser volume name as shown in this command line:

```
dthelpview -h browser.hv
```

### Command Syntax

`dthelpgen -dir [`*options*`]`

Where:

`-dir` Specifies the directory in which to place the browser volume and intermediate files. This is a required parameter.

## Options

`-generate` Specifies that a new browser help volume should be created even if the family files and help volumes on the system have not been modified.

`-file` *basename*
Specifies the name of the help volume and any intermediate files generated by `dthelpgen`. The default name is `browser.hv`.

`-lang` Specifies which language directories to search for help families and help volumes. If the -lang option is set, it takes precedence over the current value of the `LANG` environment variable.

---

**Note –** If you run `dthelpgen` while the browser volume is displayed in a help window, you should close the window, then reopen the browser volume.

---

### See Also

- "Registering Your Application and Its Help" on page 249.
- `dthelpgen(1)` man page.

# Reading the HelpTag Document Type Definition 8

This chapter explains how to read the HelpTag 1.3 Document Type Definition (DTD) and how to use it to create fully compliant Standard Generalized Markup Language (SGML) help files.



## Document Type Definition

A *Document Type Definition* (DTD) defines a set of elements to create a structured (or hierarchical) document. The DTD specifies the syntax for each element and governs how and where elements can be used in a document.

### Helptag 1.3 DTD

The Helptag 1.3 DTD tag set and its associated rules are referred to as formal markup. The DTD conforms to the Standard Generalized Markup Language (SGML) ISO specification 8879:1986. This means that you can use formal markup to create help files that are SGML compliant.

Appendix A contains the complete DTD specification. The DTD is also available in the Developer's Toolkit. It is located in the `/usr/dt/dthelp/dthelptag/dtd` directory and is named `helptag.dtd.`

#### See Also

- `dthelptagdtd(4)` man page.

## DTD Components

The DTD defines each of the HelpTag elements described in previous chapters in a technical notation. This section introduces some key terms and explains how to read the syntax of the element notations. It does not attempt to fully describe each section of the DTD.

### Element Declarations

The DTD defines each element in an element declaration. The declaration uses a precise notation to describe an element, its required components, and any elements it can or cannot contain. An element may also have characteristics defined in an attribute declaration, which is discussed in the section "Attribute List Declarations" on page 201.

The syntax of an element declaration is:

`<ELEMENT` *`element_type minimization (content model)`* `>`

Where:

*element_ type* — Specifies the element name, which is also used as the tag name. For example, the tag for the element type `head` is `<head>`.

*minimization* — A two-character entry that indicates whether a start or an end tag is required. The first character represents the start tag; the second character represents the end tag. A space separates the two characters. The letter o means that the tag is optional. A - (minus sign) indicates the tag is required. For example, an entry like this, - - , indicates that the element requires both start and end tags. The DTD for Helptag 1.3 requires start and end tags for every element.

*content model* Specifies a list of the required and optional elements that the element type can contain. It defines the sequence of elements and, if applicable, the number of occurrences that may occur.

The content model uses these notations:

| A vertical bar represents "or".

+ Element must appear at least once. It can be repeated.

* Element can appear zero or more times.

? Element can appear zero or one time.

, A comma describes sequence, that is, the element type must be followed by the element specified after the comma.

+ (*element_ type(s)*)
The + (plus sign) indicates that the listed element or elements can be used within the element type or within any of the elements it contains. It is called an inclusion. Parentheses are used to enclose one or more elements.

- (*element_ type(s)*)
A - (minus sign) indicates that the listed element or elements cannot be used within this element, or within any of the elements it contains. It is called an exclusion. Parentheses are used to enclose one or more elements.

## Examples

Each example contains a word description for the element declaration provided. Required start and end tags are assumed.

- A chapter requires a `<chaphead>` followed by text. A chapter can contain zero or more `s1` elements followed by zero or more `rsect` elements.

```
<!ELEMENT chapter - - (chaphead, text, (s1*, rsect*)) >
```

- A `chaphead` requires a head followed by an optional `abbrev`. A `chaphead` cannot contain these elements: `memo`, `location`, or `idx`.

  ```
  <!ELEMENT chaphead - - (head, abbrev?)
                         -(memo | location | idx) >
  ```

- The paragraph element requires a start tag (-) and an end tag (-). It can contain an optional head (?) followed by the `partext` element. `newline` elements can be used within `p` or any of the elements it contains.

  ```
  <!ELEMENT p - - (head?, partext) +(newline) >
  ```

- A `note` contains text. It can have an optional head. A `note` cannot contain these elements: `note`, `caution`, or `warning`.

  ```
  <!ELEMENT note - - (head?, text)
                     -(note | caution | warning) >
  ```

- A `list` may contain an optional head. It requires at least one item, which can be repeated.

  ```
  <!ELEMENT list - - (head?, item+) >
  ```

- The `book` element declaration uses an exclusion to specify that it cannot contain another `book` element.

  ```
  <!ELEMENT book - - (partext) -(book) >
  ```

## Element Declaration Keywords

Some elements include a keyword in the element declaration that describes the data content of the element. Three keywords appear in the DTD: `EMPTY`, `CDATA`, and `#PCDATA`.

| | |
|---|---|
| `EMPTY` | Specifies that the element has no data content that will be displayed in the online information. `newline` and `xref` elements are examples. |
| `CDATA` | Represents "character data"; that is, the data content of the element is not recognized as markup. |
| `#PCDATA` | Represents "parsed character data"; that is, the data content may include both text and markup characters that the Help System parser interprets accordingly. |

### Attribute List Declarations

An attribute list declares additional properties that further describe an element. An attribute list declaration has the syntax:

```
<!ATTLIST element_type attribute_values default_value>
```

For example, a `list` element has four attributes: `type`, `ordertype`, `spacing`, and `continue`. Values for each type are declared. The last column shows the default values. Because only one value exists for the `continue` attribute, a default value is omitted.

```
<!ATTLIST list type      ( order
                           bullet
                           plain
                           check )       bullet
               ordertype ( ualpha
                           lalpha
                           arabic
                           uroman
                           lroman )      arabic
                spacing   ( tight
                            loose )      tight
                continue  (continue)    #IMPLIED >
```

This markup creates a numbered list (uppercase alphabet) that supplies extra spacing between list items.

```
<list order ualpha loose>
   <item>
     <text>
       <p>
         <partext>Introducing the Front Panel></partext>
       </p>
     </text>
    </item>
```

## Formal Markup

Using a structured editor is the best approach for creating formal markup. After learning the basic set of elements, an author can get started. This is done by choosing elements from a menu. In response, the structured editor generates all of the tags required for each element. In addition, the application verifies that the structural framework being created conforms to the Document Type Definition.

See the section “Write Help Topics with HelpTag” on page 18 for a description of shorthand and formal markup, and structured editors.

## Formal Markup Caveats

Shorthand and formal markup share a common set of elements, such as chapter, section, head, list, paragraph, and so forth. However, formal markup differs from shorthand markup in these important ways:

- Every element requires a start and an end tag.
- Tags for each subcomponent of an element must be entered.
- The / (forward slash) is the end tag delimiter in formal markup. End tags use this format, </*tagname*>.
- Entity declarations use the SYSTEM parameter instead of the FILE parameter used in shorthand declarations.

### Explicit Start and End Tags

Each element, its component parts, and elements it contains must be explicitly tagged. For example, here is the formal markup for a chapter head. To read this, and other markup examples easily, tags are indented. Indentation is not required in actual markup.

```
<chaphead>
      <head>
             <partext>Front Panel Help</partext>
      </head>
</chaphead>
```

Notice the additional tags, `<head>` and `<partext>`; these are subcomponents of the `<chaphead>` element. Each of these elements requires an explicit start and end tag.

## Explicit Hierarchy of Elements

Each element declaration contributes to a set of rules that governs how and where elements can be used. Because elements contain other elements, which may contain other elements, a document is a hierarchy of elements. At the top level, <`helpvolume`> is a container for every other element.

To decide what markup is necessary to create a help topic, you need to become familiar with the rules. For example, suppose you want to create a chapter. First, look at the declaration for `chapter` listed below. It specifies that a `chaphead` is required. Next, look at the rules for `chaphead`. It, in turn, requires a `head`. Consequently, look at the declaration for `head`, and continue until you have reached the last nested element—in this case, `partext`. Until you are familiar with the elements you commonly use, this approach will help you enter markup correctly.

```
<!ELEMENT chapter - - (chaphead, text?, (s1*, rsect*)) >
<!ELEMENT chaphead - - (head, abbrev?)
                       -(memo | location | idx | footnote) >
<!ELEMENT head - - (partext)
                   -(memo | location | idx)>
<!ELEMENT partext - - ((#PCDATA . . . ))>
```

Using a structured editor minimizes what an author needs to know about the DTD. The editor application “reads” the DTD and creates each element’s required tags, many of which are intermediate structural tags.

## Example

This formal markup sample is an excerpt from the desktop Text Editor help volume. To view the corresponding online information, choose the Help Viewer in the Front Panel. Select TriTeal Enterprise Desktop and then choose Text Editor Help from the listed volumes. In the Text Editor volume, choose Text Editor Tasks and then To Open an Existing Document.

Indentation is used in this example to make it easier to read the text and corresponding element tags.

```
<s2 id=”TOOPENANEXISTINGDOCUMENT”>
<chaphead><head>
<partext>To Open an Existing Document</partext>
     </head></chaphead>
<text>
<p>
<partext>You can use Text Editor or File Manager to open an existing
document.
</partext></p>
<idx><indexprimary>
<partext>document</partext></indexprimary>
     <indexsub>
<partext>opening</partext></indexsub></idx>
```

```
<idx><indexprimary>
<partext>opening</partext></indexprimary>
     <indexsub>
<partext>existing document</partext></indexsub></idx>
<procedure>
<chaphead><head>
<partext>From Text Editor</partext>
     </head></chaphead>
<text>
<list type="ORDER">
<item><text><p>
<partext>Choose Open from the File menu.</partext></p>
<p>
<partext>The Open a File dialog box lists files and folders on your
system.You can browse the documents listed, or change to a new folder
to locate other files on your system.</partext>
     </p></text></item>
<item><text><p>
<partext>Select the document you want to open in the Files list or
type the file name in the Open a File field.</partext></p>
<p>
<partext><emph><partext>Or,</partext></emph> if the document is not
in the current folder, first change to the folder that contains your
document. Then choose a name in the Folders list or type the path
name of the folder you wish to change to in the Enter path or folder
name field.</partext></p></text></item>
<item><text><p>
<partext>Press Return or click OK.
</partext></p></text></item></list>
<figure tonumber="NONUMBER" entity="TEXTEDITOROPENFILE">
</figure></text></procedure>

<procedure><chaphead><head>
<partext>From File Manager</partext>
     </head></chaphead>
<idx><indexprimary>
<partext>opening</partext></indexprimary>
     <indexsub>
<partext>document from File Manager</partext></indexsub></idx>
<idx><indexprimary>
<partext>document</partext></indexprimary>
     <indexsub>
<partext>opening from File Manager</partext></indexsub></idx>
<idx><indexprimary>
<partext>File Manager</partext></indexprimary>
     <indexsub>
```

```
<partext>opening document</partext></indexsub></idx>
<text>
<list type="BULLET">
<item><text><p>
<partext>Display the document’s file icon in a File Manager
window.</partext>
     </p></text></item>
<item><text><p>
<partext>Do <emph><partext>one</partext></emph> of the
following:</partext></p>
<list type="BULLET">
<item><text><p>
<partext>Double-click the document’s file icon.</partext>
     </p></text></item>
<item><text><p>
<partext>Select the document, then choose Open from the Selected
menu.</partext>
     </p></text></item>
<item><text><p>
<partext>Drag the document to Text Editor’s control in the Front
Panel.</partext>
     </p></text></item></list></text>
     </item></list><text> </procedure>
<procedure><chaphead><head>
<partext>See Also</partext>
     </head></chaphead>
<text>
<list type="BULLET" spacing="TIGHT">
<item><text><p>
<partext><xref id="ENTERINGANDEDITINGTEXT"></partext>
     </p></text></item>
<item><text><p>
<partext><xref id="TOSAVEADOCUMENTTOTHECURRENTFILE"></partext>
     </p></text></item>
<item><text><p>
<partext><xref id="TABLEOFCONTENTS"></partext>
     </p></text></item></list></text>
     </procedure></text></s2>
```

## File Entity Declarations

To declare a file entity in formal markup, use this syntax:

<!entity *entityname* SYSTEM “*filename*”>

Where *entityname* is the name of the entity and *filename* is the name of the file. The keyword `SYSTEM` is required.

---

**Note –** To use entity declarations previously created for shorthand markup, you must replace the `FILE` parameter with `SYSTEM`.

---

### Example

Here are the entity declarations for a help volume that consists of three text files and contains a graphic image.

```
<!entity MetaInformation SYSTEM "metainfo>"
<!entity BasicTasks SYSTEM "basics">
<!entity AdvancedFeatures SYSTEM "advanced">
<!entity process_diagram SYSTEM "process.tif">
```

Entities are referenced in formal markup exactly as they are in shorthand markup.

## Processing Formal Markup

When you process formal markup using `dthelptag`, you must use the `-formal` command-line option. For example, to process a formal markup file named `Icons.ctg` in verbose mode, enter this command:

```
dthelptag -verbose -formal Icons.ctg
```

---

**Note –** The command option specifies the type of markup in the input file. The run-time file created by running `dthelptag` is always *volume*`.sdl`. The online format is identical whether you used shorthand or formal markup.

---

# Part 3 — The Programmer's Job

# Creating and Managing Help Dialog Boxes

9

This chapter describes the Help dialog widgets and how to create them.



## Help Dialog Boxes

For application programmers, the Help System provides a programming library that adds help dialog boxes to any OSF/Motif application. The library provides two types of help dialog boxes:

- *General help dialogs* have a menu bar, a topic tree, and a help topic display area. The topic tree lists the help volume's topics and subtopics. Users use the topic tree to select topics to view, to browse available topics, and to locate where they are in the help volume.
- *Quick help dialogs* contain a topic display area and one or more dialog buttons at the bottom.

### Standard Xt Paradigm

In terms of programming, you interact with the help dialogs the same as you do with any other OSF/Motif widgets in your applications. The two types of help dialog boxes are defined as two new widget classes: `DtHelpDialog` and `DtHelpQuickDialog`.

Nearly every attribute of the help windows—including the volume name and topic ID—are manipulated as widget resources. For instance, to display a new topic, you just execute an `XtSetValues()` call to set the `DtNhelpVolume`, `DtNlocationId`, and `DtNhelpType` resources. For more information, refer to "Displaying Help Topics" on page 218.

---

**Note –** Integrating the Help System into an application requires a working knowledge of the C programming language, the OSF/Motif programmer's toolkit, and the Xt Intrinsics toolkit.

---

## General Help Dialog

A general help dialog has two display areas: the topic tree and topic display area. The topic tree provides a scrollable list of help topics. The home topic title is always the first item. When a user chooses a title, an arrow (→) marks the title and its help information is displayed in the topic display area. Figure 9-1 on page 211 shows the topic tree and topic display area of a general help window. The current topic, "To select a palette", is displayed.

The general help dialog includes three dialog buttons: Backtrack, History, and Index. These commands are also available in the Help menus. For an overview of the Help dialogs and the graphical user interface, refer to the section, "Help User Interface" on page 7.

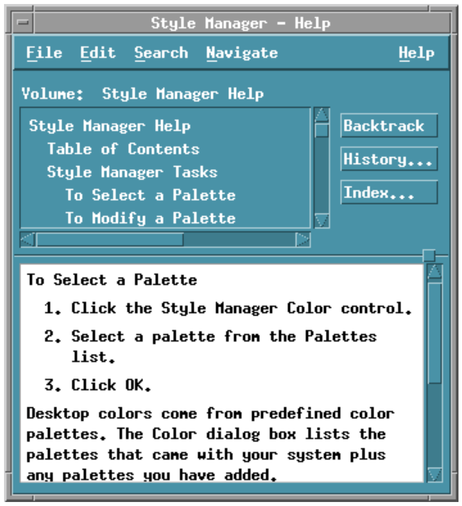



*Figure 9-1* General help dialog

## ▼ To Create a General Help Dialog

1. Include the appropriate header files:

```
#include <Help.h>
#include <HelpDialog.h>
```

2. Create an instance of the general help dialog widget:
   - Use the `DtCreateHelpDialog()` convenience function.

     *Or,* use the `XtCreateManagedWidget()` function.

3. Add a callback for handling hyperlink events that occur within the dialog. (For more information, see "Responding to Hyperlink Events" on page 232.)

4. Add a close callback for handling the Close command.

### Example

The following code segment creates a general help dialog (as a child of *parent*) using the convenience function. The dialog is left unmanaged—presumably it is managed elsewhere in the application when a help request is made.

```
Widget   mainHelpDialog, moreButton, helpButton;
ac = 0;
XtSetArg (al[ac], XmNtitle,  "My Application - Help");  ac++;
XtSetArg (al[ac], DtNhelpVolume,  "My Help Volume");  ac++;
XtSetArg (al[ac], DtNlocationId,  "Getting Started");  ac++;
XtSetArg (al[ac], DtNhelpType,  "DtHELP_TYPE_TOPIC");  ac++;

mainHelpDialog =
  DtCreateHelpDialog (parent,  "mainHelpDialog", al, ac);
```

The following two calls add the hyperlink and close callbacks to the dialog. Presumably, the functions `HyperlinkCB()` and `CloseHelpCB()` are declared elsewhere in the application.

```
XtAddCallback (mainHelpDialog, DtNhyperLinkCallback,
               HyperlinkCB, (XtPointer)NULL);
XtAddCallback (mainHelpDialog, DtNcloseCallback,
               CloseHelpCB, (XtPointer)NULL);
```

### See Also

- “Providing Help on Help” on page 237.
- “To Enable the Application-Configured Button” on page 235.
- `DtCreateHelpDialog(3)` man page.
- `DtHelpDialog(3)` man page.

## Quick Help Dialog

The quick help dialog box is designed to help you meet the first objective of online help: *Get the user back on task as quickly and successfully as possible.* This simple user interface helps keep the user focused on the information. The information should be useful enough that the user dismisses the dialog after reading it and continues working.

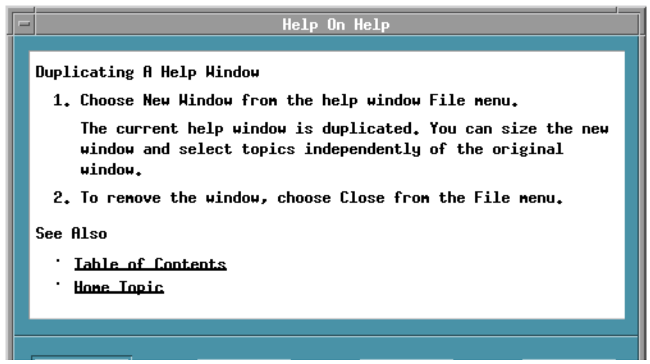



*Figure 9-2* Quick help dialog with four standard buttons

The quick help dialog has five buttons, four of which are managed. The remaining dialog button is configurable, so this button can be used for anything. However, its intended purpose is to provide a path to more help in one of these two ways:

- Let the user ask for more detailed information. In this case, the default button label (More) is appropriate. This is called *progressive disclosure*.
- Let the user open a general help dialog for general browsing of the application's help volume. In this case, Browse… is the most appropriate button label.

The Developer's toolkit includes a convenience function, `DtHelpQuickDialogGetChild()`, for determining the widget ID for any of the quick help dialog buttons.

### ▼ To Create a Quick Help Dialog

1. Include the appropriate header files:

```
#include <Help.h>
#include <HelpQuickD.h>
```

2. Create an instance of the quick help dialog widget:
   - Use the `DtCreateHelpQuickDialog()` convenience function.

     *Or,* use the `XtCreateManagedWidget()` function.
3. Add a callback for handling hyperlink events that occur within the dialog. (For more information, see "Responding to Hyperlink Events" on page 232.)
4. Add a close callback for handling the OK button.
5. Configure the dialog buttons that you want to use:
   - If you intend to use the application-configured button, manage it and add an activate callback.
   - If you want to disallow printing, unmanage the Print button.
   - Manage the Help button and add a help callback to the dialog to allow the user to get help on help.

### Example

The following code segment creates a quick help dialog (as a child of *parent*) using the convenience function. The dialog is left unmanaged; presumably, it is managed elsewhere in the application when a help request is made. In this example, the application-configured button is enabled and used to request more help.

```
Widget   quickHelpDialog, moreButton, helpButton;

ac = 0;
XtSetArg (al[ac], XmNtitle,  "My Application - Help");  ac++;
XtSetArg (al[ac], DtNhelpVolume,  "My Help Volume");  ac++;
XtSetArg (al[ac], DtNlocationId,  "Getting Started");  ac++;
XtSetArg (al[ac], DtNhelpType,  "DtHELP_TYPE_TOPIC"); ac++;


quickHelpDialog =
  DtCreateHelpQuickDialog (parent,  "quickHelpDialog", al, ac);
```

The following two calls add the hyperlink and close callbacks to the dialog. Presumably, the functions `HyperlinkCB()` and `CloseHelpCB()` are declared elsewhere in the application.

```
XtAddCallback (quickHelpDialog, DtNhyperLinkCallback,
               HyperlinkCB, (XtPointer)NULL);
XtAddCallback (quickHelpDialog, DtNcloseCallback,
               CloseHelpCB, (XtPointer)NULL);
```

Here, the application-configured button is managed and assigned an activate callback that invokes the application's `MoreHelpCB()` function.

```
moreButton = DtHelpQuickDialogGetChild (quickHelpDialog,
                                     DT_HELP_QUICK_MORE_BUTTON);
XtManageChild (moreButton);
XtAddCallback (moreButton, XmNactivateCallback,
               MoreHelpCB, (XtPointer)NULL);
```

To provide "help on help," the dialog's Help button is managed and a help callback is added to the dialog.

```
helpButton = DtHelpQuickDialogGetChild (quickHelpDialog,
                                     DT_HELP_QUICK_HELP_BUTTON);
XtManageChild (helpButton);
XtAddCallback (quickHelpDialog,DtNhelpCallback,
               HelpRequestCB, USING_HELP);
```

Like other OSF/Motif dialogs, when you add a help callback to a quick help dialog, it is used by both the F1 key and the Help button.

### See Also

- "To Enable the Application-Configured Button" on page 235.
- Chapter 12, "Providing Help on Help".
- `DtCreateHelpQuickDialog(3)` man page.
- `DtHelpQuickDialog(3)` man page.
- `DtHelpQuickDialogGetChild(3)` man page.

## Summary of Application Program Interface

Related man pages for the Help System are:

- Functions for creating and working with dialogs:

```
DtHelp(5)
DtHelpDialog(5)
DtHelpQuickD(5)
DtCreateHelpDialog()
DtCreateHelpQUickDialog()
DtHelpQuickDialogGetChild()
```

- Function for implementing item help mode:

```
DtHelpReturnSelectedWidgetId()
```

- Function for specifying the message catalog for the Help library:

```
DtHelpSetCatalogName()
```

- Applications and actions for creating and viewing a help volume:

```
dthelptag(1)
dthelpview(1)
dthelpgen(1)
dthelpaction(5)
dtmanaction(5)
```

- Document Type Definitions:

```
dthelptagdtd(4)
dtsdldtd(4)
```

# Responding to Help Requests 10

This chapter explains how to display different types of help information by setting Help Dialog widget resources.



## Requesting Help

When a user requests help while using your application, it's the application's responsibility to determine what help topic should be displayed.

### Context Sensitivity

Some help requests amount to an explicit request for specific information, such as help on "version" (which usually displays the copyright topic). Other help requests, however, may require some degree of context sensitivity. That is, some processing might be needed to determine the appropriate help topic based on the user's current context within the application.

For instance, your application might test the status of certain modes or settings to determine the appropriate help topic. Or, it might test the value of an input field and provide detailed help if the value is not valid, and general help if the value is valid.

### Entry Points

An entry point is a specific location within a help volume—usually the beginning of a topic—that can be directly accessed by requesting help within the application.

From the author's point of view, entry points are established by assigning IDs at the appropriate places within the help volume. From the programmer's point of view, entry points are created by enabling the user to request help and using the appropriate ID when a particular request is made.

There are four general ways for users to request help:

- Pressing the help key (which is F1 on most keyboards).
- Clicking the Help button in a dialog box.
- Choosing a command from the application's Help menu.
- Choosing On Item help.

## Displaying Help Topics

When a help request is made, the application determines what help topic to display. It then creates (if necessary) and manages a help dialog, and sets the appropriate resources to display a help topic.

Most requests display help topics that are part of the application's help volume. But, the Help System's help dialogs are also capable of displaying man pages, text files, and simple text strings.

The Help System's help dialogs are based exclusively on Xt Intrinsics and OSF/Motif programming, so you change the values within a help dialog just like any other widget: by setting resources.

The `DtNhelpType` resource determines what type of information is displayed. It can be set to any of these values:

- `DtHELP_TYPE_TOPIC` for displaying normal help topics that are part of a help volume. The volume is specified by setting the `DtNhelpVolume` resource; the topic is specified by setting the `DtNlocationId` resource.
- `DtHELP_TYPE_STRING` for displaying a string supplied by the application. Automatic word wrap is disabled, so line breaks are observed as specified in the string. The string is specified by setting the `DtNstringData` resource.
- `DtHELP_TYPE_DYNAMIC_STRING` for displaying a string supplied by the application, using word wrap to format the text. Line breaks within the string are used to separate paragraphs. The string is specified by setting the `DtNstringData` resource.
- `DtHELP_TYPE_FILE` for displaying a text file. The name of the file to be displayed is specified by setting the `DtNhelpFile` resource.
- `DtHELP_TYPE_MAN_PAGE` for displaying a manual reference page (man page) in a help dialog. The man page to be displayed is specified by setting the `DtNmanPage` resource.

These values are defined in the `Help.h` file.

### See Also

- "Creating and Managing Help Dialog Boxes" on page 209

## ▼ To Display a Help Topic

1. Create a help dialog.
2. Set the following resources for the help dialog:

| | |
|---|---|
| `DtNhelpType` | Set to `DtHELP_TYPE_TOPIC`. |
| `DtNhelpVolume` | Set to the volume name for your application. |
| `DtNlocationId` | Set to the topic ID that you want to display. |

   You can also set other values for the dialog, such as its size and title.

3. Manage the dialog using `XtManageChild()`.

### Example

This program segment displays a topic with the ID `getting-started` in the volume `MyVolume`.

```
ac = 0;
XtSetArg (al[ac], DtNhelpType,   DtHELP_TYPE_TOPIC);  ac++;
XtSetArg (al[ac], DtNhelpVolume,  "MyVolume");         ac++;
XtSetArg (al[ac], DtNlocationId,  "getting-started");  ac++;
XtSetArg (al[ac], DtNcolumns,    40);                  ac++;
XtSetArg (al[ac], DtNrows,       12);                  ac++;
XtSetValues (helpDialog, al, ac);
XtManageChild (helpDialog);
```

If the help volume `MyVolume` is not registered, then a complete path to the `MyVolume.sdl` file is required for the value of `DtNhelpVolume`.

## ▼ To Display a String of Text

1. Create a quick help dialog.

   You can use a general help dialog to display string data, but this isn't recommended because most of its features do not apply to string data.

2. Set the following resources for the help dialog:

   | | |
   |---|---|
   | `DtNhelpType` | Set to `DtHELP_TYPE_DYNAMIC_STRING` (if you want word wrap enabled) or `DtHELP_TYPE_STRING` (if you want the line breaks within the string to be maintained) . |
   | `DtNstringData` | Set to the string you want to display. A copy of the string is kept internally, so you need not maintain your copy of it. |

   You can also set other values for the dialog, such as its size and title.

3. Manage the dialog using `XtManageChild()`.

### Example

This program segment displays a string stored in the variable `descriptionString`.

```
ac = 0;
XtSetArg (al[ac], DtNhelpType,   DtHELP_TYPE_DYNAMIC_STRING); ac++;
XtSetArg (al[ac], DtNstringData, (char *)descriptionString);   ac++;
XtSetValues (quickHelpDialog, al, ac);
XtManageChild (quickHelpDialog);
```

If the string is no longer needed within the application, the memory can be freed, because the help dialog makes its own copy of the data.

```
XtFree (descriptionString);
```

## ▼ To Display a Text File

1. Create a quick help dialog or retrieve one from your dialog cache.

   You can use a general help dialog to display a text file, but this isn't recommended because most of its features are useful only for standard help topics.

2. Set the following resources for the help dialog:

   | | |
   |---|---|
   | `DtNhelpType` | Set to `DtHELP_TYPE_FILE`. |
   | `DtNhelpFile` | Set to the file name you want to display. If the file is not in the application's current directory, provide a path to the file. |

   You can also set other values for the dialog, such as its size and title. In particular, you might want to set the width to 80 columns, which is the standard width for text files.

3. Manage the dialog using `XtManageChild()`.

### Example

The following program segment displays a file named `/tmp/printer.list`. It also sets the size of the dialog to better suit a text file.

```
ac = 0;
XtSetArg (al[ac], DtNhelpType, DtHELP_TYPE_FILE);      ac++;
XtSetArg (al[ac], DtNhelpFile,  "/tmp/printer.list");  ac++;
XtSetArg (al[ac], DtNcolumns,  80);                    ac++;
XtSetArg (al[ac], DtNrows,     20);                    ac++;
XtSetValues (quickHelpDialog, al, ac);
XtManageChild (quickHelpDialog);
```

### ▼ To Display a Man Page

1. Create a quick help dialog.

   You can use a general help dialog to display a man page, but this isn't recommended because most of its features are useful only with standard help topics.

2. Set the following resources for the help dialog:

   | | |
   |---|---|
   | `DtNhelpType` | Set to `DtHELP_TYPE_MAN_PAGE`. |
   | `DtNmanPage` | Set to the name of the man page. The value of this resource is passed directly to the system `man` command. So, to specify a particular section of a man page, precede the man page name by a section number, just as you would if you were typing the `man` command conventionally. |

   You can also set other values for the dialog, such as its size and title.

3. Manage the dialog using `XtManageChild()`.

#### Example

The following program segment displays the man page for the `grep` command. It also sets the size of the dialog to better suit a man page.

```
ac = 0;
XtSetArg (al[ac], DtNhelpType, DtHELP_TYPE_MAN_PAGE);  ac++;
XtSetArg (al[ac], DtNmanPage,   "grep");                ac++;
XtSetArg (al[ac], DtNcolumns,  80);                     ac++;
XtSetArg (al[ac], DtNrows,     20);                     ac++;
XtSetValues (quickHelpDialog, al, ac);
XtManageChild (quickHelpDialog);
```

## Enabling the Help Key (F1)

The help key mechanism is a feature built into all OSF/Motif manager widgets and primitive widgets. The help key is enabled by adding a *help callback* to the widget where you want the help key active.

Within your application, you should add a help callback to each widget where you want a unique entry point into help. The help callback mechanism automatically "walks" up the widget hierarchy (up to the shell widget) until it finds a widget with a help callback, then invokes that callback.

If you add a help callback to a manager widget, when the help key is pressed for any of its children, the manager's help callback is invoked (unless the child widget has a help callback of its own).

## ▼ To Add a Help Callback

♦ Use the `XtAddCallback()` function as follows:

```
XtAddCallback (
      Widget          widget,
      String          DtNhelpCallback,
      XtCallbackProc  HelpRequestCB,
      XtPointer       clientData );
```

Where:

| | |
|---|---|
| *widget* | The widget where you want to activate the help key. |
| *HelpRequestCB*() | The function in your application that handles the help request when the user presses the help key. |
| *clientData* | The data you want passed to the *HelpRequestCB*() function. Typically, this data identifies the topic to be displayed. |

When the user presses the help key, the help callback is invoked for the widget with the current keyboard focus. If that widget does not have a help callback, the help callback for its nearest ancestor that does have a help callback is invoked.

If no help callbacks are found, nothing happens. Therefore, it is recommended that you add a help callback to each shell in your application. This ensures that no user requests for help are lost.

Adding a help callback to a dialog shell automatically enables the Help button on the dialog to invoke the help callback.

## Importance of Client Data

Specifying a unique value for *clientData* in each help callback you add saves you the trouble of writing a separate function to process each help callback. Your application can have a single callback procedure to process all help requests (see "To Add a Help Callback" on page 223). Within the callback procedure, use the *clientData* to identify where the user requested help. That is, each time you add a help callback, you should provide a unique value for *clientData*.

### Example

The following example demonstrates one way to associate IDs with entry points. A `HelpEntryIds.h` file is used to define a unique integer for each *clientData* value for each help callback. Also defined are two ID strings for each widget: one for normal F1 help, the other for item help mode (where the user picks a widget to get a description).

For this example, assume that the application's user interface is just a main window with three input fields: Name, Address, and Telephone Number. Here's what the `HelpEntryIds.h` file would contain:

```
#define HELP_volumeName             "MyVolume"

#define HELP_MainWindow             100
#define HELP_MainWindow_ID          "basic-tasks"
#define HELP_MainWindow_ITEM_ID     "main-window-desc"

#define HELP_NameField              101
#define HELP_NameField_ID           "specifying-a-name"
#define HELP_NameField_ITEM_ID      "name-field-desc"

#define HELP_AddressField           102
#define HELP_AddressField_ID        "specifying-an-address"
#define HELP_AddressField_ITEM_ID   "address-field-desc"

#define HELP_PhoneField             103
#define HELP_PhoneField_ID          "specifying-a-phone-no"
#define HELP_PhoneField_ITEM_ID     "phone-field-desc"
```

Within the part of the application that initially creates the widgets, a help callback is added to each widget as follows:

```
XtAddCallback (mainWindow, DtNhelpCallback,
               HelpRequestCB, HELP_MainWindow);
XtAddCallback (nameField, DtNhelpCallback,
               HelpRequestCB, HELP_NameField);
XtAddCallback (addressField, DtNhelpCallback,
               HelpRequestCB, HELP_AddressField);
XtAddCallback (phoneField, DtNhelpCallback,
               HelpRequestCB, HELP_PhoneField);
```

Within the `HelpRequestCB()` function, the *clientData* parameter is used to dispatch the help requests (through a `switch()` statement). Within each case, the value of a global flag `itemHelp` is tested to see if the help callback was invoked by the F1 key (the flag is "false") or by the user picking the widget in item help mode (the flag is "true").

```
XtCallbackProc HelpRequestCB (
      Widget     w,
      XtPointer  clientData,
      XtPointer  callData )
{
   char    *topicToDisplay;
   Boolean  useQuickHelpDialog;
   /* Determine the topic ID for the given `clientData.' */
   switch ((int)clientData)
     {
       case HELP_MainWindow:
         useQuickHelpDialog = False;
         if (itemHelpFlag)
           topicToDisplay = HELP_MainWindow_ITEM_ID;
         else
           topicToDisplay = HELP_MainWindow_ID;
         break;       case HELP_NameField:
         useQuickHelpDialog = True;
         if (itemHelpFlag)
           topicToDisplay = HELP_NameField_ITEM_ID;
         else
           topicToDisplay = HELP_NameField_ID;
         break;       case HELP_AddressField:
         useQuickHelpDialog = True;
         if (itemHelpFlag)
           topicToDisplay = HELP_AddressField_ITEM_ID;
         else
           topicToDisplay = HELP_AddressField_ID;
```

```
        break;       case HELP_PhoneField:
        useQuickHelpDialog = True;
        if (itemHelpFlag)
          topicToDisplay = HELP_PhoneField_ITEM_ID;
        else
          topicToDisplay = HELP_PhoneField_ID;
        break;       default:
        /* An unknown clientData was received. */
        /* Put your error handling code here. */
        return;
        break;
    }
  /* Display the topic. */
  ac = 0;
  XtSetArg (al[ac], DtNhelpType,   DtHELP_TYPE_TOPIC); ac++;
  XtSetArg (al[ac], DtNhelpVolume, HELP_volumeName);    ac++;
  XtSetArg (al[ac], DtNhelpType,   topicToDisplay);     ac++;
  if (useQuickHelpDialog)
    {
      XtSetValues (mainQuickHelpDialog, al, ac);
      XtManageChild (mainQuickHelpDialog);
    }
  else
    {
      XtSetValues (mainHelpDialog, al, ac);
      XtManageChild (mainHelpDialog);
    }
  /* Clear the `item help' flag. */
  itemHelpFlag = False;
}
```

The preceding function assumes that the application uses two help dialogs for all help requests (`mainHelpDialog` and `mainQuickHelpDialog`), and that those dialogs have already been created. It also assumes that `al` and `ac` (used in assembling Xt argument lists) are declared elsewhere.

## Providing a Help Menu

The *Style Guide and Certification Checklist* recommends that each menu bar include a Help menu. The Help menu may contain a variety of commands that let the user access different types of online help for your application.

The most important commands include:

- *Introduction* displays the home topic of your application's help, allowing the user to use hyperlinks to navigate to any desired information.
- *Using Help* displays help on help. This is information that tells the user how to use the Help System.
- *Version* displays your application's version and copyright information. The copyright topic (created using the `<copyright>` element) has the ID `_copyright`.

Additional commands may display help on special keyboard usage, application tasks, reference, or tutorials. You should design your Help menu to best suit your application, while staying within the guidelines and recommendations of the *Style Guide and Certification Checklist*.

### See Also

- "To Create a Home Topic" on page 41 describes how authors create the home topic for a help volume.
- "To Create a Meta Information Section" on page 43 describes how authors create a copyright topic.
- Chapter 12, "Providing Help on Help" describes how help on help is found and how to add it to your application.

## Supporting Item Help Mode

Some applications provide an On Item or Help Mode command in their Help menu. This command temporarily redefines the mouse pointer as a `?` (question mark) to prompt the user to select an item on the screen. When an item is selected, the application is expected to display a description of the item.

The convenience function, `DtHelpReturnSelectedWidgetId()`, changes the pointer to a question mark and waits for the user to pick a widget. The ID of the selected widget is returned. This function is similar to the `XmTrackingLocate()` function except that `DtHelpReturnSelectedWidgetId()` returns NULL if the user presses Escape to cancel the operation.

To display help on the selected item, your application can simply invoke the help callback for the returned widget. This is equivalent to the user pressing F1 while using that widget.

If you want the application to differentiate between item help and F1 help, you can set a flag before calling the widget's help callback. The help callback procedure can then use that flag to determine that the callback was invoked as a result of item help and alter its response accordingly.

## ▼ To Add Support for Item Help

1. Write a function that uses the `DtHelpReturnSelectedWidgetId()` function. Within this function, invoke the help callback for the selected widget. In the following steps, this function is called `ProcessOnItemHelp()`, but you can name it whatever you want.

2. Add to your Help menu a command button labeled On Item. Add an activate callback that invokes your `ProcessOnItemHelp()` function.

3. Add a help callback to each widget in your application where you want item help to be available.

If the selected widget does not have a help callback, the application should try its parent widget. Similarly, if the parent does not have a help callback, the application should continue to walk up the widget hierarchy until it finds a help callback.

### Example

The following procedure is a sample `ProcessOnItemHelp()` function that would be invoked by choosing On Item from the Help menu.

```
void  ProcessOnItemHelp(
   Widget widget)
{
 /*  Declare a variable for the selected widget.  */
  Widget selWidget=NULL;
  int status=DtHELP_SELECT_ERROR;
  /*  Get an application shell widget from our widget hierarchy to
    *  pass into DtHelpReturnSelectedWidgetId().
    */
  while (!XtIsSubclass(widget, applicationShellWidgetClass))
                   widget = XtParent(widget);
```

```
  status = DtHelpReturnSelectedWidgetId(widget, NULL, &selWidget);
  switch ((int)status)
    {
      case DtHELP_SELECT_ERROR:
        printf("Selection Error, cannot continue\n");
      break;
      case DtHELP_SELECT_VALID:
          /* We have a valid widget selection, now let's look for a registered help
           * callback to invoke.
           */
          while (selWidget != NULL)
            {
              if ((XtHasCallbacks(selWidget, XmNhelpCallback)
                                        == XtCallbackHasSome))
                {
                  /* Found a help callback, so just call it */
                  XtCallCallbacks((Widget)selWidget,
                                  XmNhelpCallback,NULL);
                   break;
                }
              else
                  /* No help callback on current widget, so try the widget's parent */
                  selWidget = XtParent(selWidget);
            }
      break;
      case DtHELP_SELECT_ABORT:
        printf("Selection Aborted by user.\n");
      break;
      case DtHELP_SELECT_INVALID:
        printf("You must select a component within your app.\n");
      break;
    }
}
```

# Handling Events in Help Dialogs 11

This chapter describes several Help dialog events that an application must be equipped to handle.



## Supporting Help Dialog Events

Like other widgets within your application, help windows have some behavior that must be supported by the application.

### Hyperlink Events

Most standard hyperlink events are handled internally by the Help System. However, there are four types of hyperlinks that your application is responsible for handling:

- *Jump-new-view hyperlinks*—Your application must create a new help dialog to honor the author's request for a topic to be displayed in a new help window.

- *Man page links*—Your application must create a new quick help dialog (or get one from your cache) to display a man page. Typically, the size of man page windows is different from all other help windows.
- *Application-defined links*—Your application must interpret the data associated with these links. Application-defined links exist only if you and the author have collaborated to create them.
- *Text file links*—Your application must create a quick help dialog (or get one from you cache) to display the text file.

### When Dialogs Are Dismissed

When the user closes a help dialog, your application needs to know so it can store the dialog in its cache, or destroy it. The general help dialog supports a help closed callback. To detect when a quick dialog is dismissed, add a callback to its Close button.

### Quick Help Buttons

The behavior for some of the buttons in quick help dialogs must be handled by your application. These buttons can be managed and unmanaged as needed. You add behavior just like any other push button: using an activate callback.

#### See Also

- “Creating Hyperlinks” on page 61 describes the types of links supported by the Help System and explains how to create them.

## Responding to Hyperlink Events

Your application needs to provide support only for the types of hyperlinks used within the help volume to be displayed. In general, it is recommended that you provide support for all link types.

For your application to be notified when a hyperlink is chosen, it must add a *hyperlink callback* to the help dialog. You must write a callback function that handles the hyperlink appropriately.

## ▼ To Provide a Hyperlink Callback

1. Add a hyperlink callback to each help dialog as shown:

   ```
   XtAddCallback (helpDialog, DtNhyperlLinkCallback,
                  HyperlinkCB, (XtPointer)NULL);
   ```

   Where *helpDialog* is the widget ID of the help dialog and *HyperlinkCB* is the name of the callback function for handling hyperlinks.

2. Write the *HyperlinkCB* function to handle the hyperlink events that can occur within the dialog.

Within the hyperlink callback, you have access to the following callback structure (which is declared in `<Dt/Help.h>`):

```
typedef struct
  {
      int      reason;
      XEvent  *event;
      char    *locationId;
      char    *helpVolume;
      char    *specification;
      int      hyperType;
      int      windowHint;
  } DtHelpDialogCallbackStruct;
```

The `hyperType` element indicates which type of link was executed. Its possible values are: `DtHELP_LINK_TOPIC`, `DtHELP_LINK_MAN_PAGE`, `DtHELP_LINK_APP_DEFINE`, or `DtHELP_LINK_TEXT_FILE`. For a description of which structure elements are valid for different types refer to the `DtHelpDialog(3)` man page.

The `windowHint` element indicates a window type. Its possible values are: `DtHELP_CURRENT_WINDOW`, `DtHELP_POPUP_WINDOW`, or `DtHELP_NEW_WINDOW.`

### Example

The following function, `HyperlinkCB()`, illustrates the general structure needed to handle hyperlink callbacks.

```
XtCallbackProc
HyperlinkCB (widget, clientData, callData)
     Widget     widget;
     XtPointer  clientData;
```

```
    XtPointer  callData;
{
    DtHelpDialogCallbackStruct *hyperData =
        (DtHelpDialogCallbackStruct *) callData;
    switch ((int)hyperData-> hyperType)
        {
            case DtHELP_LINK_TOPIC:
              /* Handles "jump new view"hyperlinks. */
              break;
            case DtHELP_LINK_MAN_PAGE:
              /* Handles "man page" hyperlinks. */
              break;
            case DtHELP_LINK_APP_DEFINE:
              /* Handles "application-defined" hyperlinks. */
              break;
            case DtHELP_LINK_TEXT_FILE:
              /* Handles "text file" hyperlinks. */
              break;
            default:
              break;
        }
```

## Detecting When Help Dialogs Are Dismissed

To detect when a general help dialog is closed, add the following callback to the dialog:

```
XtAddCallback (helpDialog, DtNcloseCallback,
               HelpCloseCB, (XtPointer)NULL);
```

Where *helpDialog* is the widget ID for the help dialog and *HelpCloseCB* is the name of the callback procedure you've written to handle closing dialogs.

To detect when a quick help dialog is closed, add the following callback to the dialog's OK button:

```
XtAddCallback (DtHelpQuickDialogGetChild (helpDialog,
DtHELP_QUICK_OK_BUTTON), XmNactivateCallback, HelpCloseCB,
(XtPointer)NULL);
```

Where *helpDialog* is the widget ID for the help dialog and *HelpCloseCB* is the name of the callback procedure you've written to handle closing dialogs.

## Using the Application-Configured Button

The quick help dialog's application-configured button lets you add custom behavior to any quick help dialog. This button can be used for anything you want, but its intended purpose is to provide a path to more help in one of these two ways:

- Lets the user progressively ask for more information. This is sometimes called progressive disclosure. In this case, the default button label (More) is appropriate.
- Lets the user open a general help dialog for general browsing of the application's help volume. In this case, Browse… is the most appropriate button label.

### ▼ To Enable the Application-Configured Button

1. Get the button's ID.
2. Add an activate callback to the button.
3. Manage the button.

#### Example

The following code segment gets the button's ID, assigns a callback, and manages the button. It assumes that `quickHelpDialog` was just created.

```
Widget  moreButton;
moreButton = DtHelpQuickDialogGetChild (quickHelpDialog,
                                          DtHELP_QUICK_MORE_BUTTON);
XtAddCallback (moreButton, XmNactivateCallback,
                MoreHelpCB, NULL);
XtManageChild (moreButton);
```

#### See Also

- "To Create a Quick Help Dialog" on page 213.
- `DtHelpDialog(3)` man page.
- `DtHelpQuickDialog(3)` man page.

# Providing Help on Help 12

This chapter explains how to incorporate a help volume into your application that describes the features of the Help System and how to use them. This help volume provides help on using the Help dialog boxes.



## Providing Help on Help

Help on help tells users how to use the Help System. Specifically, it describes such tasks as using hyperlinks, navigating topics, using the index, and printing help topics. Normally, help on help is supplied as an individual help volume named Help4Help.

The Help4Help volume and its source files are included in the Developer's Toolkit. You can use the default volume "as is," or modify it for your application's design.

### For Application Help

If you are writing application-specific help, there are two ways to ensure that your application has help on help for its own help dialogs:

- *Rely on the desktop's help on help volume.* For example, on workstations running the desktop, the standard Help4Help volume is installed.
- *Supply your own help on help volume.* The HelpTag source files for the Help4Help volume are provided in the `/usr/dt/dthelp/help4help/C` directory. A `control` subdirectory contains HelpTag processing files. You run HelpTag in this directory to create the run-time help file. Graphics files used in the help on help volume are stored in the `control/graphics` subdirectory.

### For Standalone Help

If you are writing standalone help, you are probably relying on the Helpview program already being installed and ready to use. If this is the case, you don't have to worry about help on help because Helpview accesses the standard Help4Help volume by default.

### How Help on Help Is Found

Each application that uses the Help System (including Helpview) has a `helpOnHelpVolume` resource that identifies a help volume to be accessed for help on help topics. For Helpview, this resource is set as follows:

```
DtHelpview*helpOnHelpVolume:  Help4Help
```

If you provide your own help on help volume, be sure to give it a unique name so it doesn't conflict with another help on help volume that may be installed on the system.

## Accessing Help on Help in an Application

Your application should do the following to support help on help:

- Set the `helpOnHelpVolume` resource to identify the help volume you want to access.
- Add a Using Help command to the application's Help menu.

## ▼ To Set the helpOnHelpVolume Resource

♦ Add a line to your application's application-defaults file like this:

*App-class*`*helpOnHelpVolume:` *volume*

Where *App-class* is the application's class name and *volume* is the name of the help on help volume you want to access.

*Or,* within your application, set the `helpOnHelpVolume` resource for each help dialog you create.

### Examples

- This line in `dthelpview's` application-defaults file (`DtHelpview`) specifies the help on help volume:

```
DtHelpview*helpOnHelpVolume:  Help4Help
```

- To specify the help on help volume when creating a help dialog, add it to the argument list passed to the create function as shown here:

```
ac = 0;
XtSetArg (al[ac], XmNtitle, "My Application - Help"); ac++;
XtSetArg (al[ac], DtNhelpOnHelpVolume, "Help4Help");  ac++;
helpDialog = DtCreateHelpDialog (parent, "helpDialog", al, ac);
```

## ▼ To Provide a Using Help Command

1. Add to your Help menu a button labeled Using Help. Also add the necessary activate callback to call your `HelpRequestCB()` function.

2. Add support to your `HelpRequestCB()` function to display help on help. Specifically:
   - Create a quick help dialog.
   - Set the dialog's title to Help On Help.
   - Display the home topic of the help on help volume.
   - Manage the quick help dialog.

### Example

The following lines create a menu button labeled Using Help… that calls the `HelpRequestCB()` function.

```
/* Create the `Using Help ...' button. */
labelStr = XmStringCreateLtoR ("Using Help ...",
XmSTRING_DEFAULT_CHARSET);
ac = 0;
XtSetArg (al[ac], XmNlabelString, labelStr);     ac++;
button = XmCreatePushButtonGadget (parent, "usingHelpButton", al,
ac);
   XtManageChild (button);
   XmStringFree (labelStr);
   /* Add a callback to the button. */
   XtAddCallback (button,XmNactivateCallback,HelpRequestCB,
   USING_HELP);
```

`USING_HELP` is the client data passed to the `HelpRequestCB()` function when the menu button is chosen by the user. Presumably it has been defined somewhere in the application (perhaps in a `Help.h` file) as a unique integer:

```
#define USING_HELP  47
```

To see how the `HelpRequestCB()` function handles the `USING_HELP` case, see the example in the next section, "To Display Help on Help."

## ▼ To Display Help on Help

1. Create a quick help dialog (or retrieve one from your cache).
2. Display in the dialog the home topic of your help on help volume.

Help on help can be displayed in a general help window. However, a quick help dialog is recommended because its user interface is simpler, which is less intimidating to new users who commonly need help on help.

### Example

The following program segment is part of a `HelpRequestCB()` function. Presumably, the `USING_HELP` constant is passed to the function because the user chose Using Help from the application's Help menu or chose the Help button in a quick help dialog.

This example assumes that the application never creates more than one Help On Help dialog and maintains its widget ID in a variable called `onHelpDialog`.

```
case USING_HELP:
   if (onHelpDialog == (Widget)NULL)
     {
        /* Get a quick help dialog for use as the `help on help' dialog. */
        onHelpDialog = FetchHelpDialog (True);

        if (onHelpDialog == (Widget)NULL)
        /* We didn't get a dialog! Add your error handling code here. */
     }

   /* Set the proper volume and ID to display the home topic of
      the help on help volume. Also, set the dialog's title.     */
   ac = 0;   XtSetArg (al[ac], XmNtitle,  "Help On Help");     ac++;
   XtSetArg (al[ac], XmNhelpType,   DT_HELP_TYPE_TOPIC); ac++;
   XtSetArg (al[ac], XmNhelpVolume, "Help4Help");        ac++;
   XtSetArg (al[ac], XmNlocationId, "_hometopic");       ac++;
   XtSetValues (onHelpDialog, al, ac);

   /*  If the `help on help' dialog is already managed, it might
       be in another workspace, so unmanage it.   */
   if (XtIsManaged (onHelpDialog))
     XtUnmanageChild (onHelpDialog);

   /* Manage the `help on help' dialog. */
   XtManageChild (onHelpDialog);

   break;
```

To see how the rest of the `HelpRequestCB()` function might be structured, refer to the example in "To Add a Help Callback" on page 223.

### See Also

- "To Create a Quick Help Dialog" on page 213.
- "To Display a Help Topic" on page 219

## Writing Your Own Help on Help Volume

If you need to provide your own help on help volume, you should start with the existing Help4Help volume and then make the necessary changes. All the source files used to write the Help4Help volume are provided in the `/usr/dt/dthelp/help4help/C` directory.

To prevent installation conflicts, name your help on help volume something other than Help4Help. Consider picking a name that is specific to your product. For example, if your application's help volume is Newapp, your help for help volume could be NewappH4H.

## Required Entry Points

To ensure that context-sensitive help within a help dialog operates correctly, you must provide the following entry points (IDs) within your help on help volume. (These are already included in the Help4Help source files.)

| ID | Topic Description |
|---|---|
| `_hometopic` | Displays an introduction to using the help system. This topic is displayed when you choose Using Help from the general help dialog's Help menu, or when you press F1 in a quick help dialog. (The `ID _hometopic` is created automatically by the `<hometopic>` element.) |
| `_copyright` | Displays the copyright and version information for the help on help volume. This topic is displayed when you choose Version from the general help dialog's Help menu. (The `ID _copyright` is created automatically by the `<copyright>` element.) |
| `history` | Displays a topic that describes how to use the History dialog. This topic is displayed when you choose Help or press F1 within the History dialog. |
| `printing` | Displays a topic describing how to use the Print dialog. This topic is displayed when you choose Help or press F1 within the Print dialog. |
| `index-search` | Displays a topic describing how to use the Index Search dialog. This topic is displayed when you choose Help or press F1 within the Index Search dialog. |
| `volume-select` | Displays a topic describing how to use the Search Volume Selection Dialog. This topic is displayed when you choose Help or press F1 within the Search Volume Selection Dialog. |

## To Copy the Help4Help Source Files

1. Copy the entire `/usr/dt/dthelp/help4help/C` directory to a new working directory (*new-dir*) using a command like this:

   ```
   cp -r /usr/dt/dthelp/help4help/C new-dir
   ```

   This creates *new-dir* and copies all the files and directories into it.

2. To permit editing the files (which are copied as read only), change the permissions using a command like this:

   ```
   chmod -R u+w new-dir
   ```

The Help4Help volume uses these HelpTag source files:

- MetaInfo
- Toc
- Tasks
- HomeTopic
- Concepts
- Reference
- Glossary

Also included is a `control` directory, where you run HelpTag to create the run-time help file. Graphics are stored in the `control/graphics` subdirectory.

Be sure to rename the `Help4Help.htg` file before running HelpTag. Your help on help volume should have a unique name to prevent conflicts with other help on help volumes.

### Example

The following commands create a copy of the help on help volume and make its files writable. (Presumably the `projects` subdirectory already exists.)

```
cp -r /usr/dt/dthelp/help4help/C /users/dex/projects/NewHelp4Help
chmod -R u+w /users/dex/projects/NewHelp4Help
```

To build a new version of the run-time help files, first ensure that the directory `/usr/dt/bin` is in your search path. Then, change to the new directory, rename the `Help4Help.htg` file, and run HelpTag:

```
cd /users/dex/projects/NewHelp4Help
mv Help4Help.htg NewH4H.htg
dthelptag NewH4H
```

When the HelpTag software is done, you can display the new help on help volume using this command:

```
dthelpview -helpVolume NewH4H
```

# Preparing an Installation Package 13

This chapter identifies the help files that are included in an application installation package. It also describes how help files are handled when your application is registered on the desktop.



## Overview

When it comes time to prepare your final product, you must be sure that all your help files are created and installed properly. Your product package includes both the run-time help file (*volume*.sdl) and its graphic files. Additionally, you can provide a help family file that enables your volume to be viewed using the Front Panel Help Viewer.

## Delivering Online Help

Online help can be fully integrated into an application or provided as a standalone help volume. Fully integrated help allows a user to directly access help information from an application by using a Help menu or Help key. A standalone volume on the other hand, can only be displayed using the desktop Help Viewer.

A system administrator may choose to add a standalone help volume to the desktop when an application does not provide integrated help or a customized environment provides a supplemental help volume. See "Standalone Help" on page 249 for instructions to install a standalone volume on the desktop.

## Creating an Installation Package

Your installation package should include these help files:

- Run-time help files.
- Graphics files.
- Help family file (optional).
- Application defaults file (optional).

The run-time help file and any graphics used in the online help are included in your installation package. A help family file is optional for integrated application help. However, if you want your application help to be browsable using the desktop Help Viewer, you must provide a family file. If you are delivering a standalone help volume, you must provide a help family file. See "To Create a Help Family" on page 97.

If your application's help volume includes execution links, it is recommended that the author define execution aliases in an application defaults file. This takes advantage of the Help System's default execution policy which will automatically execute links with execution aliases. However, if the help volume is viewed as an independent volume using a separate information viewer, such as the Help Viewer, the Help System will display a confirmation dialog box when an execution link is selected.

Figure 13-1 on page 247 shows a typical installation package for an application and its help files. Help files are grouped in a separate `help` subdirectory which contains a default language directory (`C` is the default). The run-time help file, family file, and graphics files are located in this directory.

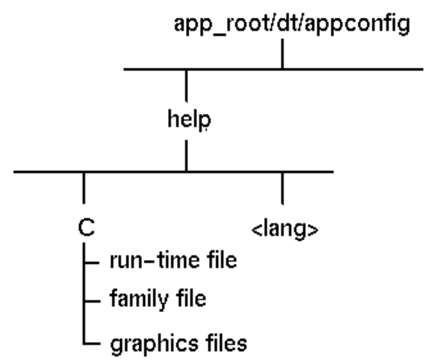



*Figure 13-1* Application installation package

If your application provides online help in multiple languages, you should create a *language* subdirectory to accommodate each language (where *language* matches the user's `LANG` environment variable). For example, an application that provides both an English and German user interface stores its corresponding online help in two subdirectories: `C` for English and `german` for German.

## Run-Time Help File

HelpTag creates a single run-time help file, *volume*`.sdl`. The base name, *volume*, is the same as the base name of your *volume*`.htg` file. The Help Viewer uses information stored in this master help file and also accesses any associated graphic files.

You don't need to ship the *volume*`.htg` or any additional files generated by the HelpTag software.

## Graphics Files

If your help volume uses graphics, the image files are typically stored in a separate directory for convenience. However, you may choose to store them in the same location as your *volume*`.htg` file.

A run-time help file does not include actual graphic images. Instead, it contains a "reference" to the location of each graphic file. When you run HelpTag, the `dthelptag` compiler incorporates the relative path names of the graphics files into the help volume.

When the help files are installed, the graphics files must be in the same relative position as when the run-time file was built. Otherwise, the help volume will be unable to locate the graphics files. For example, if your graphics files are in a subdirectory named `graphics` one level below your *volume*`.htg` file, then your installation package must preserve that relative position. The graphics files must be placed in a subdirectory named `graphics` one level below the *volume*`.sdl` file.

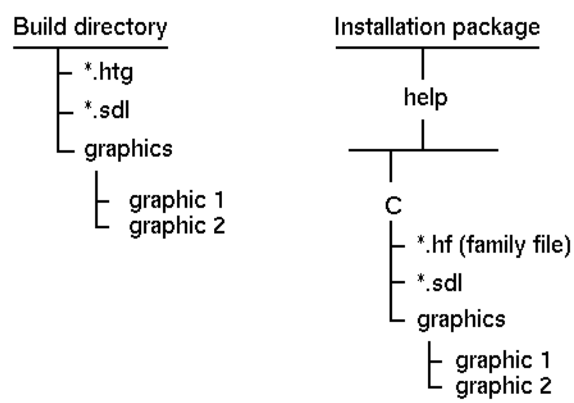



*Figure 13-2* Relationship of build directories and installation package

## Help Family File

You can optionally provide a help family file (*volume*`.hf`). A family file briefly describes your help volume and includes copyright information. It can also be used to group one or more related volumes into a single product category.

If you want your help volume to be accessible from the desktop browser volume, then you must provide a family file in your installation package. To create a family file, see “To Create a Help Family” on page 97.

## Registering Your Application and Its Help

The desktop's integration utility, `dtappintegrate`, registers your application and its help files by creating symbolic links between the installed application files and specific desktop directories. Application registration ensures that your help files are located in the directory search paths used by the Help System.

Registration enables two important features of the Help System:

- *Cross-volume hyperlinks* — A hyperlink in one help volume can refer to another help volume using just the volume name and an ID within the volume. If the destination volume is registered, the link does not have to specify where the volume is stored on the file system.
- *Help family browsing* — If you also register a "help family", then your help volumes will be browsable using the Help Viewer.

Registering your online help makes it easier to access the help you provide. For authors and programmers, it's easier because references to your volume can use just the volume name — without specifying the volume's actual location.

If you register a help family with one or more help volumes, you make your help available for general browsing from the Front Panel Help Viewer. This allows access to application-specific help without using the application. Or, if you are writing standalone help, this is the only way for users to get to your help.

### Standalone Help

A standalone help volume for an application or a customized environment can be created using the Help System Developer's Kit. To make the help volume accessible from in the desktop browser volume, a system administrator installs the run-time help file, associated graphics, and family file in the `/etc/dt/appconfig/help/`*language* directory.

Remember that the run-time help file and its graphics files must be installed in the same relative position as when the help volume was built. See "Graphics Files" on page 247 to review the installation of graphics files.

### What Happens When the Application Is Registered

Application registration creates symbolic links from the run-time help file and family located in *app_root*/dt/appconfig/help/*language* to the /etc/dt/appconfig/help/*language* directory.

Refer to the *Advanced User's and System Administrator's Guide* for detailed instructions for application registration.

### How a Help Volume Is Found

The Help System uses desktop search paths to locate help volumes. When help is requested within an application or a help volume is specified in a command line, the help volume is found by checking a set of search path directories. You can control the directory search path for help volumes by modifying several environment variables. Refer to the *Advanced User's and System Administrator's Guide* for detailed information about specifying search paths.

## Product Preparation Checklists

The following checklists should help you verify that you've prepared your product correctly. Of course, there's no substitute for testing your product by using it as a user will.

### For Authors

1. *A final version of the run-time help file was created.*

   Here are the recommended commands for creating the run-time file:

   ```
   dthelptag -clean volume
   dthelptag volume nomemo onerror=stop
   ```

   The `-clean` option removes files from any previous `dthelptag` command, the `nomemo` option ensures that writer's memos are not displayed, and the `onerror=stop` option stops processing if any parser errors occur. You should not distribute a help volume that has any parser errors.

2. *All hyperlinks have been tested.*

   Each hyperlink displays the proper topic or performs the correct action.

3. *Execution aliases have been defined for execution links.*

   Execution aliases are defined as resources in the application's application defaults file. An execution alias associates a name with a shell command to be executed. If you have used execution links in your help volume, coordinate with the application developer to add these resources to the application defaults file. For more information, refer to "Execution Aliases" on page 74.

4. *All graphics are acceptable.*

   The graphics have been tested on various color, grayscale, and monochrome displays.

## For Product Integrators

1. *The run-time file is installed.*

2. *All graphics are installed in the proper locations.*

   Each graphics file must be installed in the same relative position to the `.sdl` file that it was in relative to the `.htg` file when the HelpTag software was run.

3. *The help volume is registered.*

   The `dtappintegrate` script was run to create symbolic links from the installation directory to the registration directory.

4. *A product family file is installed and registered.*

   The family file is installed with the other help files. When `dtappintegrate` is run, it creates a symbolic link for the family file. Registering a family file for your help volume is optional. However, if you choose not to register a family file, your help volume will not be accessible from the Front Panel Help Viewer.

## For Programmers

1. *The application sets the correct values for these required resources:*

   *App-class*`*helpVolume:` *volume*
   *App-class*`*helpOnHelpVolume:` *help-on-help-volume*

The `helpVolume` resource identifies the help volume for your application.The `helpOnHelpVolume` identifies the help volume that contains the help on using the help system.

2. *Execution aliases are included in the application defaults file.*

   An author defines execution aliases as application resources. An execution alias associates a name with a shell command to be executed. If execution links have been used in the help volume, check with the author to identify the resources that need to be added. For more information, refer to "Execution Aliases" on page 74.

3. *The application sets the desired values for the following optional resources:*

   ```
   App-class*DtHelpDialogWidget*onHelpDialog*rows:      rows
   App-class*DtHelpDialogWidget*onHelpDialog*columns:   columns
   App-class*DtHelpDialogWidget*definitionBox*rows:     rows
   App-class*DtHelpDialogWidget*definitionBox*columns:  columns
   ```

   The `onHelpDialog` resources control the size of the quick help dialogs used to display Help on Help. The `definitionBox` resources control the size of the quick help dialog used for definition links.

4. *The application uses either the default font resources or defines font resources in the application's application-defaults file.*

   In most cases an application can rely on the default font resources. However, when custom fonts are used, they must be defined in the application-defaults file. Sample font schemes are provided in the `/usr/dt/dthelp/fontschemes` directory. See Chapter 14, "Native Language Support," for additional information about font schemes.

# Part 4 — Internationalization

# Native Language Support 14

This chapter identifies files used by the Help System that require modification when a help volume is provided in multiple languages.



## Internationalized Online Help

If your product is intended for an international audience, then providing online help in the user's native language is important. The Help System supports the authoring and displaying of online help in virtually any language.

When you process a help volume to create run-time help files, the HelpTag software must be told what language and character set you used to author your files. The language and character set information is also used to determine the proper fonts for displaying the help volume.

## Internationalization Factors

Several factors, which are explained in the following section, contribute to providing online help in the user's native language.

### Character Sets and Multibyte Characters

A *character set* determines how a computer's internal character codes (numbers) are mapped to recognizable characters. In most languages, single-byte characters are sufficient for representing an entire character set. However, there are some languages that use thousands of characters. These languages require two, three, or four bytes to represent each character uniquely.

Character sets supported by the Help System are listed in Table 14-1. However, some characters sets may not exist on all platforms.

*Table 14-1* TriTeal Enterprise Desktop Character Sets

| Language | Character Set Name | Description |
|---|---|---|
| Western Europe and Americas | `ISO-8859-1`<br>`HP-ROMAN8`<br>`IBM-850` | ISO Latin 1<br>HP Roman<br>PC Multi-lingual |
| Central Europe | `ISO-8859-2` | ISO Latin 2 |
| Cyrillic | `ISO-8859-5` | ISO Latin/Cyrillic |
| Arabic | `ISO-8859-6`<br>`HP-ARABIC8`<br>`IBM-1046` | ISO Latin/Arabic<br>HP Arabic8<br>PC Arabic |
| Hebrew | `ISO-8859-8`<br>`HP-HEBREW8`<br>`IBM-856` | ISO Latin/Hebrew<br>HP Hebrew8<br>PC Hebrew |
| Greek | `ISO-8859-7`<br>`HP GREEK8` | ISO Latin/Greek<br>HP Greek8 |
| Turkish | `ISO-8859-9`<br>`HP-TURKISH8` | ISO Latin 5<br>HP Turkish8 |
| Japanese | `EUC-JP` | Japanese EUC (JISX0201, JISX0208, JISX0212) |

*Table 14-1* TriTeal Enterprise Desktop Character Sets *(Continued)*

| Language | Character Set Name | Description |
|---|---|---|
| | HP-SJIS | HP Japanese Shift JIS |
| | HP-KANA8 | HP Japanese Katakana8 (JISX0201 1976) |
| | IBM-932 | PC Japanese Shift JIS |
| Korean | EUC-KR | Korean EUC |
| Chinese | EUC-CN | Simplified Chinese EUC (China) (GB2312) |
| | EUC-TW | Traditional Chinese EUC (Taiwan) (CNS 11643.*) |
| | HP-BIG5 | HP Traditional Chinese Big5 |
| | HP-CCDC | HP Traditional Chinese CCDC |
| | HP-15CN | HP Traditional Chinese EUC |
| Thai | TIS-620 | Thai |

When writing HelpTag files, you may use multibyte characters for any help text. However, the HelpTag markup itself (tag names, entity names, IDs, and so on) must be entered using eight-bit characters

## Language and Territory Names

When choosing a language, you select both a character set and a language and territory name. The language and territory name is used to accommodate variations, such as currency and date format, for a given country or region.

The language and territory names supported by the Help System are listed in the following table. Before you choose a language, refer to your system documentation to identify the languages and character sets supported on your platform.

*Table 14-2* Help System Language and Territory Names

| Languages | Language/Territory Name | Language, Territory |
|---|---|---|
| Standards compliance | | |
| | C | C |
| | POSIX | C |
| Western Europe/Americas | | |
| | da_DK | Danish, Denmark |
| | de_AT | German, Austria |
| | de_CH | German, Switzerland |
| | de_DE | German, Germany |
| | en_AU | English, Australia |
| | en_CA | English, Canada |
| | en_DK | English, Denmark |
| | en_GB | English, U.K. |
| | en_IE | English, Ireland |
| | en_MY | English, Malaysia |

*Table 14-2* Help System Language and Territory Names *(Continued)*

| Languages | Language/Territory Name | Language, Territory |
|---|---|---|
| | en_NZ | English, New Zealand |
| | en_US | English, USA |
| | es_AR | Spanish, Argentina |
| | es_BO | Spanish, Bolivia |
| | es_CL | Spanish, Chile |
| | es_CO | Spanish, Columbia |
| | es_CR | Spanish, Costa Rica |
| | es_EC | Spanish, Ecuador |
| | es_ES | Spanish, Spain |
| | es_GT | Spanish, Guatemala |
| | es_MX | Spanish, Mexico |
| | es_PE | Spanish, Peru |
| | es_UR | Spanish, Uruguay |
| | es_VE | Spanish, Venezuela |
| | et_EE | Estonian, Estonia |
| | fi_FI | Finnish, Finland |
| | fo_FO | Faroese, Faeroe Island |
| | fr_BE | French, Belgium |
| | fr_CA | French, Canada |
| | fr_CH | French, Switzerland |
| | fr_FR | French, France |
| | is_IS | Icelandic, Iceland |
| | it_CH | Italian, Switzerland |
| | it_IT | Italian, Italy |
| | kl_GL | Greenlandic, Greenland |
| | lt_LT | Lithuanian, Lithuania |
| | lv_LV | Latvian, Latvia |
| | nl_BE | Dutch, Belgium |
| | nl_NL | Dutch, The Netherlands |
| | no_NO | Norwegian, Norway |
| | pt_BR | Portuguese, Brazil |
| | pt_PT | Portuguese, Portugal |
| | sv_FI | Swedish, Finland |
| | sv_SE | Swedish, Sweden |
| Central Europe | | |
| | cs_CS | Czech |
| | hr_HR | Croatian, Croatia |
| | hu_HU | Hungarian, Hungary |
| | pl_PL | Polish, Poland |
| | ro_RO | Rumanian, Romania |
| | sh_YU | Serbocroatian, Yugoslavia |
| | si_CS | Slovenian |
| | si_SI | Slovenian |
| | sk_SK | Slovak |
| Cyrillic | | |
| | bg_BG | Bulgarian, Bulgaria |
| | mk_MK | Macedonian |
| | ru_RU | Russian |
| | ru_SU | Russian |
| | sp_YU | Serbian, Yugoslavia |
| Arabic* | | |
| | ar_SA | Arabic |

*Table 14-2* Help System Language and Territory Names *(Continued)*

| Languages | Language/Territory Name | Language, Territory |
|---|---|---|
| | `ar_AA` | Arabic |
| | `ar_DZ` | Arabic |
| Hebrew | | |
| | `iw_IL` | Hebrew, Israel |
| Greek | | |
| | `el_GR` | Greek, Greece |
| Turkish | | |
| | `tr_TR` | Turkish, Turkey |
| Asia | | |
| | `ja_JP` | Japanese, Japan |
| | `ko_KR` | Korean, Korea |
| | `zh_CN` | Chinese, China |
| | `zh_TW` | Chinese, Taiwan |
| Thai | | |
| | `th_TH` | Thai, Thailand |

* *No ISO territory name exists for the Arabic-speaking regions of the world. Vendors have supplied their own, which have been adopted for use in the TriTeal Enterprise Desktop.*

## Locale and Character Set

A help volume's default language and character set can be defined as an entity in the `helplang.ent` file. To specify a complete locale name, combine the language and territory name with the character set name using this syntax:

*language-and-territory-name*`.`*character-set-name*

For a description of the `helplang.ent` file, see "helplang.ent File" on page 261.

### Examples

- The following entity declaration specifies a complete locale name for the `C` standard language and the `ISO-8859-1` character set:

  ```
  <!ENTITY LanguageElementDefaultLocale   SDATA "C.ISO-8859-1">
  ```

- The same information could also be entered using two entity declarations as follows:

  ```
  <!ENTITY LanguageElementDefaultLocale       SDATA "C">
  <!ENTITY LanguageElementDefaultCharset      SDATA "ISO-8859-1">
  ```

- To specify the German language using the same character set, use this declaration:

```
<!ENTITY LanguageElementDefaultLocale   SDATA "de_DE.ISO-8859-1">
```

- Or, to specify the Japanese language using the EUC-JP character set, use this declaration:

```
<!ENTITY LanguageElementDefaultLocale   SDATA "ja_JP.EUC-JP">
```

If the locale is not specified in the helplang.ent file, then the value is derived from the value of the LANG environment variable.

## HelpTag Software

When you process a help volume to create run-time help files, the HelpTag software must be told what language and character set you used to author your files. The language and character set information is used to determine the proper fonts for displaying help topics. If you do not specify a language and character set, HelpTag assumes the default, which is English and ISO-8859-1.

The language and character set can be defined in the helplang.ent file (see page 261). Or, the character set can be specified as an option on the command line when running dthelptag in a terminal window.

---

**Note –** When writing HelpTag files, you may use multibyte characters for any help text. However, the HelpTag markup itself (tag names, entity names, IDs, and so on) must be entered using eight-bit characters.

---

## DtHelp Message Catalog

The menus, buttons, and labels that appear in help dialogs should also be displayed in the user's native language. To enable this, Help dialogs read such strings from a *message catalog* named DtHelp.cat.

The message catalog source file, DtHelp.msg, contains strings for menus, buttons, and messages. If the language you need is not supplied, you must translate the sample message catalog (/usr/dt/dthelp/nls/C/DtHelp.msg) and then use the gencat command to create the run-time message catalog file. See "To Create a Message Catalog" on page 267 for instructions.

Refer to your system documentation to determine the correct directory where your new message catalog should be installed.

## LANG Environment Variable

The user's LANG environment variable is important for two reasons:

- The value of LANG is used to locate the correct help volume.
- When a help topic is displayed, the correct fonts and formatting rules are chosen based on the user's LANG variable. This is especially important for Asian languages that have word-wrap rules that are more sophisticated than European and American languages.

### See Also

- *Internationalization Programmer's Guide*.
- NLS documentation for your computer's operating system or programmer's kit.

## helplang.ent File

The helplang.ent file defines text entities used by the Helptag software to determine the default locale and character set for a help volume. See "Locale and Character Set" on page 259 to learn how to specify a language and character set for your help volume.

The helplang.ent file also defines text entities for default strings such as Note, Caution, and Warning. If you want to override the English strings built into the HelpTag software, copy the file and localize the strings. The file is located in the directory /usr/dt/dthelp/dthelptag.

Here is an excerpt from the helplang.ent file:

```
<!ENTITY LanguageElementDefaultLocale         SDATA "C.ISO-8859-1">
<!ENTITY NoteElementDefaultHeadingString        SDATA "NOTE">
<!ENTITY CautionElementDefaultHeadingString     SDATA "CAUTION">
<!ENTITY WarningElementDefaultHeadingString     SDATA "WARNING">
<!ENTITY ChapterElementDefaultHeadingString     SDATA "Chapter">
<!ENTITY FigureElementDefaultHeadingString      SDATA "Figure">
<!ENTITY GlossaryElementDefaultHeadingString    SDATA "Glossary">
.
.
.
```

### Formatting Tables

A multibyte language, such as Japanese or Chinese, requires a *formatting table*. This table specifies a list of characters that cannot start a line and those characters that cannot end a line. When help files are processed, the formatting table ensures that lines wrap correctly. "Creating a Formatting Table" on page 265 explains how to create a new table or edit the sample table provided in the Help Developer's Kit.

### Font Schemes

One of the primary functions of the HelpTag software is to convert your marked-up files into a run-time format that the Help System understands. Text is formatted by specifying particular attributes such as type family, size, slant, and weight. A *font scheme* is simply a name, like an alias, that the Help System uses to assign fonts to HelpTag elements such as heads, procedures, lists, and so forth. It provides a way to map a group of text attributes used by the Help System with specific fonts.

Applications that use the standard TriTeal Enterprise Desktop fonts do not need to define additional font resources. If your application relies on a different set of fonts, you must create and add a font scheme to your application.

#### See Also

- `DtStdInterfaceFontNames (5)` man page.
- `DtStdAppFontNames (5)` man page.

## Understanding Font Schemes

When you write a help volume using the HelpTag markup language, you don't specify the fonts and sizes of the text. When you run the HelpTag software, the elements that you've entered are formatted into run-time help files that include text attributes.

A font scheme maps text attributes to actual font specifications. For example, if a help topic is formatted using a bold, sans serif typeface, the font scheme identifies which TriTeal Enterprise Desktop standard font or X font is actually used to display the text.

One of the primary uses of font schemes is to provide a choice of font sizes. The HelpTag software formats the body of most topics as 10-point text. However, because the actual display font is determined by the font scheme being used, all 10-point text could be specified to use a 14-point font.

## Font Resources

Each font scheme is actually a set of X resources. These resources are read by the application displaying the help. If you want to change the font scheme, you can set font resources in your application's application defaults file.

Each resource within a font scheme has this general form:

**pitch.size.slant.weight.style.lang.char-set*: *font specification*

Where:

*pitch* — Specifies the horizontal spacing of characters. This field should be either `p` (proportional) or `m` (monospace).

*size* — Specifies the height of the desired font. For help files formatted with HelpTag, this value should be `6`, `8`, `10`, `12`, or `14`.

*slant* — Specifies the slant of the desired font. Usually this field is either `roman` for upright letters or `italic` for slanted letters

*weight* — Specifies the weight of the desired font. Usually this field is either `medium` or `bold`.

*style* — Specifies the general style of the desired font. For help files formatted with HelpTag, this value should be either `serif` or `sans_serif`.

*lang* — Specifies that volumes compiled using this language should use these fonts. Usually the entry uses an * (asterisk) so that all volumes using the specified *char_set* will use these fonts.

*char-set* — Specifies the character set used to author the help text. This value must match the character set that was specified when HelpTag was run. The default is `ISO-8859-1`. Some special characters are displayed using a `symbol` character set.

An * (asterisk) can be used in a field to specify a font that has any value of that particular attribute. For instance, the `symbol` set (for special characters and special symbols) distinguishes a unique font based only on size and character set.

Its font resources appear like this within a font scheme:

```
*.6.*.*.*.*.DT-SYMBOL-1:  -adobe-symbol-medium-r-normal-*-*-60-*-*-
p-*-adobe-fontspecific
*.8.*.*.*.*.DT-SYMBOL-1:  -adobe-symbol-medium-r-normal-*-*-80-*-*-
p-*-adobe-fontspecific
*.10.*.*.*.*.DT-SYMBOL-1: -adobe-symbol-medium-r-normal-*-*-100-*-
*-p-*-adobe-fontspecific
*.12.*.*.*.*.DT-SYMBOL-1: -adobe-symbol-medium-r-normal-*-*-120-*-
*-p-*-adobe-fontspecific
*.14.*.*.*.*.DT-SYMBOL-1: -adobe-symbol-medium-r-normal-*-*-140-*-
*-p-*-adobe-fontspecific
```

The *char-set* field is the only field that cannot use the `*` (asterisk).

To display multibyte languages, such as Japanese or Korean, font resources must be specified using a font set. A font set is actually a group of fonts. A resource entry for a font set is similar to a single font, except a `,` (comma) separates multiple font names and the specification ends with a `:` (colon). Here is an example of a fully specified font resource for a Japanese font set.

```
bridge-gothic-medium-r-normal--18-180-75-75-c-80-jisx0201.1976-0,
bridge-gothic-medium-r-normal--18-180-75-75-c-160-jisx0208.1983-0,
bridge-gothic-medium-r-normal--18-180-75-75-c-160-jisx0212.1990-0:
```

You can also specify fonts for a multibyte language by providing a minimal XLFD font specification and allowing the system to supply the character set value to produce a font set.

```
*.12.roman.medium.*.ja_JP.EUC-JP: -*-*-*-*-*-*-*-120-*-*-*-*-*-*:
```

When specifying a font set, remember to end the specification with a `:` (colon). This instructs the Help System to load a set of fonts to display the information. Font sets are used to display multibyte languages. For volumes containing single-byte information, use the standard font specification.

### Sample Font Schemes

The `/usr/dt/dthelp/fontschemes` directory contains four font schemes:

| | |
|---|---|
| `fontDef.fns` | Default fonts used by the Help System |
| `fontLarge.fns` | Example of a larger font |
| `fontMulti.fns` | Example of a multi-byte font |
| `fontX11.fns` | Example of standard X11 fonts |

## To Choose a Font Scheme

♦ Edit the application-defaults file for the application that displays the online help. Replace the current font resources (if any) with the new scheme.

If you are making this change just for yourself, copy the application-defaults file into your home directory before editing it.

### Example

To use a larger size font (in the help dialogs) of a personal application named `DtStopWatch`, perform these steps:

Change to your home directory:

```
cd
```

Then copy the `DtStopWatch` application-defaults file and make it writable:

```
cp /usr/dt/app-defaults/C/DtStopWatch .
chmod u+w DtStopWatch
```

Edit the `DtStopWatch` file to add the largest scheme (`fontLarge.fns).` Go to the end of the file, and insert the contents of this file:

```
/usr/dt/dthelp/fontschemes/fontLarge.fns
```

Save your new `DtStopWatch` file.

Start the `DtStopWatch` application, select Help, and verify that help topics are displayed using the new font scheme.

# Creating a Formatting Table

A multibyte language, such as Japanese or Chinese, requires a *formatting table*. This table contains three message sets. The first set consists of characters that cannot start a line; the second set lists any characters that cannot end a line; and the third set indicates how to handle newline characters that occur between a single-byte and a multibyte character.

A formatting table is an ASCII file whose file name must end with a `.msg` extension. Figure 14-1 shows an excerpt from a formatting table for Simplified Chinese.

hpcvxbs3:/users/julia

```
$set 1
$
$ This file is only necessary for languages with multibyte character
$ sets.  For single byte character sets (I.E.  English, German, French,
$ etc.), the system has a built in default list of characters that can
$ not begin and end a line.  For single byte languages, the system will
$ also always replace newlines with spaces.
$
$ This table is for S-Chinese (GB code).
$
$ message #1 indicates the list of 2byte punctuation, special characters
$ and double consonants that cannot start a line.
$
$
1 、。¨…’ ”〕〉》」』〗】°′″！＇）,：；？]｝
$
$ message #2 indicates the list of 2byte punctuation, special characters
$ and double consonants that cannot end a line.
$
2  ‘“〔〈《「『〖【（［｀｛
$
$ message #3 indicates whether the language wants all end-of-lines in
$ text to be changed into spaces.
$
$ Therefore, the values for message #3 should be
$        1   - means that newlines are always turned into spaces.
$        0   - means that newlines are turned into space only if they
$              occur between a multibyte character and a single byte
$              character.
$
3 0
>
```

*Figure 14-1* Sample formatting table

Any line that begins with a `$` (dollar sign) followed by a space is a comment.

## Sample Formatting Table

A  sample formatting table for a multibyte character set is located in the `/usr/dt/dthelp/nls/zh_CN.hp15CN` directory and is named `fmt_tbl.msg`.

The sample table can be modified by adding or removing characters. To edit the formatting table, use an editor capable of composing characters in the language you have chosen for the help information. If you intend to create help information using a multibyte language, you need to create a formatting table.

### ▼ To Create a Message Catalog

If you translate the `DtHelp.msg` file, create a new formatting table, or modify the sample table (`fmt_tbl.msg`), you must update the message catalog used by the Help System.

♦ Use this command syntax to generate the catalog file:

`gencat` *file*`.cat` *file*`.msg`

Message catalogs for the standard desktop applications are located in the `/usr/dt/lib/nls/msg/`*lang* directory. To install a message catalog, refer to your operating system documentation for guidelines.

#### See Also

- `gencat (1)` man page.

## Displaying a Localized Help Volume

To view a help volume created for a locale different from your current system, you must set your `LANG` environment variable to match the help volume. The value of the `LANG` environment variable is platform-specific. If you are not familiar with this variable, check with your system administrator for the correct value and procedure to set your environment.

## Preparing Online Help for International Audiences

The following checklist summarizes the questions you should answer when providing online help for international audiences.

- Are help topics written with an international audience in mind?
- Did you copy the `/usr/dt/dthelp/dthelptag/helplang.ent` file and localize the string entities it contains? Using the entities in this file, you can override the English strings built into the HelpTag software.

- Was the HelpTag software run using the correct character set and language option? If you author in another character set, you may have to translate the `DtHelp.msg` message catalog file and provide a font scheme that supports the new character set.
- Within your HelpTag markup, are all tag names, entity names, and IDs entered using an eight-bit character set, even if the help text uses multibyte characters?
- When the user's LANG environment variable is set to the correct language, are the help files installed so they are found and displayed appropriately?
- If you have integrated the Help System into an application, have you properly set the locale using the `XtSetLanguageProc()` function?

### See Also

- "How a Help Volume Is Found" on page 250.
- `XtSetLanguageProc(3)` man page.
- `gencat (1)` man page.
- NLS documentation for your computer's operating system or programmer's kit.

# HelpTag 1.3 DTD

A

The HelpTag Document Type Definition (DTD) defines each HelpTag element and the syntax for its use. If you are not familiar with DTDs, refer to Chapter 8, “Reading the HelpTag Document Type Definition,” for a description of the specification.

The HelpTag 1.3 DTD is also available in the Developer’s Toolkit. It is located in the `/usr/dt/dthelp/dthelptag/dtd` directory and named `helptag.dtd`.

```
HelpTag 1.3 DTD
<!SGML “ISO 8879:1986”
--  SGML Declaration--
CHARSET
BASESET “ISO 646-1983//CHARSET International Reference Version
          (IRV)//ESC 2/5 4/0”
DESCSET      0        9      UNUSED
             9        2      9
            11        2      UNUSED
            13        1      13
            14       18      UNUSED
            32       95      32
           127        1      UNUSED
BASESET “ISO Registration Number 100//CHARSET ECMA-94
          Right Part of Latin Alphabet  Nr. 1//ESC  2/13  4/1”
DESCSET    128       32      UNUSED
           160        5      32
           165        1      UNUSED
           166       88      38
           254        1      127
           255        1      UNUSED
CAPACITY SGMLREF
    TOTALCAP 350000
    ENTCAP   100000
    ENTCHCAP  50000
    ELEMCAP   50000
    GRPCAP   210000
    EXGRPCAP  50000
    EXNMCAP   50000
    ATTCAP    50000
    ATTCHCAP  50000
    AVGRPCAP  50000
    NOTCAP    50000
    NOTCHCAP  50000
    IDCAP     50000
    IDREFCAP  50000
    MAPCAP   210000
    LKSETCAP  50000
    LKNMCAP   50000
SCOPE DOCUMENT
SYNTAX -- The Core Reference Syntax except with ATTCNT,LITLEN,
          NAMELEN,GRPCNT, and GRPGTCNT changed --
```

```
SHUNCHAR CONTROLS   0   1   2   3   4   5   6   7   8   9
                   10  11  12  13  14  15  16  17  18  19
                   20  21  22  23  24  25  26  27  28  29
                   30  31 127 255

BASESET "ISO 646-1983//CHARSET International Reference Version
          (IRV)//ESC 2/5 4/0"
DESCSET       0      128      0
FUNCTION      RE       13
              RS       10
              SPACE    32
              TAB      SEPCHAR    9
NAMING
   LCNMSTRT ""
   UCNMSTRT ""
   LCNMCHAR "-."
   UCNMCHAR "-."
   NAMECASE
       GENERAL YES
       ENTITY  YES
DELIM
   GENERAL  SGMLREF
   SHORTREF SGMLREF -- Removed short references --
   NAMES    SGMLREF
   QUANTITY SGMLREF
       ATTCNT    140
       LITLEN   4096
       NAMELEN    64
       GRPCNT    100
       GRPGTCNT  253
       TAGLVL     48
FEATURES
    MINIMIZE
       DATATAG    NO
       OMITTAG    NO
       RANK       NO
       SHORTTAG   YES
   LINK
       SIMPLE     NO
       IMPLICIT   NO
       EXPLICIT   NO
   OTHER
```

```
        CONCUR     NO
        SUBDOC     NO
        FORMAL     NO
        APPINFO  NONE
>
<!DOCTYPE helpvolume [
<!ELEMENT helpvolume   - - (metainfo?,
                            hometopic?,
                            (chapter* | (s1*, rsect*)),
                            message?,
                            glossary?)
                           +(memo | idx) >
<!ELEMENT metainfo     - - (idsection, abstract?, otherfront*)
                            -(footnote) >
<!ELEMENT idsection    - - (title, copyright?) >
<!ELEMENT title        - - (partext)
                           -(memo | location | idx) >
<!ELEMENT partext     - - ((#PCDATA | acro    | emph     | computer |
                            user    | term    | var      | circle   |
                            quote   | keycap  | graphic  | super    |
                            sub     | book    | xref     | footnote |
                            esc     | link    | location | newline )*)

<!ELEMENT acro         - - ((#PCDATA | esc | super | sub)*) >
<!ELEMENT emph         - - (partext) -(emph) >
<!ELEMENT computer     - - ((#PCDATA | quote | var | user | esc)*) >
<!ELEMENT user         - - ((#PCDATA | var | esc)*) >
<!ELEMENT term         - - (partext)
                           -(emph  | computer | term | var     |
                             quote | user     | book | footnote) >
<!ATTLIST term             base         CDATA              #IMPLIED
                           gloss        (gloss | nogloss) gloss >
<!ELEMENT var          - - ((#PCDATA | esc)*) >
<!ELEMENT circle       - - CDATA >
<!ELEMENT quote        - - (partext) -(quote) >
<!ELEMENT keycap       - - ((#PCDATA | super | sub | esc)+) >
<!ELEMENT graphic      - O EMPTY >
<!ATTLIST graphic          id           ID                 #IMPLIED
                          entity       ENTITY            #REQUIRED >
<!ELEMENT super        - - (#PCDATA) >
<!ELEMENT sub          - - (#PCDATA) >
<!ELEMENT book         - - (partext) -(book) >
<!ELEMENT xref         - O EMPTY >
<!ATTLIST xref            id           IDREF             #REQUIRED >
<!ELEMENT footnote     - - (p+) -(footnote) >
<!ELEMENT esc          - - CDATA >
```

```
<!ELEMENT link          - - (partext) -(link | xref) >
<!ATTLIST link              hyperlink    CDATA             #REQUIRED
                            type         (jump          |
                                          jumpnewview   |
                                          definition    |
                                          execute       |
                                          appdefined    |
                                          man           )    jump
                            description CDATA              #IMPLIED >
<!ELEMENT location      - - (partext) -(location) >
<!ATTLIST location          id           ID                #REQUIRED >
<!ELEMENT copyright     - - (text)
                             -(memo | location | idx) >
<!ELEMENT text        - - ((p       | note      | caution   | warning |
                          lablist |    list   |     ex      | vex     |
                          esc     | otherhead | procedure | syntax  |
                              figure  | image      )* ) >
<!ELEMENT p             - - (head?, partext)
                            +(newline) >
<!ATTLIST (p | image)       indent       (indent)          #IMPLIED
                            id           ID                #IMPLIED
                            gentity      ENTITY            #IMPLIED
                            gposition    (left | right)    left
                            ghyperlink   CDATA             #IMPLIED
                            glinktype    (jump          |
                                          jumpnewview   |
                                          definition    |
                                          execute       |
                                          appdefined    |
                                          man           )    jump
                           gdescription CDATA              #IMPLIED >
<!ELEMENT head          - - (partext)
                            -(memo | location | idx) >
<!ELEMENT newline       - O EMPTY >
<!ELEMENT (note    |
           caution |
           warning )   - - (head?, text)
                            -(note | caution | warning | footnote) >

<!ELEMENT lablist       - - (head?, labheads?, lablistitem+) >
<!ATTLIST lablist           spacing      (loose | tight)   loose
                            longlabel    (wrap  | nowrap)  wrap >
<!ELEMENT labheads      - - (labh, labhtext)
                            -(memo | location | idx) >
<!ELEMENT labh          - - (partext) >
<!ELEMENT labhtext      - - (partext) >
```

```
<!ELEMENT lablistitem  - - (label, text) >
<!ELEMENT label        - - (partext) >
<!ELEMENT list         - - (head?, item+) >
<!ATTLIST list             type         (order   |
                                         bullet  |
                                         plain   |
                                         check   )        bullet
                           ordertype    (ualpha  |
                                         lalpha  |
                                         arabic  |
                                         uroman  |
                                         lroman  )        arabic
                           spacing      (tight   |
                                         loose   )        tight
                          continue     (continue)        #IMPLIED >
<!ELEMENT item         - - (text) >
<!ATTLIST item            id           ID                #IMPLIED >
<!ELEMENT ex           - - (head?, (exampleseg, annotation?)+)
                           -(ex      |
           vex     |
           note    |
           caution |
           warning |
           syntax  |
           footnote) >
<!ATTLIST ex               notes        (side | stack)    side
                           lines        (number   |
                                         nonumber )       nonumber
                           textsize     (normal   |
                                         smaller  |
                                         smallest )       normal >
<!ELEMENT exampleseg   - - (partext) +(lineno) >
<!ELEMENT annotation   - - (partext) +(newline) >
<!ELEMENT lineno       - O EMPTY >
<!ATTLIST lineno          id           ID                #IMPLIED >

<!ELEMENT vex          - - CDATA >
<!ATTLIST vex              lines        (number   |
                                         nonumber )       nonumber
                           textsize     (normal   |
                                         smaller  |
                                         smallest )       normal >
<!ELEMENT otherhead    - - (head, text?) >
<!ELEMENT procedure    - - (chaphead, text?)
                           -(procedure) >
```

```
<!ELEMENT chaphead     - - (head, abbrev?)
                           -(memo | location | idx | footnote) >
<!ELEMENT abbrev       - - (partext) -(footnote) >
<!ELEMENT syntax       - - (head?, synel) >
<!ELEMENT synel        - - ((#PCDATA | esc | var |
                            optblock | reqblock  )+) >
<!ELEMENT (optblock |
           reqblock )  - - (synel+) >
<!ELEMENT figure       - - (caption?)
                           -(figure | graphic) >
<!ATTLIST figure           number       NUMBER            #IMPLIED
                           tonumber      (number  |
                                          nonumber)       number
              id           ID                 #IMPLIED
              entity       ENTITY             #REQUIRED
              figpos       (left   |
                            center |
                            right  )          #IMPLIED
              cappos       (capleft   |
                            capcenter |
                            capright  )       #IMPLIED
              ghyperlink   CDATA              #IMPLIED
              glinktype    (jump         |
                            jumpnewview |
                            definition   |
                            execute      |
                            appdefined   |
                            man          )    jump
              gdescription CDATA              #IMPLIED >
<!ELEMENT caption      - - (partext, abbrev?)
                           -(memo | location | idx) >
<!ELEMENT image        - - (head?, partext) -(footnote) >
<!ELEMENT abstract     - - (head?, text?, frontsub*) >
<!ELEMENT frontsub     - - (head?, text) >
<!ELEMENT otherfront   - - (head?, text?, frontsub*) >
<!ATTLIST otherfront      id           ID                #IMPLIED >
<!ELEMENT hometopic    - - (chaphead, text?) >
<!ELEMENT chapter      - - (chaphead, text?, (s1*, rsect*)) >
<!ATTLIST (chapter |
           s1      |
           s2      |
           s3      |
           s4      |
           s5      |
           s6      |
```

```
           s7       |
           s8       |
           s9      )       id          ID                #IMPLIED >
<!ELEMENT s1           - - (chaphead, text?, s2*, rsect*) >
<!ELEMENT s2           - - (chaphead, text?, s3*, rsect*) >
<!ELEMENT s3           - - (chaphead, text?, s4*, rsect*) >
<!ELEMENT s4           - - (chaphead, text?, s5*, rsect*) >
<!ELEMENT s5           - - (chaphead, text?, s6*, rsect*) >
<!ELEMENT s6           - - (chaphead, text?, s7*, rsect*) >
<!ELEMENT s7           - - (chaphead, text?, s8*, rsect*) >
<!ELEMENT s8           - - (chaphead, text?, s9*, rsect*) >
<!ELEMENT s9           - - (chaphead, text?) >
<!ELEMENT rsect        - - (chaphead, text?, rsub*) >
<!ATTLIST rsect            id          ID                #IMPLIED >
<!ELEMENT rsub         - - (chaphead, text?) >
<!ELEMENT message      - - (chaphead?, text?, (msg+ | msgsub+)) >
<!ELEMENT msg          - - (msgnum?, msgtext, explain?) +(newline) >
<!ELEMENT msgnum       - - ((#PCDATA | esc)+) >
<!ELEMENT msgtext      - - (partext) >
<!ELEMENT explain      - - (text) >
<!ELEMENT msgsub       - - (chaphead, text?, msg+) >
<!ELEMENT glossary     - - (text?, glossent+) >
<!ELEMENT glossent     - - (dterm, definition) >
<!ELEMENT dterm        - - (partext) -(term) >
<!ELEMENT definition   - - (text) >
<!ELEMENT idx          - - (indexprimary, indexsub?)
                           -(term | footnote | location | idx) >
<!ELEMENT indexprimary - - (partext, sort?) >
<!ELEMENT indexsub     - - (partext, sort?) >
<!ELEMENT sort         - - ((#PCDATA | esc)+) >
<!ELEMENT memo         - - CDATA >
<!ENTITY MINUS     SDATA "-">
<!ENTITY PM        SDATA '[plusmn]'>    <!-- ISOnum  -->
<!ENTITY DIV       SDATA '[divide]'>    <!-- ISOnum  -->
<!ENTITY TIMES     SDATA '[times ]'>    <!-- ISOnum  -->
<!ENTITY LEQ       SDATA '[le    ]'>    <!-- ISOtech -->
<!ENTITY GEQ       SDATA '[ge    ]'>    <!-- ISOtech -->
<!ENTITY NEQ       SDATA '[ne    ]'>    <!-- ISOtech -->
<!ENTITY COPY      SDATA '[copy  ]'>    <!-- ISOnum  -->
<!ENTITY REG       SDATA '[reg   ]'>    <!-- ISOnum  -->
<!ENTITY TM        SDATA '[trade ]'>    <!-- ISOnum  -->
<!ENTITY ELLIPSIS  SDATA '[hellip]'>    <!-- ISOpub  -->
<!ENTITY VELLIPSIS SDATA '[vellip]'>    <!-- ISOpub  -->
<!ENTITY PELLIPSIS SDATA "....">
<!-- ellipsis followed by a period -->
<!ENTITY A.M.      SDATA "a.m.">
```

```
<!ENTITY P.M.      SDATA “p.m.”>
<!ENTITY MINUTES   SDATA ‘[prime ]’>    <!-- ISOtech -->
<!ENTITY SECONDS   SDATA ‘[Prime ]’>    <!-- ISOtech -->
<!ENTITY DEG       SDATA ‘[deg   ]’>    <!-- ISOnum  -->
<!ENTITY SQUOTE    SDATA “`”>
<!ENTITY DQUOTE    SDATA ‘”’>
<!ENTITY ENDASH    SDATA “–”>
<!ENTITY EMDASH    SDATA ‘[mdash ]’>    <!-- ISOpub  -->
<!ENTITY VBLANK    SDATA “_”>
<!ENTITY CENTS     SDATA ‘[cent  ]’>    <!-- ISOnum  -->
<!ENTITY STERLING  SDATA ‘[pound ]’>    <!-- ISOnum  -->
<!ENTITY SPACE     SDATA “ ”>
<!ENTITY SIGSPACE  SDATA “& ”>
<!ENTITY SIGDASH   SDATA “&–”>
<!ENTITY MICRO     SDATA ‘[micro ]’>    <!-- ISOnum  -->
<!ENTITY OHM       SDATA ‘[ohm   ]’>    <!-- ISOnum  -->
<!ENTITY UP        SDATA ‘[uarr  ]’>    <!-- ISOnum  -->
<!ENTITY DOWN      SDATA ‘[darr  ]’>    <!-- ISOnum  -->
<!ENTITY LEFT      SDATA ‘[larr  ]’>    <!-- ISOnum  -->
<!ENTITY RIGHT     SDATA ‘[rarr  ]’>    <!-- ISOnum  -->
<!ENTITY HOME      SDATA “home key”>
<!ENTITY BACK      SDATA “<--”>
<!ENTITY HALFSPACE SDATA “ ”>

<!ENTITY % user-defined-entities SYSTEM “helptag.ent”>
%user-defined-entities;
] >
```

# Glossary

**application help**

Online help for a particular application (software).

**application-defined link**

A hyperlink designed especially for invoking some application behavior. To invoke the behavior, the help must be displayed in dialogs created by the application. (Application-defined hyperlinks are ignored by the Help Viewer.)

**automatic help**

Help presented by the system as the result of a particular condition or error. Sometimes called "system initiated" help. For example, error dialogs are a form of "automatic help." See also *semi-automatic help* and *manual help*.

**browser volume**

The desktop uses the Helpview program as a "help browser" by displaying a special browser volume that lists the help installed on the system. A utility called `dthelpgen` creates this volume in the user's home directory.

**caution**

A warning to the user about possible loss of data. See also *note* and *warning*.

**close callback**

An application function called when a help dialog box is closed.

**context-sensitive help**

Online information that is relevant to what the user is doing within an application. Sometimes, pressing the F1 key is referred to as "context-sensitive help" because the choice of help topic is based on the user's context.

**cross-volume hyperlink**

A hyperlink that jumps to a topic in a different help volume. Cross-volume hyperlinks are entered using the `<link>` element, where the `hyperlink` parameter specifies the volume name and an ID (separated by a space):

`<link hyperlink="`*volume*`">`*text*`<\link>`

**dialog cache**

A list of help dialogs that has been created but may not be in use. When the application needs a new help dialog, it first searches its dialog cache for an unused dialog. If one is found, it is used. Otherwise, all dialogs are in use, so a new one is created.

**Document Type Definition**

A description of a set of elements used to create a structured (or hierarchical) information. The Document Type Definition (DTD) specifies the syntax for each element and governs how and where elements can be used in a document.

**element**

A logical portion of information, such as a book title, a paragraph, a list, or a topic. Normally, the extent of an element is marked by *tags*, although the tags for some elements are assumed by context.

**emphasis**

An element of text that calls attention to the text (usually by being formatted as *italic*).

**entity**

A text string or file with a name. Most entities are named by the author (using the `<!entity>` element), but some entities are predefined. See also *entity declaration* and *entity reference*.

**entity declaration**

Markup that establishes an entity name and its value. See also *entity* and *entity reference*.

**entity reference**

Use of an entity name preceded by an `&` (ampersand) and followed by a `;` (semicolon) that indicates to HelpTag that the entity is to be inserted where the entity name appears. See also *entity* and *entity declaration*.

**entry point**

A point within a help volume that may be displayed directly as the result of a request for help. That is, a topic where the user may "enter" or begin reading online help. Any topic, or location within a topic, that has an ID can become an entry point.

**example listing**

A body of text in which line breaks are left as they are and which is displayed in a computer font. The text is typically an example of a portion of a computer file. Example listings are entered using the `<ex>` or `<vex>` elements.

**execution alias**

A resource that assigns a name to a command string or script that an execution link executes.

**execution link**

A hyperlink that executes a shell command or script.

**execution policy**

The Help System provides a resource that can be set to control the behavior of execution links. This enables a system administrator or user to establish an appropriate level of security for any given application.

**figure**

A graphic or illustration that appears in the help information.

**formal markup**

A tag set and accompanying usage rules that are specified in the Helptag 1.3 Document Type Defnition (DTD). By following the rules set forth in the DTD, an author can produce Standard Generalized Markup Language (SGML) compliant help source files.

**general help dialog box**

A window in which help information is displayed. General help dialog boxes have a menu bar, a topic tree (which provides a list of topics), and a help topic display area. See also *quick help dialogbox*.

**help callback**

An application function called when the user presses the F1 key.

**help family**

A set of help volumes that are related to one another because the applications they refer to are related.

**help key**

A designated key, usually the F1 function key, used to request help on the current context. Some keyboards have a dedicated Help key that may take the place of F1. In OSF/Motif applications, the help key is enabled by adding a help callback to a widget.

**help on help**

Help information about how to use the help dialog boxes. The user gets this information by pressing F1 while using a help window, or by choosing Using Help from the Help menu in a general help dialog box.

**help volume**

A complete body of information about a subject. Also, this term can refer to either the set of source files that contain the marked-up text or the run-time files generated by running HelpTag.

**History dialog box**

A dialog box that shows a list of the sequence of topics the user has visited. The history sequence can be traversed in reverse order to make it easy for the user to return to earlier topics.

**home topic**

The topic at the top of the hierarchy in a help volume. This is the topic that is displayed when the user indicates a desire to browse a help volume. HelpTag provides a built-in ID for the home topic: `_hometopic`.

**hyperlink**

A segment of text (word or phrase) or graphic image that has some behavior associated with it. The most common type of hyperlink is a "jump" link, which connects to a related topic. When the user chooses a jump link, the related topic is displayed. Hyperlinks can also be used to invoke other kinds of behavior, such as executing a system command or invoking specific application behavior.

**hyperlink callback**

An application function that is invoked when a user chooses a hyperlink. This function is responsible for handling the types of hyperlinks not handled automatically within the help dialog.

**index**

A list of important words and phrases that appear throughout a help volume. The index is an alphabetical list of the words or phrases that can be searched to find help on a subject. The Help System displays the index when the user chooses the Index button (in a general help dialog box). See also *Index Search dialog box.*

**Index Search dialog box**

A dialog box that shows a list of index entries for a help volume. An index can be displayed for the current volume, selected volumes, or all help volumes. A user can search the index for a word or phrase and any corresponding topics that contain the search string will be listed.

**inline graphic**

A small graphic (illustration) that appears within a line of text.

**jump-new-view hyperlink**

A hyperlink that, when chosen, displays its information in a new dialog box. Jump-new-view links are intended for cross-volume links. The user senses a "new context" by a new window being displayed.

**man page link**

A hyperlink that, if activated, displays a "man page," which is a brief online explanation of a system command. The information in man pages are not supplied through the HelpTag system.

**manual help**

A style of online help that requires the user to know what help is needed and how to get it. For example, most commands in a Help menu are considered "manual" help because the user chooses when and what to view. See also *automatic help* and *semi-automatic help*.

**note**

A message to the user that draws attention to important information. If the information is critically important, a caution or warning is used instead. See also *caution* and *warning*.

**parser**

The portion of the HelpTag software that reads the source files (which are created by the author) and converts them into run-time help files that the Help System dialogs can read. If the author uses markup incorrectly (or incompletely), the parser detects the problems and indicates that "parser errors" have occurred.

**quick help dialog box**

A streamlined help dialog box that has a help topic display area and one or more push buttons. See also *general help dialog box*, which offers additional capabilities.

**registration**

The process of declaring a help volume to be accessible for browsing or cross-volume linking.

**run-time help files**

The files generated by the `dthelptag` command. These are the files distributed to users who will use the Help System.

**Search Volume Selection dialog box**

A dialog box that lists the help volumes available on a user's system. When a user chooses Selected from the Index Search dialog box, this dialog box lists help volumes that the user can select. One or more volume names can be selected and the corresponding index information is reported in the Index Search dialog box.

**semi-automatic help**

A style of online help in which the user requests help and the system decides, based on the current circumstances, which help information to display. "Context-sensitive" help (pressing the F1 key) is an example of semi-automatic help. See also *automatic help* and *manual help*.

**short form markup**

An abbreviated way of marking an element where the end tag is marked with a single vertical bar and the last character of the begin tag is also a vertical bar. For example, the short form of the `<book>` element is:

`<book`|*text*|

**shorthand markup**

An abbreviated way of marking an element where the start and end tags are replaced with a special two-character sequence. For example, the shorthand form of the `<computer>` element is two opening single quotation marks followed by two closing single quotation marks like this:

``` ``text'' ```
```

**standalone help**

Help information intended to be used independently of application software. For example, online help that explains the basics of computer programming may not be associated with a particular application. A standalone help volume can be displayed using the `dthelpview` command.

**Standard Generalized Markup Language (SGML)**

An international standard [ISO 8879: 1986] that establishes a method for information interchange. SGML prescribes constructs for marking the structure of information separate from its intended presentation or format. The HelpTag markup language conforms to this SGML standard.

**tag**

A text string that marks the beginning or end of an element. A start tag consists of a < (left angle bracket) followed by a special character string (consisting of only letters), optional parameters and values, and terminated by a > (right angle bracket).

An end tag consists of a < (left angle bracket), a \ (backslash), the same special character string, and a > (right angle bracket). Formal markup uses a `/` (forward slash) in the end tag syntax.

**Tagged Image File Format (TIFF)**

A standard graphics file format. The Help System dialog boxes support TIFF 5.0 images. TIFF images are identified by the `.tif` file-name extension.

**topic**

Information about a specific subject. Usually, this is approximately one screenful of information. Online help topics are linked to one another through hyperlinks.

**topic hierarchy**

A help volume's branching structure in which the home topic branches out (through hyperlinks) to progressively more detailed topics. See also *home topic*.

**topic tree**

In a general help dialog box, a list of topics that can be selected to display help information.

**warning**

Information that warns the user about possible injury or unrecoverable loss of data. See also *caution* and *note*.

**widget**

The fundamental building block of graphical user interfaces. The OSF/Motif widget set provides widgets of all sorts, suitable for constructing an application user interface.

**X bitmap**

A two-tone image that has one foreground color and one background color. Bitmap image files are identified by the `.bm` file-name extension.

**X pixmap**

A multicolor image. Pixmap image files are identified by the `.pm` file-name extension.

**X window dump**

An image captured from an X Window System display. The `xwd` utility is used to capture a window image. X window dump image files are identified by the `.xwd` file-name extension.

# Index

# D

## E

## G



## H

# I

## J



## K



## L

## Q



## R



## S

# T

# X