HP Help System

Developer's Guide

Version 3.0

HP Part No. B1171-90077
Printed in USA January 1995

Second Edition
DRAFT 4/7/98 12:49

Notice

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Printing History

The printing date will change when a new edition is printed. Minor changes may be made at reprint without changing the printing date. The manual part number will change when extensive changes are made.

Manual updates may be issued between editions to correct errors or document product changes. To ensure that you receive these updates or new editions, see your HP sales representative for details.

July 1992 ... First Edition ... B1171-90055

January 1995 ... Second Edition ... B1171-90077

Hewlett-Packard Company
Workstation Technology Division - CVL
1000 NE Circle Boulevard
Corvallis, Oregon 97330 USA

Contents

1

Introducing the HP Help System

The **HP Help System** is a complete system for developing online help for application software. It allows authors to write online help that includes rich graphics and text formatting, hyperlinks, and communication with the application. HP Help provides a programmer's toolkit for integrating the help facilities into an application.

More specifically, here's what the HP Help System Developer's Kit includes:

For Authors

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

Refer to "The Author's Job" to learn more about writing online help.

For Programmers

- *The Xvh programming library*—an Application Programmer's Interface (API) for integrating help windows into your application.
- *A demonstration program*—a simple example that shows how to integrate the HP Help System into an OSF/Motif application.

Refer to "The Programmer's Job" to learn more about programming with the HP Help System.

This Manual Online

This manual is available online.

- If you are using the HP Visual User Environment (HP VUE), version 3.0 or later, use Help Manager. Open the "HP Help System" family, then choose the "HP Help System Developer's Guide" title.
- *Or,* if you are not using HP VUE, execute the following command in a terminal window:

```
/usr/vhelp/bin/helpview -helpVolume HPHelpKit &
```

Tip! 

The online version of this manual offers the time-saving advantage of making the examples accessible electronically. You can copy and paste them into your HelpTag files and then edit them to fit your needs.

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. Too much help can often be too much of an interruption or "context switch." In developing your online help, it's important to keep in mind the objectives for providing online help.

Objectives for Online Help

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

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

How Users Get 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.
- **Semi-Automatic Help**—The user decides when help is needed, but the system determines what to present. Semi-automatic help is initiated by a user's gesture or exclamation 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.

The 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 *OSF/Motif Style Guide* 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 have a Help key on the keyboard.

The 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 *OSF/Motif Style Guide* makes recommendations regarding the commands contained in a Help menu.

Help Buttons

Many dialogs also provide a Help button to get help on the dialog. The *OSF/Motif Style Guide* recommends that choosing the Help button in a dialog 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 is appropriate.

The HP Help Information Model

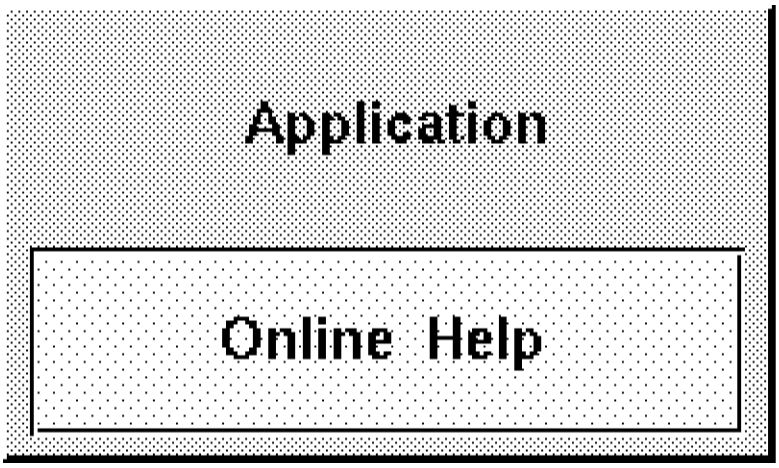
There are two general styles of online help:

- **Application help**, whose primary role is to be an integrated part of an OSF/Motif application.
- **Stand-alone 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 stand-alone help.

Part of the Application

HP 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.

A "volume" is a collection of "topics"

A **help volume** is a collection of related topics. Normally, the topics within a volume are arranged in a hierarchy.

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.

A "product family" is a group of related help volumes

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, the multi-volume set forms a product family.

Assembling your help volumes into a product family is optional. It is required only if you want your help to be available for browsing within a **help browser** such as the HP VUE Help Manager.

The top level of the HP VUE Help Manager lists product families. The second level, under each family, lists the volumes that are members of the product family.

Even if you have only a single help volume, it must belong to a product family to be browsable via the HP VUE Help Manager.

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.

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 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 HP VUE Help Manager), making the information more widely accessible. If another help volume contains hyperlinks to topics in your help volume, your help volume must be registered.

Stand-Alone Help Volumes

If you are writing a stand-alone help volume (a help volume not associated with an application) you may choose to do things very 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 via at least one hyperlink. Also, because there's no application associated with your help, you must rely on a help viewer (such as Helpview) to display your help volume.

Collaborate with the application programmer

If you are writing application help, you should work closely with the application programmer. The degree to which the HP 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.

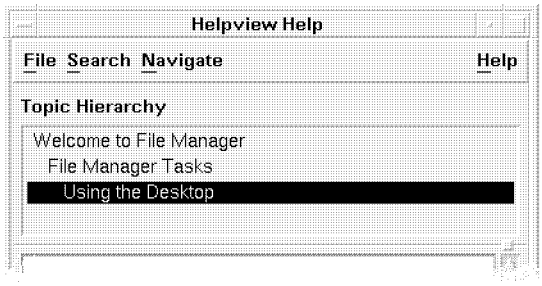
In contrast, the application could provide specific help for nearly every situation in the application. This requires more work, but benefits the user if done well.

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

Organize and write your topics

The HP Help System best supports a hierarchical organization of topics. This familiar way of organizing information helps users know where they are when viewing a particular topic.

The General Help Dialog provided by HP Help includes a "Topic Hierarchy" list that indicates the path from the **home topic** (at the top of the hierarchy) to the current topic. Essentially, this is "you are here" information for the user.



Although HP Help has been optimized for information that is organized in a hierarchy, you are free to create any kind of organization you want. The Quick Help Dialog is a simpler help window that does not include the Topic Hierarchy. In this window you can present topics that are not organized in a hierarchy. Using **hyperlinks** you can connect related topics organized in any way you want, including "webs," "chains," and "loops."

Writing 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 HP 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" in Chapter 2 describes the details of using markup. Chapter 11 includes a description of each element and the tags needed to enter it.

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>OSF/Motif Style 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, HP 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.

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. There are two ways to do this:

- Using the `helpview` command, you can display any help topic in any help volume.
- If you are writing application help and the HP 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.

The Programmer's Job

The programmer adds code to an 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.

Example Program

The `/usr/vhelp/examples/helpdemo/` directory contains the source code for a sample program called `helpdemo`. 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 taking into account:

- What confusions 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.

Create and manage help dialogs

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

- The *quick help dialog*, which provides a simple help window with a topic display area and a few dialog buttons.
- The *general help dialog*, which provides a help window that includes a menu bar and a topic hierarchy in addition to the help topic display area.

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

2

Organizing and Writing a Help Volume

Within a help volume, information is organized into five major sections: the home topic, topics and subtopics, entity declarations, meta information, and the glossary.

Home Topic

The **home topic** is the first in the topic hierarchy. 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. The first level of subtopics (following `<hometopic>`) may begin with `<chapter>` or `<s1>`. The `<chapter>` level is the only level you can skip. That is, subtopics of an `<s1>` must be entered with `<s2>`, subtopics of an `<s2>` must be entered with `<s3>`, and so on. There is no visible difference to the user if you start your hierarchy with `<chapter>` or `<s1>`. The only difference is the number of run-time help files that will be generated (one per chapter).

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. All entity declarations must be entered before any other markup in your help volume. Entity **references** can be used anywhere within your help volume. When you process your help volume with the HelpTag software (`helptag` command), each entity reference is replaced with the text or file that the entity represents.

Meta Information

Meta information is *information about your information*. It includes information such as the volume's title, copyright notice, and abstract. Meta information also includes general help topics that are *not* part of the normal topic hierarchy.

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.

A Help Volume at a Glance

The following markup illustrates important elements of a help volume and the tags used to enter them. 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
                 Body of the home topic goes here ...

          <s1> Title of First Subtopic Goes Here
               Body of the first subtopic goes here ...

          <s1> Title of Second Subtopic
               Body of the second subtopic goes here ...

            ⋮

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

<\helpvolume>
```

The rest of this chapter describes how to organize your help volume. Writing individual topics is covered in the following chapter.

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 HP 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.

Creating Your *volume*.htg File

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

HelpTag expects a primary input file named *volume*.htg, where *volume* is a name you choose. 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 HP VUE File Manager is "Vuefile," so its helpvolume was named Vuefile.htg.

The details of running HelpTag are covered in "To run 'helptag'" in Chapter 4.

Multiple Source Files

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.)

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>The OSF/Motif Style Guide<\book>
```

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

Short Form Markup

Short form markup provides another way to enter the markup for many inline elements. 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 above:

```
<book|The OSF/Motif Style Guide|
```

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

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

Shorthand Markup

Some elements support an even shorter form where the begin 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* ... `!!`

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 `&&`).

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

Here is a complete step-by-step example that demonstrates how to create and view a stand-alone help volume. (As a stand-alone volume, it does not involve interaction with an application.)

The markup language used in the text files is explained in Chapter 2, Chapter 3, and Chapter 11.

1. Create a directory where you will put most of your text files.
2. Create one or more text files that explain the system you are documenting. For this simple example, all the information is put into a single file named `Commands` in the directory just created.

Tip! 

If you want to try this example, copy and paste the example text directly from online help into your text editor, instead of typing the text.

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

```
<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>)
```

3. 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; 1997 Voice Activation Company.  All rights reserved.

     <abstract>Help for Using the &product;.

<\metainfo>
```

Create a "build" directory

4. Create a new subdirectory named `build/` that is below the directory that contains the `Commands` and `Metainfo` files.

Create the master HelpTag file

5. 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`.

6. 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;
```

Prepare to run HelpTag

7. 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.

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

8. Add the `/usr/vhelp/bin/` directory to your PATH environment variable. If you're not sure how to do this, consult the online help or documentation for your operating system or see your system administrator.

Create the run-time help files

9. From the `build/` subdirectory, execute the following command:

```
helptag -verbose voiceact
```

   This command executes the HelpTag software to create a run-time version of your online help volume. The `-verbose` option tells HelpTag to display, on your screen, its progress.

10. If HelpTag reports that errors occurred, fix them by editing or renaming the text files as needed. (If the errors are parser errors, they are listed in a file named `voiceact.err`.)

Display the help volume

11. From the `build/` subdirectory, execute the following command. It displays the new help volume:

```
helpview -h voiceact &
```

    You can now scroll the information and jump to related information by choosing hyperlinks.

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 hierarchies down to `<s9>`. However, information more than three or four levels deep often leads many readers to feel lost.

Within each topic that has subtopics, you should provide a path for the reader to get to the subtopics. This requires assigning unique IDs to all topics and creating hyperlinks within the body of each topic to its subtopics.

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
- *How Report Hot Links Work*
- *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> How Report Hot Links Work
    <s1> Sharing Reports within a Workgroup

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

You could have created an identical hierarchy by starting with `<s1>`s in place of the `<chapter>` tags, `<s2>`s for the next level, and `<s3>`s for the third level. The only difference—not seen by the reader—is the number of files created by HelpTag when you process the help volume.

Again, 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" in this chapter describes assigning IDs to topics.
- "Creating Hyperlinks" in Chapter 3 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.
```

To help users who may be new to hyperlinks, you may want to include a reminder to use F1 to get help on help.

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

The body of this getting started topic should
include a hyperlink to each of the subtopics.

<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>`.

The Parent-Child Metaphor

Sometimes a parent-child-sibling metaphor is used to describe the relationships between topics in a hierarchy. In the above 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 "Notice" 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 general help topics that are not part of the normal topic hierarchy. These topics are useful for creating custom definition links that pop-up a topic in a quick help dialog.

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>`

Notes

- 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 may use the abstract to present a brief description of the volume. Because the abstract might be displayed in plain text windows (that are not capable of multi-font or graphics formatting), you should 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 Hewlett-Packard Company 1992
    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 `&copy;` entity inserts the copyright symbol.

See Also

- "To link to a meta information topic" in Chapter 3

To add a non-hierarchical 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 may 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 "Popup!" and its ID is `my-popup-topic`.

```
<metainfo>
  <title>My Help
  <copyright>
     This is My Help, Version 1.0.  &copy; 1992.
        .
        .
        .

  <otherfront id=my-popup-topic><head>Popup!

  This is a popup 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 "Popup!" topic in a quick help dialog.

See Also

- "Creating Hyperlinks" in Chapter 3

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.

Rules for ID Names

- ID strings may contain letters (A - Z and a - z), digits (0 - 9), plus (+), and minus (−), 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 pre-defined 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>
<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.

Examples

Here's a figure with the ID `my-big-picture`. Whenever you assign an ID to a figure, be sure to provide a caption.

```
<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> -- just press F1.
```

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 `helptag` 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

- All entity declarations must come before any other markup in your help volume.
- Entity names may contain letters (A - Z and a - z), digits (0 - 9), plus (+), and minus (−), 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 "`&`" and "`;`" characters are required for the HelpTag software to properly recognize the entity reference.

Example

The following line declares a text entity named "ProductName" that contains the string "HP Visual User Environment":

```
<!entity ProductName "HP Visual User Environment">
```

The following sentence includes a reference to the entity:

```
Welcome to the &ProductName;!
```

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 HP Visual User Environment!
```

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* `... >`

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 run 'helptag'" in Chapter 4.)

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

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

See Also

- "Displaying Graphics" in Chapter 3

3

Writing a Help Topic

Each topic you write should have an element that marks the start of the topic:

`<`*element* `id=`*id*`>` *Topic 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 *Topic 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 *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 *Topic 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 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** let you partion sections within a topic.
- **Graphics** can be included within your text as *inline graphics* or displayed between paragraphs as stand-alone *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** enables you 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):

```
Many people prefer online help over printed
manuals because finding information can be
done more quickly.  For those people, HP Help
works well on screen.

For people who prefer to read a printed page,
HP Help offers full WYSIWYG printing.  Printed
output includes multi-font text formatting and
graphics.
```

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>
Hewlett-Packard Company
User Interface Technology Division
Corvallis, Oregon
<\image>
```

See Also

- "To wrap text around a graphic" later in this chapter shows how to include a graphic with a paragraph.

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 (`*`).

- *Or*, 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`.

- *Or*, to create a labeled list 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>`

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>
* chocolate
* raspberry
* vanilla
<\list>
```

To format the same list without bullets or numbers and with reduced spacing between items, you would use:

```
<list plain tight>
* chocolate
* raspberry
* vanilla
<\list>
```

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\ How to install the system.
  \Chapter 3\ How to use the system.
  \Appendix A\ A quick-reference description of all system features.
<\lablist>
```

See Also

- "<list>" in Chapter 11 summarizes the use of the `<list>` element and provides additional examples.
- "<lablist>" in Chapter 11 summarizes the use of the `<lablist>` and `<labheads>` elements and provides additional examples.

To provide subheadings within a topic

- For midsize 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 affect the topic hierarchy.

Example

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

```
<procedure>Online Help is Preferred by Many

Many people prefer online help over printed
manuals because finding information can be
done more quickly.  For those people, HP Help
works well on screen.
```

```
<procedure>WYSIWYG Printing!

For people who prefer to read a printed page,
HP Help offers full WYSIWYG printing.  Printed
output includes multi-font text formatting and
graphics.
```

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 show:

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

- For computer listings that *do* 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`.

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 reader can scroll to see all the example text.

See Also

- "To display a computer literal"
- "<ex>" in Chapter 11
- "<vex>" in Chapter 11

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 pre-defined 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, simply 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`). Sample icons are provided in `/usr/vhelp/helptag/icons/`.

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 allocated.

Example

If you declare the following entity:

```
<!ENTITY CautionElementDefaultIconFile  FILE  "caution.pm">
```

and include the following note and caution in your help volume:

```
<note>
  Pay attention -- this is important.
<\note>

<caution>
  Don't try this at home!
<\caution>
```

then the note is displayed *without* an icon, and the warning is displayed using the `caution.pm` graphic as its icon. (The `caution.pm` file must be in your HelpTag search path, which is specified in your `helptag.opt` file.)

See Also

- "To run 'helptag'" in Chapter 4 includes information about using a `helptag.opt` file.
- "Using Entities" in Chapter 2

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 HP Help System Developer's Guide|
```

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 marked to be displayed in computer text.

```
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

Here's a variable within a normal sentence.

```
When you enter your %%password%%, the
computer unlocks your screen and keyboard.
```

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'' 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

HP Help supports five types of hyperlinks:

- *Traditional hypertext* to "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* to 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* to display any man page installed on the system.
- *Execution links* to execute a shell command or program. This greatly expands the possibilities for what happens when the user activates a hyperlink.
- *Application-defined links* to create custom links that the application interprets. This provides facilities for communication between the help system and the application.

To create a "jump" 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.

- *Or,* to jump to a topic (within the same volume) that has a pre-defined ID, use the `<link>` element as shown:

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

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

- *Or,* 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 to the volume's *volume*`.hv` file.

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

- *Or,* if you are linking to an element with a title, use the `<xref>` element as shown:

  `<xref` *id*`>`

  When you use `<xref>` to create a link, the title of the topic with the ID of *id* is inserted in place of the `<xref>` element and becomes the active hyperlink.

  Also, `<xref>` always creates a standard "jump" link; other types of links must be created using the `<link>` element.

Note

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.

Examples

Here's a simple hyperlink to the topic with the ID "Welcome." (Notice that capitalization of the ID is not significant.)

```
This is a <link welcome>simple jump<\link> link.
```

Here's the same link, but the title of the `Welcome` topic is inserted using an `<xref>`:

```
Refer to the <xref welcome> topic.
```

Here's a link to the same topic, but this one requests a new window:

```
This is a <link welcome JumpNewView>new-view jump<\link> link.
```

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

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

This link jumps to the home topic of the `Vuefile` help volume:

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

See Also

- "<link>" in Chapter 11
- "<xref>" in Chapter 11

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.

- *Or*, 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.

- *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 to the volume's *volume*`.hv` file.

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 (underlined) hyperlink text.

See Also

- "Creating a Glossary" later in this chapter.
- "<term>" in Chapter 11
- "<link>" in Chapter 11

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 by hand.

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" in Chapter 6. (The Helpview application 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 than 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>" in Chapter 11

To create an execution link

- Use the `<link>` element with the `Execute` parameter as shown:

  `<link hyperlink="`*command* `&" Execute>`*text*`<\link>`

  Where *command* is the command string you want to execute and *text* is the portion of your help text that you want to use as the underlined hyperlink text.

Note

If the command you are executing doesn't finish immediately, you should 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.

Example

The following link starts the `xclock` program running in the background:

```
<link hyperlink="xclock &" Execute>Start the clock<\link>
```

The phrase "Start the clock" becomes the underlined hyperlink text.

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 underlined hyperlink text.

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 prepared to handle these special links and interpret the hyperlink strings, it could generate the appropriate report based on the hyperlink chosen by the user.

To link to a meta information topic

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

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

  Where *id* is the pre-defined 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 pre-defined IDs, so they *do not* allow author-defined IDs. The pre-defined 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).

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

See Also

- "To add an ID to a topic" in Chapter 2 lists the predefined IDs for meta information topics.

Displaying Graphics

HP 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 filename extensions listed above.

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 to be displayed above the graphic.

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

If you want the figure to be a hyperlink, use the 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, in which case 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>" in Chapter 11
- "<link>" in Chapter 11

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. Larger images may overlap the text on the preceding line.

Example

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

```
<!entity SmallStopSign  FILE "stopsign.pm">
```

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

```
Whenever you see the <graphic entity=SmallStopSign> symbol,
stop and think about what you are doing.
```

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*`">`

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

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 sample-multicolor-icon  FILE "sample.pm">
```

Then, enter the paragraph:

```
<p gentity=sample-multicolor-icon>Many HP VUE components
support multicolor icons, in addition to the two-color
images used in previous versions of HP VUE.
```

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

```
<p gentity=sample-multicolor-icon gposition=right>Many
HP VUE components support multicolor icons, in addition
to the two-color images used in previous versions of HP VUE.
```

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 HP VUE components support
multicolor icons, in addition to the two-color images
used in previous versions of HP VUE.
```

See Also

- "<p>" in Chapter 11

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`. To access them, 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 12 for a complete list of the available characters.

To include a special character

1. Refer to Chapter 12 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*`;`

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

   ```
   search=/usr/vhelp/helptag
   ```

   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.

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 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 `helptag` 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.

```
helptag  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 a Keyword Index

The keyword index for a help volume is similar to the index for a book. The keyword index is displayed as a sorted list of the keywords you marked using the `<idx>` element.

In any general help dialog displaying your volume, the user can access the keyword index by choosing Keyword from the Search menu (Ctrl+K). The Filter field in the dialog lets the user reduce the number of items by typing an initial letter or two. When a keyword is selected, the titles of the topics in which it occurs are listed. To display a topic, the user selects one of the listed titles.

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>" in Chapter 11
- "<glossary>" in Chapter 11
- "<term>" in Chapter 11

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 prescribes constructs for marking the
structure of information separate from its
intended presentation or format.  The HP HelpTag
markup language is based on the SGML standard.
```

4

Processing and Displaying a Help Volume

Before a help volume can be displayed, you must create *run-time help files* by processing your files with the HelpTag software.

What HelpTag Does

The HelpTag software, which is invoked by the `helptag` command, does three significant tasks:

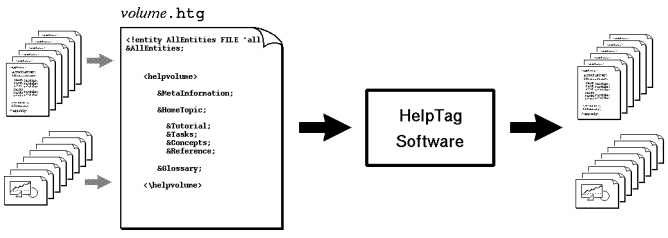
1. The HelpTag **parser** converts your marked-up files into an internal format understood by the the HP Help System. If you've made any markup errors, the errors are listed in a file named *volume*`.err`.
2. If there are no parser errors, the master help volume file (*volume*`.hv`) and keyword index file (*volume*`.hvk`) are created.
3. Finally, the help topic files (*volume*`*.ht`) are compressed to save disk space.

Viewing Your Volume

When HelpTag finishes, your help volume is ready to be displayed. You can display it using the `helpview` command. Or, 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 (by using the `helptag` command), it reads your *volume*`.htg` file and any additional source files that are included using entities. Also, graphics file names are validated.



The output from HelpTag is a set of run-time help files. These files, plus your graphics files, are read by the HP Help System to display help topics.

HelpTag Output

All the run-time help files have the same base name as your *volume*`.htg` file. For example, if your *volume*.htg is named `DeskScan.htg`, then each generated file will start with `DeskScan`.

See Also

For more information about the files generated by HelpTag, refer to "Gathering Run-Time Help Files" in Chapter 8.

To run 'helptag'

1. Be sure the `/usr/vhelp/bin/helptag` command is in your search path. (If you're not sure how to do this, contact your system administrator.)
2. Change to the directory where your *volume*`.htg` file is located.
3. Run the `helptag` command as follows:

   `helptag` *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 ('helptag')" in Chapter 13 lists all available options.

Example: Commands

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

```
helptag MyVolume
```

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

```
helptag -verbose MyVolume
```

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

```
helptag -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.

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:
...he 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.

Viewing a Help Volume

The Helpview application 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 run 'helpview'

- If the *volume*`.hv` file for the volume you want to display is either in the current directory or has been "registered," execute this command:

  `helpview -helpVolume` *volume*

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

  `helpview -helpVolume /`*full-path*`/`*volume*`.hv`

- *Or,* if you are using HP VUE 3.0 (or later), open a File Manager view of the directory where the *volume*`.hv` file is and double-click its icon. The default action for a `.hv` file is to run `helpview`.

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

Example

Suppose you just edited your help volume. You would first process it with the HelpTag software:

```
helptag MyVolume
```

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

```
helpview -h MyVolume &
```

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/rivers/help/` and your help volume is named `Myvolume`.

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

```
mkdir -p $HOME/vhelp/volumes/C
```

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

```
ln -s /projects/rivers/help/Myvolume.hv $HOME/vhelp/volumes/C/Myvolume.hv
```

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

```
helpview -helpVolume Myvolume
```

To display the online version of the guide you are now reading, you can execute this command from any directory:

```
helpview -helpVolume HPHelpKit
```

See Also

- "Displaying Help Topics ('helpview')" in Chapter 13 lists options available for the `helpview` command.

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 (they are not underlined).
- 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 from Helpview does not ensure that the links will operate correctly with 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 Helpview for each ID:

  `helpview -helpVolume` *volume* `-locationId` *id*

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

Testing Graphics

- Physically run your application on various displays to verify that the graphics are acceptable on color, grayscale, and monochrome displays.
- Print your help volume to verify that the printed graphics are acceptable. Graphics appear in hardcopy the same as on a monochrome display.
- If you are using HP VUE 3.0 or later, you can also simulate other situations by changing the "HP VUE Color Use" in Style Manager's Color dialog.

See Also

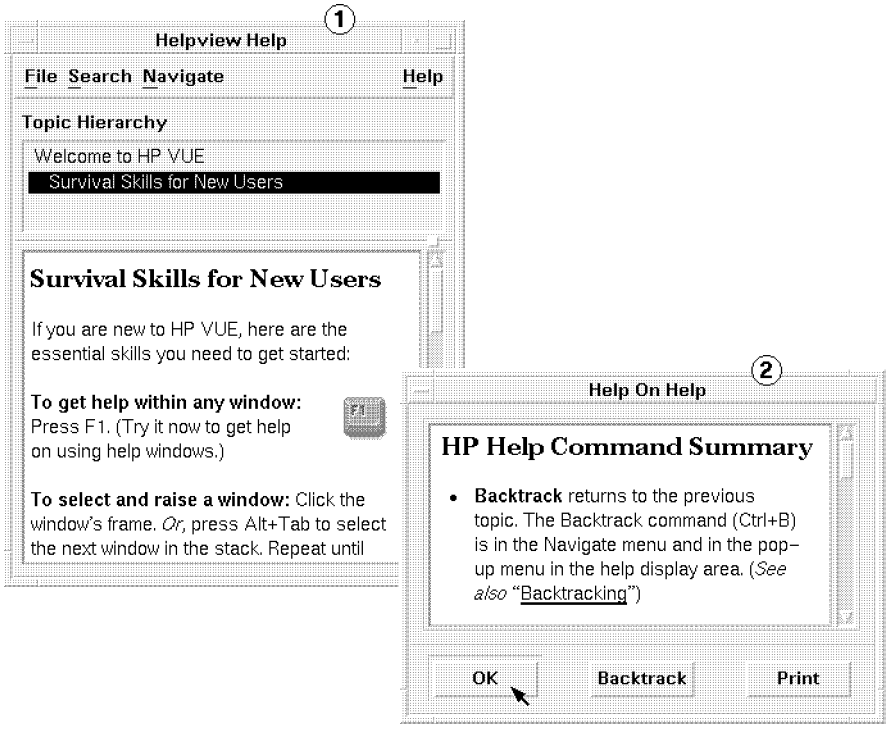
- "Printing Help Topics ('helpprint' and 'helpprintrst')" in Chapter 13

5

Creating and Managing Help Dialogs

For application programmers, the HP Help System provides a programming library that adds help dialogs to any OSF/Motif application. The library provides two types of help dialogs:

- **General help dialogs** have a menu bar, a Topic Hierarchy (that tells you where you are), and a help topic display area. (See ① below.)
- **Quick help dialogs** have just a topic display area and one or more dialog buttons at the bottom. (See ② below.)



The Quick Help Dialog

The quick help dialog is designed to help you meet the first objective of online help: *"Get the user back on task as quickly and successfully as possible."*

The quick help dialog, which never has more than five buttons, has a simple user interface, which helps keep the user focused on the information. Hopefully the information is useful enough that the user dismisses the dialog after reading it and continues working.

The General Help Dialog

The general help dialog has a few user interface features beyond the features of the Quick Help Dialog. Most notably, the Topic Hierarchy list, which appears just above the help topic display area, indicates the location of the current topic within the hierarchy. The home topic is always the first title in the hierarchy and the current topic's title is always the last title in the list.

Standard Xt Paradigm

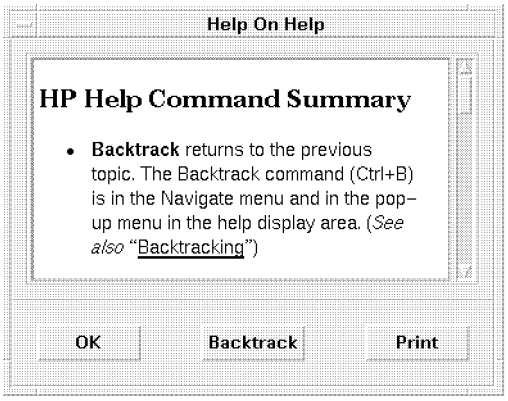
In terms of programming, you interact with the help dialogs the same as you do with any other OSF/Motif widgets in your applications. The two types of help dialogs are defined as two new widget classes: XvhQuickHelpDialogWidgetClass and XvhHelpDialogWidgetClass.

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 `XmNhelpVolume`, `XmNlocationId`, and `XmNhelpType` resources. For more information, refer to "Displaying Help Topics" in Chapter 6.

Prerequisite Knowledge

Integrating the HP Help System into an application requires a working knowledge of the the C programming language, the OSF/Motif programmer's toolkit, and the Xt Intrinsics toolkit.

The Quick Help Dialog



Within a quick help dialog, users can get to more help using hyperlinks within the displayed topic.

One of the dialog's buttons is application-defined, 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."
- *Or,* 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 HP Help programmer's toolkit includes a convenience function 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 Xvh.h
#include QuickHelpD.h
```

2. Create an instance of the quick help dialog widget:
   - Use the `XvhCreateQuickHelpDialog()` convenience function.
   - *Or,* use the `XtCreateManagedWidget()` function.
3. Add a callback for handling hyperlink events that occur within the dialog. (This is described in more detail in "Responding to Hyperlink Events" in Chapter 6.)
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-defined 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-defined button is enabled and used to request "more" help.

```
Widget   quickHelpDialog, moreButton, helpButton;

ac = 0;
XtSetArg (al[ac], XmNtitle, "My Application - Help");  ac++;
quickHelpDialog =
  XvhCreateQuickHelpDialog (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, XmNhyperLinkCallback,
               HyperlinkCB, (XtPointer)NULL);
XtAddCallback (quickHelpDialog, XmNcloseCallback,
               CloseHelpCB, (XtPointer)NULL);
```

Here, the application-defined button is managed and assigned an activate callback that invokes the application's `MoreHelpCB()` function.

```
moreButton = XvhQuickDialogGetChild (quickHelpDialog,
                                     XvhDIALOG_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.

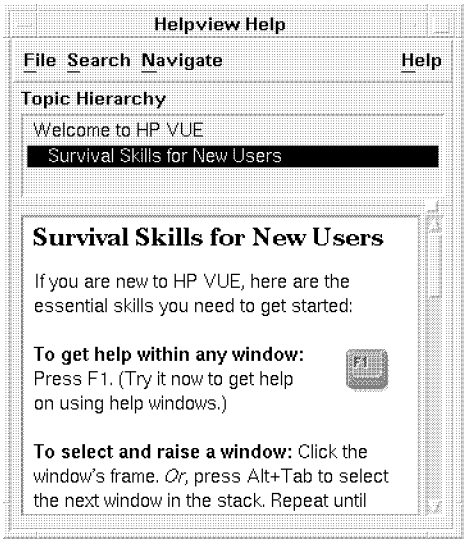
```
helpButton = XvhQuickDialogGetChild (quickHelpDialog,
                                     XvhDIALOG_HELP_BUTTON);
XtManageChild (helpButton);
XtAddCallback (quickHelpDialog, XmNhelpCallback,
               HelpRequestCB, USING_HELP);
```

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-defined button" in Chapter 6
- "To provide help on help for a quick help dialog" in Chapter 9

The General Help Dialog



The menu bar contains several commands, including commands that display these additional dialogs:

- Choose Print from the File menu to display the **Print** dialog. This dialog lets the user choose which topics are to be printed: "All," "Current," and "Current and Down." (Current and Down prints the current topic and all of its subtopics.)
- Choose Keyword from the Search menu to display the **Keyword Index** dialog. This dialog lists all the words and phrases that the author has marked as index entries. Selecting a keyword, then one of the topics where the keyword occurs, displays that topic in the general help dialog.
- Choose History from the Search menu to display the **History** dialog. This dialog lists the topic titles for each topic the user has visited. To return to any topic in the list, select its title.
- Choose Using Help from the Help menu to display the **Help On Help** dialog. This help information describes how to use the help system itself.

To create a general help dialog

1. Include the appropriate header files:

```
#include Xvh.h
#include HelpDialog.h
```

2. Create an instance of the general help dialog widget:
   - Use the `XvhCreateHelpDialog()` convenience function.
   - *Or,* use the `XtCreateManagedWidget()` function.
3. Add a callback for handling hyperlink events that occur within the dialog. (This is described in more detail in "Responding to Hyperlink Events" in Chapter 6.)
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++;
mainHelpDialog =
  XvhCreateHelpDialog (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, XmNhyperLinkCallback,
               HyperlinkCB, (XtPointer)NULL);
XtAddCallback (mainHelpDialog, XmNcloseCallback,
               CloseHelpCB, (XtPointer)NULL);
```

See Also

- Chapter 9
- "To enable the application-defined button" in Chapter 6

Creating a Dialog Cache

Because authors can create hyperlinks that request a new window, your application must be able to create an arbitrary number of help windows. But, creating and destroying widgets as they are needed can cause your application to run slower. So, to optimize performance and make efficient use of resources, caching help dialogs is recommended.

A **dialog cache** is a list of the help dialogs that your application has already created. When the user dismisses a dialog, the widget stays in the cache instead of being destroyed.

The next time the user requests help that would otherwise require a new widget, your application can scan the cache list looking for a dialog that isn't currently being used.

To create a dialog cache

1. Declare a structure that you can use to create a linked list of dialogs. Typically, a help dialog cache structure has these attributes:
   - Keeps track of the Widget ID for each help dialog.
   - Maintains a flag that indicates which type of help dialog it is (quick help or general help).
   - Maintains an "in use" flag.
2. When you create a help dialog, be sure to add it to your cache.
3. When the user closes a help dialog, return the dialog to your cache by clearing its "in use" flag, then unmanage it.

Example

The following type definition demonstrates a simple structure that you could use to build a dialog cache. Instances of this structure would be connected (via the `next` element) to form a linked list.

```
typedef struct _HelpDialogCacheStruct {
     Widget   dialog;                          /* The dialog's handle. */
     Boolean  inUse;                           /* The 'in use' flag. */
     Boolean  isQuickHelpDialog;               /* The dialog type flag. */
     struct   _HelpDialogCacheStruct  *next;   /* Next in the list. */
}  HelpDialogCacheStruct;
```

To retrieve a dialog from your cache

- Scan the cache looking for a dialog of the correct type whose "in use" flag is false.

  If there are no dialogs available in the cache, create a new one, add it to the cache (marked "in use") and use it.

Example

The following `FetchHelpDialog()` function scans a dialog cache for a help dialog of the specified type and returns a Widget ID. If an unused dialog is not found, one is created, added to the cache and returned. (If you are viewing this example online, you can copy and

paste this example directly into your source code to save the trouble of typing it.)

```
Widget
FetchHelpDialog(Boolean  lookingForQuickHelpDialog)
  {
     /* Declare a local pointer for walking the cache. */
     HelpDialogCacheStruct *pCacheStruct;

     /* Declare another, in case we need to add a new item to the cache. */
     HelpDialogCacheStruct *pNewCacheStruct;

     /* Set the local pointer to the first item in the cache list. */
     pCacheStruct = pFirstHelpDialogCacheStruct;

     /* Scan the cache for an unused help dialog. */
     while (pCacheStruct != (_HelpDialogCacheStruct)NULL)
       {
          /* Is this dialog available? */
          if (  (pCacheStruct->inUse == False)
                  /* And, is it the correct type? */
              &&  (  (   lookingForQuickHelpDialog
                      && pCacheStruct->isQuickHelpDialog)
                  || (   !lookingForQuickHelpDialog
                      && !pCacheStruct->isQuickHelpDialog)))
            {
               /* Yes!  This is a match. */
               pCacheStruct->inUse = True;
               return ((Widget)pCacheStruct->dialog);
            }
          else
            {
               /* Nope.  Go on to the next item. */
               pCacheStruct = pCacheStruct->next;
            }
       }

     /* Searching the cache was unsuccessful.*/

     /* Create a new item in the cache. */
     pNewCacheStruct = (HelpDialogCacheStruct *)
          XtMalloc((sizeof(HelpDialogCacheStruct)));

     /* Fill in the new structure. */
     pNewCacheStruct->inUse = True;
     pNewCacheStruct->isQuickHelpDialog =
          lookingForQuickHelpDialog;
     pNewCacheStruct->next =
          pFirstHelpDialogCacheStruct->next;
     pFirstHelpDialogCacheStruct = pNewCacheStruct;

     /* Create the new help dialog widget. */
     ac = 0;
     XtSetArg (al[ac], XmNtitle, "My Application - Help"); ac++;
     if (lookingForQuickHelpDialog)
       {
          pNewCacheStruct->dialog = XvhCreateQuickHelpDialog
               (topLevel, "quickHelpDialog", al, ac);
       }
     else
       {
```

```
            pNewCacheStruct->dialog = XvhCreateHelpDialog
                 (topLevel, "helpDialog", al, ac);
          }

        /* Done.  Return the new dialog. */
        return ((Widget)pNewCacheStruct->dialog);
    }
```

The example above assumes the following:

- That the pointer to the head of the cache (`pFirstHelpDialogCacheStruct`) is initialized to NULL during application start up.
- That the calling routine tests the return value to determine if a valid widget ID is returned.
- That the parent for all help dialogs is a widget named `topLevel`.
- That the variables `al` and `ac` (used in Xt argument lists) are declared external to this function.
- The help volume name is "MyApplication."

To return a dialog to your cache

- Unmanage the dialog and clear the dialog's "in use" flag.

Example

The following function is called when a help dialog is closed (via the close callback):

```
Boolean
HelpCloseCB (
     Widget     closedDialog,
     XtPointer  clientData,
     XtPointer  callData )
  {
    /* Declare a local pointer for walking the cache. */
    HelpDialogCacheStruct *pCacheStruct;

    /* Search the cache list for dialog. */
    pCacheStruct = pFirstHelpDialogCacheStruct;

    while (pCacheStruct != (HelpDialogCacheStruct *)NULL)
      {
        if (pCacheStruct->dialog == closedDialog)
          break;
        pCacheStruct = pCacheStruct->next;
      }

    /* Unmanage the dialog. */
    XtUnmanageChild(closedDialog);

    /* If the dialog wasn't found, the cache is corrupt.  Return failure. */
    if (pCacheStruct == (HelpDialogCacheStruct *)NULL)
      return (False);

    /* Mark the dialog unused, then return success. */
    pCacheStruct->inUse = False;
    return (True);
  }
```

6

Responding to Help Requests

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 three general ways for users to request help:

- By pressing the **help key** (which is F1 on most keyboards).
- By choosing the Help button in a dialog.
- By choosing a command from the application's Help menu.

Displaying 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 HP Help System's help dialogs are also capable of displaying man pages, text files, and simple text strings.

Displaying Help Topics

The HP 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 `XmNhelpType` resource determines what type of information is displayed. It can be set to any of these values:

- `XvhHELP_TYPE_TOPIC` for displaying normal help topics that are part of a help volume. The volume is specified by setting the `XmNhelpVolume` resource; the topic is specified by setting the `XmNlocationId` resource.
- `XvhHELP_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 `XmNstringData` resource.
- `XvhHELP_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 `XmNstringData` resource.
- `XvhHELP_TYPE_FILE` for displaying a text file. The name of the file to be displayed is specified by setting the `XmNhelpFile` resource.
- `XvhHELP_TYPE_MAN` for displaying a manual reference page ("man page") in a help dialog. The man page to be displayed is specified by setting the `XmNmanPage` resource.

  These values are defined in the `Xvh.h` file.

See Also

- Chapter 5 includes information on creating new dialogs and maintaining a dialog cache.

To display a help topic

1. Create a help dialog or retrieve one from your dialog cache.
2. Set the following resources for the help dialog:

`XmNhelpType`	Set to `XvhHELP_TYPE_TOPIC`.
`XmNhelpVolume`	Set to the volume name for your application.
`XmNlocationId`	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], XmNhelpType,   XvhHELP_TYPE_TOPIC); ac++;
XtSetArg (al[ac], XmNhelpVolume, "MyVolume");         ac++;
XtSetArg (al[ac], XmNlocationId, "getting-started");  ac++;
XtSetArg (al[ac], XmNcolumns,    40);                 ac++;
XtSetArg (al[ac], XmNrows,       12);                 ac++;
XtSetValues (helpDialog, al, ac);
XtManageChild (helpDialog);
```

If the help volume `MyVolume` is not registered, then a complete path to the `MyVolume.hv` file is required for the value of `XmNhelpVolume`.

To display a string of text

1. Create a quick help dialog or retrieve one from your dialog cache.

   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:

`XmNhelpType`	Set to `XvhHELP_TYPE_DYNAMIC_STRING` (if you want wordwrap enabled) or `XvhHELP_TYPE_STRING` (if you want the line breaks within the string to be maintained).
`XmNstring-Data`	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], XmNhelpType,   XvhHELP_TYPE_DYNAMIC_STRING); ac++;
XtSetArg (al[ac], XmNstringData, (char *)descriptionString);   ac++;
XtSetValues (quickHelpDialog, al, ac);
XtManageChild (quickHelpDialog);
```

If the string is no longer needed within the application, the memory

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], XmNhelpType, XvhHELP_TYPE_MAN);  ac++;
XtSetArg (al[ac], XmNmanPage,  "grep");            ac++;
XtSetArg (al[ac], XmNcolumns,  80);                ac++;
XtSetArg (al[ac], XmNrows,     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          XmNhelpCallback,
     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.

The 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 (referred to above as *HelpRequestCB*()). Within that 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 `Help.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, we'll 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 `Help.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, XmNhelpCallback,
               HelpRequestCB, HELP_MainWindow);
XtAddCallback (nameField, XmNhelpCallback,
               HelpRequestCB, HELP_NameField);
XtAddCallback (addressField, XmNhelpCallback,
               HelpRequestCB, HELP_AddressField);
XtAddCallback (phoneField, XmNhelpCallback,
               HelpRequestCB, HELP_PhoneField);
```

Within the `HelpRequestCB()` function, the *clientData* parameter is used to dispatch the help requests (via 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], XmNhelpType,   XvhHELP_TYPE_TOPIC); ac++;
  XtSetArg (al[ac], XmNhelpVolume, HELP_volumeName);    ac++;
```

```
    XtSetArg (al[ac], XmNhelpType,   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 above 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 *OSF/Motif Style Guide* 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 *OSF/Motif Style Guide*.

See Also

- "To create a home topic" in Chapter 2 describes how authors create the home topic for a help volume.
- "To create a meta information section" in Chapter 2 describes how authors create a copyright topic.
- Chapter 9 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 pick an item on the screen. When an item is picked, the application is expected to display a description of the item.

The HP Help System provides a convenience function, `XvhReturnSelectedWidgetId()`, that 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 `XvhReturnSelectedWidgetId()` returns NULL if the user presses Esc 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 `XvhReturnSelectedWidgetId()` function and lets the user pick a widget. Within that function, invoke the help callback for the selected widget. In the following steps, this function is called `HelpMode()`, 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 `HelpMode()` 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 `HelpModeCB()` function that would be invoked by choosing On Item from the Help menu.

```
Boolean  HelpModeCB()
 {
    /* Declare a variable for the selected widget. */
    Widget  selectedWidget = (Widget)NULL;

    /* Let the user select a widget. */
    selectedWidget = XvhReturnSelectedWidget (topLevelShell);

    while (selectedWidget != (Widget)NULL)
       {
          /* If the selected widget has a help callback, invoke it and return 'success.' */
          if ((XtHasCallbacks (selectedWidget, XmNhelpCallback)
                              == XtCallbackHasSome))
             {
                itemHelpFlag = True;
                XtCallCallbacks (selectedWidget, XmNhelpCallback,
                                 (XtPointer)NULL);
                return (True);
             }
          /* Otherwise, try the widget's parent. */
          else
             selectedWidget = XtParent (selectedWidget);
       }
    /* No help callback was found, return 'failure.' */
    return (False);
 }
```

Using the Topic Access Functions

The HP Help System programming toolkit provides three functions for retrieving the text within help topics. These functions are intended for use in character-based applications running on a terminal or in a terminal emulator window. It's up to the application to control the user interface to the help information.

Here are descriptions of the functions:

- `XvhGetTopicData()` retrieves help text from a help volume, based on the volume file location and an ID. The topic's text is returned in a list of lines, formatted to a given column width. Information about the hyperlinks that occur within the topic is also returned.
- `XvhProcessLinkData()` processes a particular hyperlink, returning the destination volume and ID (which can, in turn, be processed by `XvhGetTopicData()`).
- `XvhFreeTopicData()` frees the data returned by `XvhGetTopicData()`.

Graphics and special characters are ignored by the topic access functions.

See Also

- Chapter 14 includes a more detailed description of each function.

Like most 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 HP Help System. However, there are three types of hyperlinks that your application is responsible for handling:

- **Jump-new-view hyperlinks**—Your application must create the new help dialog (or get one from your cache) to honor the author's request for a "new view."
- **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 than 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.

When Dialogs Are Dismissed

When the user closes a help dialog, your application needs to know so it can return the dialog to its cache, or destroy it. The general help dialog supports a "help closed" callback. To detect when a general help dialog is dismissed, add a callback to its OK 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" in Chapter 3 describes the types of links supported by the HP 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, XmNhyperLinkCallback,
                  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 `<Xvh/Xvh.h>`):

```
typedef struct
  {
     int      reason;
     XEvent  *event;
     char    *locationId;
     char    *helpVolume;
     char    *specification;
     int      hyperType;
  } XvhHelpDialogCallbackStruct;
```

The `hyperType` element indicates which type of link was executed. Its possible values are `XvhLINK_JUMP_NEW`, `XvhLINK_MAN`, and `XvhLINK_APP_DEFINED`.

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;
  {
     XvhHelpDialogCallbackStruct *hyperData =
        (XvhHelpDialogCallbackStruct *) callData;

     switch ((int)hyperData->hyperType)
        {
           case XvhLINK_JUMP_NEW:
             /* Handles "jump new view" hyperlinks. */
             break;
```

```
            case XvhLINK_MAN_PAGE:
              /* Handles "man page" hyperlinks. */
              break;

            case XvhLINK_APP_DEFINED:
              /* Handles "application-defined" 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, XmNcloseCallback,
               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 (XvhQuickDialogGetChild (helpDialog,
                                       XvhDIALOG_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.

A sample *HelpCloseCB* function is included in "To return a dialog to your cache" in Chapter 5.

Using the Application-Defined Button

The quick help dialog's application-defined 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.
- *Or,* 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-defined 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 = XvhQuickDialogGetChild (quickHelpDialog,
                                     XvhDIALOG_MORE_BUTTON);
XtAddCallback (moreButton, XmNactivateCallback,
               MoreHelpCB, NULL);

XtManageChild (moreButton);
```

See Also

- "To create a quick help dialog" in Chapter 5

8

Preparing Your Product

When it comes time to prepare your final product, you must be sure that all your help files are created and installed properly.

Where to Install Your Help Files

There are no restrictions regarding where your run-time help files can be installed. However, these conventions are recommended:

- Application-specific help is usually installed with the rest of the application's files. For example, if your application is installed in a directory named `/opt/myapp/` then your run-time help files should be installed in a help subdirectory in the same area: `/opt/myapp/help/`.
- If you've written a stand-alone help volume, your run-time help files should go in their own subdirectory named `/etc/vhelp/help/`*product*`/`, where *product* is a meaningful directory name for your product.

  Within your *product*`/` subdirectory, an additional subdirectory is recommended for the run-time help files for each help volume (*volume*`/`). For example, help volumes for the HP-UX operating system are installed in `/usr/lib/vhelp/help/hpux/`, which has a separate subdirectory for each HP-UX help volume.

You should also provide a *language*`/` subdirectory to accommodate help in multiple languages (where *language* matches the user's LANG environment variable). For example, the German version of the HP VUE online help is stored in `/usr/vue/help/de_DE.iso88591/`. The default version is stored in `/usr/vue/help/C`.

Registration

An additional important step in installing your help files is **registration**. The registration process enables two important features of the HP 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 filesystem.
- **Help family browsing**—If you also register a "product family" that contains one or more help volumes, then your help volumes will be browsable using the HP VUE Help Manager.

How a Help Volume is Found

When you specify a help volume, you can use a complete path to the *volume*`.hv` file, or you can provide just the base name of the volume. If you specify a complete path, the HP Help System looks only for the exact directory and file name you provide.

However, if you provide just the base name (*volume*), the help system searches several directories for the volume. The search ends when the first matching *volume*`.hv` file is found. The value of the user's LANG environment variable is also used to locate help in the proper language (if it's available).

Personal Help Volumes

HP Help first looks in the user's home directory, searching these subdirectories (in the following order):

```
vhelp/volumes/language/
vhelp/volumes/
vhelp/
vhelp/volumes/C/
```

Individual users can override this list of directories by setting the XVHHELPUSERSEARCHPATH environment variable.

System-Wide Help Volumes

If no match is found after searching for personal help volumes, the system-wide locations are searched. They are:

```
/etc/vhelp/volumes/language/
/etc/vhelp/volumes/
/etc/vhelp/volumes/C/
```

The default system search paths can be overridden by setting the XVHHELPSYSTEMSEARCHPATH environment variable. Generally, this environment variable is set for all users by the system administrator.

For systems running HP VUE (version 3.0 or later), the `/etc/vue/config/Xsession` script is recommended for setting the XVHHELPSYSTEMSEARCHPATH variable for all users. The `Xsession` script is executed for each user when they log into HP VUE.

Specifying Paths

Both environment variables use a colon-separated list of directories. That is, you separate multiple directories with the colon character (`:`). In addition, the paths use a special variable syntax in which a percent character (`%`) followed by a letter indicates where values should be inserted before the search takes place.

For example, the default system-wide search paths are specified using this syntax:

```
/etc/vhelp/%T/%L/%H:
/etc/vhelp/%T/%H:
/etc/vhelp/%T/%L/%H.hv:
/etc/vhelp/%T/%H.hv:
/etc/vhelp/%T/C/%H:
/etc/vhelp/%T/C/%H.hv
```

Where `%T` is replaced with the type of file being searched for (which is `volumes`, in most cases), `%L` is replaced by the value of the user's LANG environment variable, and `%H` is the name of the file being searched for. Both `%H` and `%H.hv` are specified for each directory, which means the `.hv` extension is optional when specifying a help volume.

Creating Symbolic Links

To "register" a help volume, you create a symbolic link in one of the directories that the help system searches. The link should have the same name as your *volume*`.hv` file and it should use a *relative* path to the `.hv` file.

Creating symbolic links during installation is recommended rather than putting all your run-time help files directly into one of the help directories. (You create symbolic links with the `ln` command. Refer to your operating system's online help or documentation to learn more about symbolic links.)

To change the help search paths

- For a personal configuration, set the XVHHELPUSERSEARCHPATH environment variable to override the default personal search path. This is useful during project development to access a help volume that you don't want others to access.
- For system-wide configuration, set the XVHHELPSYSTEMSEARCHPATH environment variable to override the system search path. Changing this variable is useful for accessing help installed on another system in your network. Typically, this task is performed by the system administrator for all users. On systems running HP VUE 3.0 or later, edit the `/etc/vue/config/Xsession` script to change this variable.

Example

This example shows how the XVHHELPSYSTEMSEARCHPATH environment variable is used within HP VUE to access help installed on another computer on the network. (These tasks require superuser permission to mount a new filesystem and edit the `Xsession` script.)

To make the remote files accessible, "mount" the remote system's `/usr/` disk volume on your local system in the directory

/usr/*hostname*/usr/. (Refer to the operating system documentation or online help to learn how to "mount" a network file system.)

Once the remote files are physically available, you must tell the HP Help System where to look for them. You do this in HP VUE by editing the /etc/vue/config/Xsession file. Search for the section of the file that defines XVHHELPSYSTEMSEARCHPATH, then uncomment and edit the lines so that they look like this:

```
XVHHELPSYSTEMSEARCHPATH=\
/etc/vhelp/%T/%L/%H:\
/etc/vhelp/%T/%H:\
/etc/vhelp/%T/%L/%H.hv:\
/etc/vhelp/%T/%H.hv:\
/etc/vhelp/%T/C/%H:\
/etc/vhelp/%T/C/%H.hv:\
/net/hostname/etc/vhelp/%T/%L/%H:\
/net/hostname/etc/vhelp/%T/%H:\
/net/hostname/etc/vhelp/%T/%L/%H.hv:\
/net/hostname/etc/vhelp/%T/%H.hv:\
/net/hostname/etc/vhelp/%T/C/%H:\
/net/hostname/etc/vhelp/%T/C/%H.hv
```

Where *hostname* is the name of the directory you created for the remote system.

When you're done editing, save the file, then log out and back in. To update your Help Manager, run this command:

```
/usr/vue/bin/helpgen
```

Gathering Run-Time Help Files

The run-time help files generated by the HelpTag software include the following:

volume.hv	The master help volume file accessed by the HP Help System. Information stored in this file is used to access the actual help topics stored in the .ht files.
volume.hvk	The keyword index file for the volume.
volumeNN.ht	The *help topic* files, where *NN* numbers the files sequentially (00, 01, 02 ...). If you didn't use any <chapter> elements within your help volume, you'll have only a single topic file (*volume*00.ht). Otherwise, you'll have an additional .ht file for each chapter.

When you move the run-time help files to a new location, they must all stay together in the same directory. More importantly, when you move graphics files, they must remain in the same relative location, with respect to the other run-time files.

Caution

Never rename a run-time help file or graphics file after running HelpTag. The information stored in the *volume*.`hv` file depends on the original names.

If you rename your *volume*.`htg` file or any of your graphics files, be sure to rerun HelpTag.

To gather the run-time help files for a volume

- From the directory where you ran HelpTag, copy these files into the installation directory:

  volume.`hv`
  volume.`hvk`
  volume`*.ht`

  Where the basename, *volume*, is the same as the basename of your *volume*.`htg` file.

- If your volume includes any graphics, copy the graphics files too.

Note

Graphics files *must* be installed in the same relative position to the *volume*.`hv` file that they were in when the `helptag` command was run.

For example, if your graphics files are in a subdirectory named `graphics/` one level below your *volume*.`htg` file, then when you install your help files in a different location, the graphics must again be placed in a subdirectory named `graphics/` one level below the *volume*.`hv` file.

You don't need to ship the *volume*.`htg` or any additional files generated by the HelpTag software.

Example

Here's a method for copying the run-time help files for your volume and installing them in another directory. This example assumes that the graphics for this help volume are stored in a `graphics/` subdirectory.

First, change to the directory where you ran HelpTag to create the run-time help files. Then, run this `tar` (*tape archive*) command:

```
tar cvf /tmp/myhelp.tar volume.hv* volume*.ht graphics
```

This creates a file named `/tmp/myhelp.tar`. To install your help files in another directory (*install-directory*), change to that directory:

```
cd install-directory
```

Then, execute this command:

```
tar xvf /tmp/myhelp.tar
```

To test the files you've moved, run Helpview to display your volume. If you are still in the directory where you unpacked the files, you

don't need to specify a path to the volume (because Helpview always looks first in the current directory for the requested volume):

```
helpview -helpVolume volume &
```

Registering Your Online Help

Registering your online help is important because it 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 product family with one or more help volumes, you make your help available for general browsing. This allows access to application-specific help without using the application. Or, if you are writing stand-alone help, this is the only way for users to get to your help.

Registering Help Volumes

After the run-time files for a help volume have been installed, the volume is registered by creating a symbolic link to the volume's *volume*`.hv` file. The link is created in one of the directories that the help system searches for volumes. For most help volumes, the appropriate place for the link is `/etc/vhelp/volumes/`*language*`/`, where *language* is `C` for the default computer language (which is usually English).

Registering a Product Family

If you are also registering a product family, you create and install a help family file (*product*`.hf`) with the rest of the product's help files. You register the family file by creating a symbolic link to the *product*`.hf` file. For most products, the appropriate place for the link is `/etc/vhelp/families/`*language*`/`.

Family files are read by the `helpgen` program (which is part of HP VUE), which uses them to create a special help volume that lists the families (and the volumes within each family) installed on the system.

To register a help volume

- In one of the standard help directories, put a relative symbolic link that points to the directory containing your *volume*`.hv` file.

  The standard help directory most often used as the location of the relative symbolic link is `/etc/vhelp/volumes/`*language*`/`, where *language* is `C` for the default computer language (which is usually English).

Create the symbolic link after all the help files and graphics have been installed.

Use the `ln` command to create the relative symbolic link. If the link you create is not relative, the help volume will not be accessible from a remote system. A relative symbolic link uses `../` to designate the directory at the next higher level.

Example

If the *volume*`.hv` file is `/opt/myapp/help/C/myapp.hv`, and the help volume is being registered in `/etc/vhelp/volumes/C/`, the following relative symbolic link would be used:

```
/etc/vhelp/volumes/C/myapp.hv
   ->   ../../../../opt/myapp/help/C/myapp.hv
```

There are four occurrences of `../` because `/etc/vhelp/volumes/C/` is four levels deep. Remember that each `../` refers to moving *up* one level in the directory hierarchy.

Why *Relative* Links?

The `/opt/` location can be at different levels on different systems, so an absolute symbolic link will not work on all systems. In contrast, the location of the *volume*`.hv` file relative to the directory in which it is registered is typically the same on all systems.

For example, suppose one system has the `opt/` disk at `/net/`*hostname*`/opt/` and a second system has the `opt/` disk at `/opt/`. A relative symbolic link that begins with `../../../../opt/myapp/help/C/` will correctly point to `/net/`*hostname*`/opt/myapp/help/C/` on the first system and `/opt/myapp/help/C/` on the second system.

To create and register 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 characterset used by the *family title* and *family abstract* strings. 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 *relative* to the `/etc/vue/help/`*language*`/Browser/` directory. (See the example below.) 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.

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

3. When you prepare your final product, you should install your *family*`.hf` file with the rest of your help files, then create a relative symbolic link in `/etc/vhelp/families/`*language*`/` that points to the *family*`.hf` file. (See the example below.)

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

Example

Here's a family file for the online version of this manual. Notice that comments at the top of the file identify the file:

```
!##############################################
!#                                            #
!#   HP Help Developer's Kit Product Family   #
!#                                            #
!#                 Version 3.0                #
!#                                            #
!##############################################


*.charSet:      iso8859-1
*.title:        HP Help System, Version 3.0
*.bitmap:       ../../../../usr/vhelp/help/C/HelpKit/HPHelpKit.pm
*.abstract:     Online reference for authors and programmers using the HP Help System.

*.volumes:  HPHelpKit.hv  helpdemo.hv
```

The help family file actually included with the HP VUE software may not exactly match this example. (If HP VUE 3.0 is installed on your system, see the file `/etc/vhelp/families/C/HPHelpKit.hf`.)

To update the "browser" help volume

- Your installation process should execute this command after installing all your help files and creating the links for registration:

  `/usr/vue/bin/helpgen`

The `helpgen` program creates a new volume stored in `/usr/vue/help/`*language*`/Browser/` that contains a two-level hierarchy. The first level lists each of the help families; the second level lists the volumes within each family. (The `helpgen` program is part of HP VUE.)

The volume titles that appear within a product family are jump-new-view links to the home topic of each volume.

To display the browser volume, execute this command:

```
helpview -helpVolume browser &
```

Note

If you run `helpgen` while the browser volume is displayed in a help window, you should close the window, then rerun `helpview`.

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 just as a user will.

For Authors

□ *A final version of the run-time help files was created.*

Here are the recommended commands for creating the run-time files:

```
helptag -clean volume
helptag volume nomemo onerror=stop
```

The `-clean` option removes files from any previous `helptag` 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.

□ *All hyperlinks have been tested.* Each hyperlink displays the proper topic or performs the correct action.

□ *All graphics are acceptable.* The graphics have been tested on various color, grayscale, and monochrome displays.

For Product Integrators

□ *The proper run-time files are installed.* They include:

volume`.hv`
volume`.hvk`
volume`00.ht`, *volume*`01.ht`, ... , *volume*`NN.ht`

The number of `.ht` files depends on how many `<chapter>`s the author created.

□ *All graphics are installed in the proper locations.* Each graphics file must be installed in the same relative position to the `.hv` file that it was in relative to the `.htg` file when the HelpTag software was run.

□ *A relative symbolic link was created to register each help volume.* The link is in the `/etc/vhelp/volumes/`*language*`/` directory and points to the actual *volume*`.hv` files. For example, this command registers the `myapp` volume:

```
ln -s ../../../../opt/myapp/help/C/myapp.hv /etc/vhelp/volumes/C/myapp.hv
```

Making the path relative is important to making the registration work properly when help is shared on a network.

□ *A product family file is installed and registered.* The family file is installed with the other help files. It is registered by creating a symbolic link in the `/etc/vhelp/families/`*language*`/` directory. For example, this command registers the `myapp` family file:

```
ln -s ../../../../opt/myapp/help/C/myapp.hf /etc/vhelp/families/C/myapp.hf
```

For Programmers

- *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.

- *The application sets the desired values for the following optional resources:*

  App-class`*XvhHelpDialogWidget*onHelpDialog*rows:` *rows*
  App-class`*XvhHelpDialogWidget*onHelpDialog*columns:` *columns*

  App-class`*XvhHelpDialogWidget*definitionBox*rows:` *rows*
  App-class`*XvhHelpDialogWidget*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:

- *A help font scheme has been appended to the application's app-defaults file.* Each font scheme is a set of resources. Sample font schemes are provided in the `/usr/vhelp/examples/fontschemes/` directory.

9

Providing Help On Help

"Help on help" tells users how to use the HP Help System. Specifically, it describes such tasks as using hyperlinks, using the keyword index, and printing help topics. Normally, help on help is supplied as an individual help volume named "Help4Help."

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 an existing help on help volume.* For example, on HP workstations running HP-UX 9.0 or later, the standard Help4Help volume is installed with the X Window System.
- *Or, supply your own help on help volume.* The HelpTag source files for the Help4Help volume are provided in the `/usr/vhelp/help/C/Help4Help/` directory.

For Stand-Alone Help

If you are writing stand-alone 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 HP 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:

```
Helpview*helpOnHelpVolume:  Help4Help
```

The run-time help files for the Help4Help volume are installed in `/usr/vue/help/`*language*`/Help4Help/`. A symbolic link to the `Help4Help.hv` file is created in the `/etc/vhelp/volumes/`*language*`/` directory to register the volume. (HP VUE does not have to be running to access this help volume.)

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.
- Add support for getting help on quick help dialogs.

To set the 'helpOnHelpVolume' resource

- Add a line to your application's app-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 general help dialog you create. (Quick help dialogs do not support this resource.)

Examples

Here's the line from Helpview's app-defaults file that specifies the help on help volume:

```
Helpview*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], XmNhelpOnHelpVolume, "Help4Help");  ac++;
helpDialog = XvhCreateHelpDialog (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 within your `HelpRequestCB()` function to display help on help. Specifically:
   - Create a quick help dialog (or retrieve one from your cache).
   - 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 "To display help on help".

To provide help on help for a quick help dialog

1. After creating the quick help dialog, do the following:
   - Manage the dialog's Help button.
   - Add a help callback to the dialog.
2. Add support in your `HelpRequestCB()` function to handle the case when a user requests help in a quick help dialog. Specifically:
   - Create a quick help dialog (or retrieve one from your cache).
   - 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 program segment creates a quick help dialog, manages its Help button, and adds a help callback to the dialog:

```
/* Create a quick help dialog. */
ac = 0;
XtSetArg (al[ac], XmNtitle, "My Application - Help"); ac++;
helpDialog = XvhCreateHelpDialog (parent, "helpDialog", al, ac);

/* Manage the dialog's Help button. */
XtManageChild (XvhQuickDialogGetChild (quickHelpDialog,
                                       XvhDIALOG_HELP_BUTTON));

/* Add a help callback to enable the F1 key and the Help button. */
XtAddCallback (quickHelpDialog, XmNhelpCallback,
               HelpRequestCB, USING_HELP);
```

To see how the `HelpRequestCB()` function handles the `USING_HELP` case, see the example in "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,   XvhHELP_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" in Chapter 6.

See Also

- "To create a quick help dialog" in Chapter 5
- "To retrieve a dialog from your cache" in Chapter 5 (includes a sample `FetchHelpDialog()` function)
- "To display a help topic" in Chapter 6

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, then make the necessary changes. All the source files used to write the Help4Help volume are provided in the `/usr/vhelp/help/C/Help4Help/` directory.

It is important that you name your help on help volume something other than Help4Help, to prevent installation conflicts. Consider picking a name that is specific to your product. For example, if your application's help volume is "Newapp," perhaps 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.
`keyword-index`	Displays a topic describing how to use the Keyword Search dialog. This topic is displayed when you choose Help or press F1 within the Keyword Search dialog.

To copy the Help4Help source files

1. Copy the entire `/usr/vhelp/help/C/Help4Help/` directory to a new working directory (*new-dir*) using a command like this:

   ```
   cp -r /usr/vhelp/help/C/Help4Help 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 source for the Help4Help volume includes these files:

```
MetaInfo        Printing      Commands
HomeTopic       TopicMap      Config
Hyperlinks      History       KeywordIndex
```

Also included is a `build/` directory, where you run HelpTag to create the run-time help files. Graphics are stored in the `build/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/vhelp/help/C/Help4Help /home/dex/projects/NewHelp4Help
chmod -R u+w /home/dex/projects/NewHelp4Help
```

To build a new version of the run-time help files, first ensure that the directory `/usr/vhelp/bin/` is in your search path. Then, change to the new directory, rename the `Help4Help.htg` file, and run HelpTag:

```
cd /home/dex/projects/NewHelp4Help
mv Help4Help.htg NewH4H.htg
helptag NewH4H
```

When the HelpTag software is done, you can display the new help on help volume using this command:

```
helpview -helpVolume NewH4H &
```

10

Native Language Support

If your product is intended for an international audience, then providing online help in the users' native language is important. The HP Help System supports the authoring and displaying of online help in virtually any language. Several factors, which are explained below, 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. Some of these languages require two or four bytes to represent each character uniquely.

The Helpview application supports multi-byte character sets.

The HelpTag Software

When you process a help volume to create run-time help files, the HelpTag software must be told what character set you used to author your files. The character set information is used to determine the proper fonts for displaying help topics. If you do not specify a character set, HelpTag assumes the default, which is ISO 8859-1.

Note

When writing HelpTag files, you may use multi-byte 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.

The Xvh 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 `Xvh.cat`.

The `Xvh.cat` file has been translated into several languages and these translations are included with the HP Help System Developer's Kit. Look in the `/usr/vhelp/nls/` directory. If the language you need is not supplied, you'll have to translate the message catalog (`/usr/vhelp/nls/C/Xvh.msg`) and then use the `gencat` command to create the needed run-time message catalog file.

Font Schemes

One of the primary functions of the HelpTag software is to convert your marked-up files into run-time formats that the HP Help System understands. Text is formatted by specifying particular attributes such as type family, size, slant, and weight. Font schemes are used to "map" combinations of attributes to actual font specifications.

Formatting of some languages also requires a *formatting table*. This table specifies rules for word wrap and other processing. If you are preparing a Japanese help volume, be sure the sample formatting table in `/usr/vhelp/nls/ja_JP.SJIS/fmt_tbl.cat` is installed in the `/usr/lib/nls/japanese/` directory. (If you are using the Japanese EUC character set, use the `euc/` subdirectory.)

The LANG Environment Variable

The user's LANG environment variable is important for these 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

- "How a Help Volume is Found" in Chapter 8
- Also refer to the NLS documentation for your computer's operating system or programmer's kit.

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/vhelp/helptag/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 option? Here are the most common character set names:

```
iso8859-1
hp-roman8
hp-japanese15
hp-korean15
```

If you author in another character set, you may have to translate the `Xvh.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 multi-byte 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 HP Help System into an application, have you properly set the "locale" using the `setlocale()` function?

See Also

- "How a Help Volume is Found" in Chapter 8 describes how search paths (which use the value of LANG) are used to locate help files.
- The man page for the `setlocale()` function describes when and how to use the `setlocale()` function.
- The man page for the `gencat` command describes how to create a message catalog file. You'll use this command if you translate the `Xvh.msg` file.

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 structural information you've entered is formatted into run-time help files which include text attributes.

A **font scheme** maps text attributes to actual font specifications. For example, if a help topic has some text formatted as "sans serif, bold, italic," the font scheme dictates which 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. For example, if you use Helpview to display a help volume, the font scheme is included in Helpview's app-defaults file (`/usr/lib/X11/app-defaults/Helpview`). For application help, the resources may be added to the application's app-defaults file.

Each resource within a font scheme has this general form:

**pitch.size.slant.weight.style.char-set*: *font*

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 `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`.

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 `iso8859-1`. 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:

```
*.8.*.*.*.symbol:   -adobe-symbol-medium-r-normal--8-*-*-*-p-*-adobe-fontspecific
*.10.*.*.*.symbol:  -adobe-symbol-medium-r-normal--10-*-*-*-p-*-adobe-fontspecific
*.12.*.*.*.symbol:  -adobe-symbol-medium-r-normal--12-*-*-*-p-*-adobe-fontspecific
*.14.*.*.*.symbol:  -adobe-symbol-medium-r-normal--14-*-*-*-p-*-adobe-fontspecific
```

The `/usr/vhelp/examples/fontschemes/` directory contains some sample font schemes. The naming convention of the sample files uses the actual pixel height of the font used to display text formatted as "10-point" text. For example, the file `help014.fns` contains a font scheme where all 10-point text is displayed using a 14-pixel font.

To choose a font scheme

- Edit the app-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 app-defaults file into your home directory before editing it.

Example: Fonts for Helpview

To use a larger size font in your Helpview windows, first change to your home directory:

```
cd
```

Then copy the Helpview app-defaults file and make it writable:

```
cp /usr/lib/X11/app-defaults/Helpview
chmod u+w Helpview
```

Edit the `Helpview` file to replace the existing font scheme with the largest scheme (`help017.fns`). Search for the block of comments titled "Font Scheme." Delete *all* the lines that follow those comments. Then, insert the contents of this file:

```
/usr/vhelp/examples/fontschemes/help017.fns
```

Save your new `Helpview` file. Any new instance of Helpview that you start will now use the new font scheme. Try it by displaying the *HP Help System Developer's Guide*:

```
helpview -helpVolume HPHelpKit &
```

11

HelpTag Markup Reference

All the HelpTag markup **elements** (and their associated tags) are described 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>`

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>`
`<term>` (shorthand: `++`*text*`++`)
`<user>` (user input)
`<var>` (variable) (shorthand: `%%`*text*`%%`)
`<newline>`
`<p>` (paragraph)
`<quote>`
`&` ... `;` (see `<!entity>`)

Important information:

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

Lists:

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

Graphics:

`<figure>`
`<graphic>`

Glossary and keyword 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>`

Override meaning of HelpTag markup:

`<esc>` (escape from markup recognition)
`<vex>` (verbatim example)

<!-- ... -->

Comment

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=DeltaGee>
Before and After Processing
<\figure>  -->
```

See Also

"<memo>"

<abbrev>

Abbreviated title

Indicates an alternate, typically shorter, heading for a topic that has a long title. The abbreviated title is used within the HP Help System's dialogs whenever the title of a topic is displayed in a list (such as in the Topic Hierarchy and the History dialog).

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 text display area, but you do want a title to appear in the Topic Hierarchy. The following markup shows how this can be done:

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

See Also

"<chapter>"
"<rsect>"

<abstract>

Abstract

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 the 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.

Example

This markup briefly describes the contents of a help volume:

```
<abstract>
Online help for the HP VUE File Manager, Version 3.0.
<\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>"

<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.

Examples

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.

Syntax

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

The default heading for the caution 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/vhelp/helptag/icons/` directory.

Example

Here is a caution message:

```
<caution>
Do not press the DELETE key at this time.
<\caution>
```

For an example that shows how to use the `<head>` element to specify a non-default heading, refer to "<note>".

See Also

"<note>" includes an example of changing a heading.
"<warning>"
"<figure>"
"<head>"

<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>`.

Examples

Here are two markups that begin a new topic:

```
<chapter>A Manual of Style

<chapter id=writer>The Careful Writer
```

See Also

"<abbrev>"
"<link>"
"<rsect>"
"<s1> ... <s9>"
"<xref>"

<computer>

Computer literal

Displays text that represents computer input or output.

Syntax

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

Or:

‘‘*text*’’

The shorthand form uses two left apostrophes or grave accents (‘‘) 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, the following markup:

```
‘‘void  DisplayTopic (%%topic%%);’’
```

produces:

`void  DisplayTopic (`*topic*`);`

See Also

“<ex>”
“<user>”
“<var>”

<copyright>

Copyright notice

Used to enter text for the copyright notice.

Syntax

```
<metainfo>
  <title>Title (always before copyright)
  <copyright>
  &copy; Copyright notice here ...
```

This element is required within the `<metainfo>` section and must follow the `<title>` element.

The end tag is not required.

The predefined entity `&copy;` produces the copyright symbol (©).

Example

```
<metainfo>

<title>HP Help System Developer's Guide

<copyright>
&copy; Copyright 1992 Hewlett-Packard Company.
All rights reserved.
```

See Also

"<metainfo>"
"<title>"

<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>ex libris
From the books. Used before the owner's name on bookplates.

<dterm>key word
A word exemplifying the meaning or value of a letter or symbol.
```

See Also

"<glossary>"
"<term>"

<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.

Examples

Either of the following two markups:

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

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

produce:

A thousand times *no*.

See Also

"<book>"
"<quote>"
"<term>"
"<var>"

<!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 (<), the 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 ending 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 "HP Precision Architecture" to be displayed wherever the `&hppa;` entity reference appears in the marked-up files.

```
<!entity hppa "HP Precision Architecture">
```

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" in Chapter 2
"<figure>"
"<xref>"
Chapter 12

<esc>

Escape

Causes text to be passed directly to the run-time help files without being interpreted by HelpTag.

Syntax

`<esc>`*text*`<\esc>`

Or:

`<esc|`*text*`|`

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). The `<\esc>` end tag is required.

If the short form is used, the text cannot contain the vertical bar character (`|`).

Example

The following markup:

```
<esc|characters !! or \|
```

produces:

characters !! or \

See Also

"General Markup Guidelines" in Chapter 2
"<vex>"

<ex>

Computer example

Shows computer text without changing the spacing or line breaks.

Syntax

```
<ex [nonumber | number] [smaller | smallest]>
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` or `smallest`	Displays the example using smaller fonts. This makes long lines fit within a narrower width.

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:

```
!!
--
++
"
```

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

Example

The following markup:

```
<ex>
Examples are printed
in computer font.
<\ex>
```

produces:

```
Examples are printed
in computer font.
```

See Also

"<computer>"
"<user>"
"<vex>"

<figure>

Figure

Inserts a graphical image.

Syntax

```
<figure [nonumber] entity=entity id=id number=n>
caption string
<\figure>
```

`nonumber`	Suppresses the word "Figure" and the automatically generated figure number.
`entity=`*name*	Specifies a file entity that identifies the file which contains the graphic image to be inserted.
`id=`*name*	Optional. Defines an ID name that can be used in cross-references to this figure.
`number=`*n*	Optional. Used to override the automatically generated figure number.
`ghyperlink="`*id*`"`	Optional. Specifies that the graphic portion of the figure be a hyperlink. 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`.
`gdescription="`*text*`"`	Optional. Provides a description of the hyperlink. This description is used by the topic access functions.

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 "mappic.xwd">
   :
<figure entity=MapFigure>
Caption for Figure
<\figure>
```

The following markup inserts a figure that is numbered but does not have a caption. The figure is referred to later in the text.

```
<!entity MyPicture FILE "mappic.xwd">
   :
<figure id=Layout entity=MapFigure>
<\figure>
   :
<xref Layout> shows the layout of ...
```

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>"
"<graphic>"
"<link>"
"<xref>"

<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 *filename*.`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>"
"<dterm>"

<graphic>

Inline graphic

Used for inserting a graphical element within a line of text.

Syntax

```
<graphic entity=name>
```

Where:

name	An entity name which 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.

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 for very small images.
```

The following markup defines the inline graphic as a hyperlink to a topic whose ID is `mini-icon-topic`:

```
The <link mini-icon-topic><graphic entity=mini-icon><\link> icon
is used for very small things.
```

See Also

"<!entity>"
"<figure>"
"<link>"
"<p>"
"To include a special character" in Chapter 3

<head>

Heading

Indicates the title for elements that normally do not have a title (such as image, lablist, list, and otherfront) or have a default title (such as note, caution and warning).

Syntax

`<`*element*`><head>`*title text*

A heading starts with the first non-blank 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
```

The following markup overrides the default "Note" heading. The ampersand (&) indicates that the heading continues on the following line.

```
<note><head>Tips&
& Traps
Take special note of this.
<\note>
```

See Also

"<otherfront>"
"<note>"
"<caution>"
"<warning>"

<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

"A Help Volume at a Glance" in Chapter 2
"<!entity>"
"<hometopic>"
"<metainfo>"

<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>`.

Linking to the Home Topic

The `<hometopic>` element does not support an author-defined ID identifier. The HelpTag software assigns the pre-defined identifier `_hometopic`. To create a hyperlink to the home topic, use `<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" in Chapter 2
"To create a home topic" in Chapter 2
"<metainfo>"

<idx>

Index entry

Defines an entry to appear in the keyword 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.

The keyword index is displayed by choosing Keyword from the Search menu in a general help dialog. (The keyword index is not available in quick help dialogs.) When the index entry is chosen in the Keyword Index dialog, the topics that contain the index entry are listed. Choosing one of the listed topics displays that topic. The Keyword Index dialog remains available for further keyword index access.

Either the `<idx>` start and end tags or the short form can be used.

The `<sort>` element changes the sort order for a keyword 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. The index entries are indented to make the source text easier to read.

```
        <idx|keyboard|
        <idx|disk drive|
        <idx|screen, LCD|

An HP Portable Vectra CS PC has a full
size keyboard, built-in disk drives and
a detachable LCD screen.
```

The following example displays "+" in the keyword index, but it appears where "plus" would appear.

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

<image>

As-is image

Shows text with the same line breaks as are in the source text.

Syntax

```
<image [id=id] [indent]>
text
<\image>
```

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.

All in-line text elements and special characters are recognized.

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.

If the displayed text is too wide to fit within the display area, a horizontal scroll bar automatically appears.

See Also

"<ex>"
"<vex>"
"<p>"

<item>

List item

Identifies an item in a list.

Syntax

```
<list>
 * List item
 * List item
<\list>
```

Or:

```
<list order>
  <item id=name> List item
  <item id=name> List item
  <item id=name> 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 list because the automatically-assigned item numbers are used in the cross-reference text (which HelpTag substitutes for the `<xref>` element).

See Also

"<list>"
"<head>"
"<xref>"

<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]>

[ <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.

Backslashes (\) indicate the start and end of a label; leading and trailing spaces are ignored. 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.
- The start of another labeled item.
- The `<\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.

Labels that are wider than the predefined label area extend into the right column.

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 (\) character. 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
  \pc\ picas
  \pt\ points
  \mm\ millimeters
  \cm\ centimeters
```

```
<\lablist>
```

produces this output:

Unit	Meaning
in	inches
pc	picas
pt	points
mm	millimeters
cm	centimeters

See Also

"<head>"
"<list>"

<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 definition window, the topic which contains the ID *hyperlink*.
`Execute`	Executes the *hyperlink* string as a command.
`Man`	Displays a man page using the *hyperlink* string as the parameter(s) to the `man` command.
`AppDefined`	Sends the *hyperlink* string to the hosting application for special processing.

The *text* between the begin and end tags becomes the "hot spot" that the user will choose to invoke the link.

Capitalization is not significant for the *hyperlink* and *type* values.

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 above.
- 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 the topic with the ID `Welcome`. Notice that capitalization of the ID is not significant.

```
Refer to the <link welcome>Welcome<\link> topic.
```

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 `Welcome` topic is automatically supplied by HelpTag.

```
Refer to the <xref welcome> topic.
```

The following markup defines a hyperlink that is activated when the inline graphic is chosen. A new window is opened to display the referenced help information.

```
The <link clockInfo JumpNewView>
<graphic entity=ClockIcon><\link> icon ...
```

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>.
```

See Also

"<abstract>"
"<figure>"
"<graphic>"
"<hometopic>"
"<idx>"
"<location>"
"<xref>"

<list>

List

Starts a list consisting of items that are optionally marked with bullets or automatically-generated numbers.

Syntax

```
<list [bullet | order | plain] [continue] [loose | tight]>
  * first item
  * second item
    ⋮
<\list>
```

Where:

`bullet`	(Default.) Displays a bullet before each item.
`plain`	Does not put a bullet, number or letter in front of each item.
`order`	Displays a number in front of each item. The numbers are automatically generated and begin with the number one.
`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.

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:

```
<list>
* chocolate
* raspberry
* vanilla
<\list>
```

produces:

- chocolate
- raspberry
- vanilla

The following markup:

```
<list plain tight>
* Word Processing
* Graphics
* Printing
<\list>
```

produces:

Word Processing
Graphics
Printing

See Also

"<item>"
"<lablist>"
"<head>"

<location>

Location

Defines an ID as referring to the location of the `<location>` element. The ID is usually used as a destination for a hyperlink or cross-reference.

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>` and `<figure>`) that has a built-in ID or accommodates an author-defined `id` parameter.

The `<location>` element enables portions of a topic or section to serve as destinations for hyperlinks and cross-references.

Example

The following markup names a location and elsewhere creates a hyperlink to the location.

```
<s1 id=ConfigTopic> Configuration
    ...
<location id=SecondHalfConfigTopic>some text<\location>
    ...
<s1 id=ConfigTopic> Usage
    ...
See <link SecondHalfConfigTopic>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>` is entered. In contrast, a link to the topic's ID ("ConfigTopic" in this case), always goes to the top of the topic.

See Also

"<xref>"
"<link>"

<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.

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|
```

<metainfo>

Meta information

Starts the meta information section, which contains information about the information contained in the help volume. This information includes the volume's title and a copyright notice.

Syntax

```
<helpvolume>
  <metainfo>
    <title> volume title
    <copyright>
    &copy; Copyright XYZ Company 1992...
    <abstract>
    brief description of help volume
      :
    <\metainfo>
<hometopic>...
  :
```

The meta information section is required. The title and copyright subsections are required within the meta information section. Inclusion of the abstract subsection is strongly recommended.

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 1992 Hewlett-Packard Company.
All rights reserved.

<abstract>
Explains how to use the Inventory Tracking Software

<\metainfo>
```

See Also

"<title>"
"<copyright>"
"<abstract>"
"<otherfront>"

<newline>

New line

Starts a new line within a paragraph.

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 file name begins on a new line:

```
Put your files for the manual in the special directory
<newline>/userguide/draftdoc.
```

See Also

"<vex>"
"<ex>"

<note>

Note

Creates a special format that attracts attention to text which 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 `<\note>` end tag is required.

Examples

Here is a note that uses the default heading:

```
<note>
Pay attention; this is important.
<\note>
```

The following markup specifies a different heading:

```
<note><head>Read This
Pay attention; this is important.
<\note>
```

See Also

"<caution>"
"<warning>"
"<head>"

<otherfront>

Other meta information (front matter)

Used for meta information (front matter) that does not fit within one of the predefined categories.

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>"
"<head>"

<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 alter the topic hierarchy and its title does not appear in the Topic Hierarchy list.

The `<\otherhead>` end tag is not needed.

Example

Here is an example in which `<otherhead>` elements identify two subsections within an `<s1>` topic:

```
<s1>Getting Started
text

<otherhead>Copying Files
text

<otherhead>Editing Configuration Files
text
```

See Also

"<head>"
"<procedure>"
"<rsect>"
"<s1> ... <s9>"

<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 from the current left margin.
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.
pos	Optional. Either `left` or `right` to indicate whether the optional graphic is to be left-justified or right-justified.
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. (The *id* value, not the *gid* value, would be used to reference this paragraph's location.)
type	Optional. Specifies the type of hyperlink. The default type is `Jump`. Other type values include `JumpNewView`, `Definition`, `Man`, `Execute`, and `AppDefined`.
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 only if you need to indent a paragraph or wrap the paragraph around a graphic.

A `<\p>` end tag is not required.

Example

Here are two paragraphs, the second of which is indented:

```
Some people do not like to read manuals.

<p indent>This is not always a good idea.
```

See Also

"<head>"
"<procedure>"
"<rsect>"
"<s1> ... <s9>"
"To wrap text around a graphic" in Chapter 3

<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 part of the topic hierarchy and are not listed in the Topic Hierarchy list.

An end tag is not needed.

Example

This paragraph and the "Example" heading were produced using the following markup:

```
<procedure>Example

This paragraph and the "Example" heading were
produced using the following markup:
```

See Also

"<head>"
"<otherhead>"
"<s1> ... <s9>"

<quote>

Quote

Puts text within directional quotation marks.

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>"
"<computer>"
"<term>"
"<user>"
"<var>"

<rsect>

Reference section

Identifies an entry in the reference section.

Syntax

```
<rsect [id=id] >reference section heading
  ⋮
<rsub>reference subsection heading
```

A reference section (`<rsect>`) is used within a topic or section, typically for 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 or `<rsub>`s.

Each `<rsect>` section can have multiple `<rsub>` sections. Each `<rsub>` element must have a heading, but the heading does not appear in the table of contents. A cross-reference to a reference subsection is not allowed.

`<rsect>` headings (but not `<rsub>` headings) appear in the table of contents.

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
```

See Also

"<abbrev>"
"<chapter>"
"<s1> ... <s9>"

<s1> ... <s9>

Subsection (<s1>, <s2>, ... , <s9>*)*

Starts a topic in the hierarchy.

Syntax

<s*n* [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 <s*n*> 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 example illustrates a three-level hierarchy within a topic.

```
<chapter>Running the Processor
topic text...

<s1>Getting Started
To run the program, type in the user
code 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>"
"<head>"
"<rsect>"

<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`	Allows the term to be missing from the glossary.

The shorthand form for `<term>` is double plus signs (`++`) used before and after the 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 both do not appear in the glossary and do not use the `nogloss` parameter.

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 to indicate it is a glossary term and creates a hyperlink to the glossary. Because the `nogloss` parameter is not used, HelpTag ensures that the singular form ("structural element") appears in the glossary.

```
SGML views a document as a hierarchy of
<term "structural element"|structural elements|.
```

See Also

"<glossary>"
"<dterm>"

<title>

Help volume title

Specifies the title of the help volume.

Syntax

```
<metainfo>

<title>help volume title
```

The `<title>` element is a required subelement of the `<metainfo>` (meta information) element. It follows immediately after the `<metainfo>` tag. Because this is the title of the volume, and the title may be displayed by other applications (help managers, for example) that may not be able to format the title, you should avoid anything other than 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>"

<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 dialogue. 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>
```

See Also

"<user>"

<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.

In the shorthand form, the text is delimited with double percent signs (**%%**).

Examples

These markups:

```
INPUT %%filename%%
```

Or:

```
INPUT <var>filename<\var>
```

produce:

INPUT *filename*

See Also

"<ex>"
"<computer>" includes an example of a variable within computer text.
"<user>"

<vex>

Verbatim example

Indicates a verbatim example in which HelpTag elements are not interpreted as elements.

Syntax

```
<vex [smaller | smallest]>
example text
<\vex>
```

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 use shorthand forms of tags that would otherwise be interpreted as markup. The effect is similar to using `<esc>` in text, with output similar to `<ex>`. 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 smaller>
You can use shorthand form characters, such as %%
or !! or ++ without using the <esc> element to allow
them to print.
<\vex>
```

produces:

```
You can use shorthand form characters, such as %%
or !! or ++ without using the <esc> element to allow
them to print.
```

See Also

"<esc>"
"<ex>"
"<image>"

<warning>

Warning

Calls the reader'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 `<\warning>` end tag is required.

Examples

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>"
"<caution>"
"<head>"

<xref>

Cross-reference

Inserts text that identifies another location of the help volume and creates a hyperlink to that location.

Syntax

```
<xref id>
```

Where:

id	The identifier of the topic or location that is being cross-referenced.

Cross-references are translated into section titles or topic, figure, list item or line numbers. The cross-reference text becomes a hyperlink that, when chosen by a user, jumps to the cross-referenced location.

The `id` parameter must be defined in the element to which `<xref>` refers. In both the `xref` parameter and the `id` parameter, the name must be spelled exactly the same. Capitalization, however, is not significant.

The `id` parameter is important because it allows you to cross-reference the element referred to. For example, if you want to refer readers to an appendix for more information, you would write, "Refer to <xref publish> for details." This prints out as "Refer to Appendix X for details", where X is the number of the appendix containing the information.

The `id` parameter can appear with:

```
<chapter>
<s1>, <s2>, ... <s9>
<rsect>
<appendix>
<figure>
<lineno>
<item>
```

A cross-reference to an *id* that contains an underscore (such as "_abstract" or "_hometopic") is not allowed.

Example

Suppose an ID named "analyzer" were defined in the following markup:

```
<s1 id=analyzer>Logic Analyzers
```

Here is markup that contains a cross-reference to the above topic:

```
The HP 16500A logic analysis system, described in
<xref analyzer>, can be configured to a user's needs.
```

After translation by the `helptag` command, the `<xref>` element would be replaced by "Logic Analyzers" as shown here:

```
The HP 16500A logic analysis system, described in
Logic Analyzers, can be configured to a user's needs.
```

The text "Logic Analyzers" would appear as a hyperlink that, when chosen by a user, jumps to the cross-referenced help topic.

See Also

"<!entity>"
"<link>"
"<location>"
"<term>"
"<figure>"
"<chapter>"
"<s1> ... <s9>"
"<otherhead>"
"<rsect>"

12

Summary of Special Character Entities

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.

Note

To use any of the entity names that are marked with an asterisk, you must use the `helpchar.ent` file as explained in "To include a special character" in Chapter 3. The file is in `/usr/vhelp/helptag/`.

Symbol and entity name	Description
Current Date and Time	
4/7/98 `&date;`	Today's date (when HelpTag is run)
12:49 `&time;`	Current time (when HelpTag is run)
Typographical Symbols	
© `&copy;`	Copyright symbol
® `&reg;`	Registered symbol
™ `&tm;`	Trademark symbol
– `&endash;`	En dash (short dash)
— `&emdash;`	Em dash (long dash)
• `&bullet;` *	Bullet
... `&ellipsis;`	Ellipsis (horizontal)
.... `&pellipsis;`	Ellipsis (end-of-sentence)
⋮ `&vellipsis;`	Vertical ellipsis
' `&squote;`	Single quote
" `&dquote;`	Double quote
␣ `&vblank;`	Vertical blank
( ) `&empty;`	Empty (no text)
( ) `&sigspace;`	Significant space
§ `&S;` *	Section
¶ `&P;` *	Paragraph

Basic Math Symbols

−	`&minus;`	Minus
±	`&pm;`	Plus over minus
÷	`&div;`	Divide
×	`&times;`	Multiply
≤	`&leq;`	Less than or equal to
≥	`&geq;`	Greater than or equal to
≠	`&neq;`	Not equal to

Units

AM	`&a.m.;`	AM
PM	`&p.m.;`	PM
°	`&deg;`	Degrees
′	`&minutes;`	Minutes, prime, or feet
″	`&seconds;`	Seconds, double prime, or inches

Currency Symbols

¢	`&cents;`	Cents
£	`&sterling;`	Sterling

Lowercase Greek Letters

α	`&alpha;` *	Lowercase Greek Alpha
β	`&beta;` *	Lowercase Greek Beta
χ	`&chi;` *	Lowercase Greek Chi
δ	`&delta;` *	Lowercase Greek Delta
ε	`&varepsilon;` *	Alternate lowercase Greek Epsilon
φ	`&phi;` *	Lowercase Greek Phi
ϕ	`&varphi;` *	Open lowercase Greek Phi
γ	`&gamma;` *	Lowercase Greek Gamma
η	`&eta;` *	Lowercase Greek Eta
ι	`&iota;` *	Lowercase Greek Iota
κ	`&kappa;` *	Lowercase Greek Kappa
λ	`&lambda;` *	Lowercase Greek Lambda
μ	`&mu;` *	Lowercase Greek Mu
ν	`&nu;` *	Lowercase Greek Nu
π	`&pi;` *	Lowercase Greek Pi
ϖ	`&varpi;` *	Alternate lowercase Greek Pi (or Omega)
θ	`&theta;` *	Lowercase Greek Theta

ϑ	`&vartheta;` *	Open lowercase Greek Theta
ρ	`&rho;` *	Lowercase Greek Rho
σ	`&sigma;` *	Lowercase Greek Sigma
τ	`&tau;` *	Lowercase Greek Tau
υ	`&upsilon;` *	Lowercase Greek Upsilon
ω	`&omega;` *	Lowercase Greek Omega
ξ	`&xi;` *	Lowercase Greek Xi
ψ	`&psi;` *	Lowercase Greek Psi
ζ	`&zeta;` *	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	`&Upsi;` *	Uppercase Greek Psi

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
$\equiv$	`&equiv;` *	Exactly equals
$\neq$	`&not-eq;` *	Not equal to
$\approx$	`&approx;` *	Approximate sign (two wavy lines)
$\neg$	`&neg;` *	Not
$\cap$	`&cap;` *	Cap (Set intersection)
$\cup$	`&cup;` *	Cup (Set union)
$\vee$	`&vee;` *	Vee (Logical OR)

∧	`&wedge;` *	Wedge (Logical AND)
∈	`&in;` *	In
⊂	`&subset;` *	Proper subset
⊆	`&subseteq;` *	Subset
⊃	`&supset;` *	Proper superset
⊇	`&supseteq;` *	Superset
∀	`&forall;` *	For all (Universal symbol)
∃	`&exists;` *	There exists (Existential symbol)
f	`&function;` *	Function symbol (or florin sign)
∠	`&angle;` *	Angle
≅	`&cong;` *	Congruent
∝	`&propto;` *	Proportional to
⊥	`&perp;` *	Perpendicular to
·	`&cdot;` *	Centered dot
⊕	`&oplus;` *	Plus in circle
⊗	`&otimes;` *	Times in circle
⊘	`&oslash;` *	Slash in circle (Empty set)
∂	`&partial;` *	Partial differential delta
∑	`&sum;` *	Summation (Uppercase Greek Sigma)
∏	`&prod;` *	Product (Uppercase Greek Pi)

Arrows

←	`&leftarrow;` *	Left arrow
→	`&rightarrow;` *	Right arrow
↑	`&uparrow;` *	Up arrow
↓	`&downarrow;` *	Down arrow
↔	`&leftrightarrow;` *	Left/right arrow
⇐	`&bigleftarrow;` *	Big left arrow
⇒	`&bigrightarrow;` *	Big right arrow
⇑	`&biguparrow;` *	Big up arrow
⇓	`&bigdownarrow;` *	Big down arrow
⇔	`&bigleftrightarrow;` *	Big left/right arrow

Card Suits

♢	`&diamondsuit;` *	Diamond suit
♡	`&heartsuit;` *	Heart suit

♠	`&spadesuit;` *	Spade suit
♣	`&clubsuit;` *	Club suit

Miscellaneous Symbols

◇	`&diamond;` *	Diamond
ℵ	`&aleph;` *	Hebrew Aleph
∇	`&nabla;` *	Nabla (Inverted uppercase Greek Delta)
√	`&surd;` *	Radical segment, diagonal
℘	`&wp;` *	Weierstraussain symbol
ℜ	`&Re;` *	Fraktur R
ℑ	`&im;` *	Fraktur I

See also

- "General Markup Guidelines" in Chapter 2 explains how to include special HelpTag characters (`&`, `\`, and `<`).

13

Command Summary

The commands summarized here are:

`helptag`	Compiles HelpTag source files into run-time files.
`helpview`	Displays a help volume, text file, or man page.
`helpprint` and `helpprintrst`	Print all or part of a help volume. If the help volume contains multi-byte characters, `helpprintrst` is used.

Processing HelpTag Files ('helptag')

The HelpTag software, invoked with the `helptag` command, compiles your HelpTag source files into run-time help files. You run `helptag` in the directory where your *volume*`.htg` file is located.

Command Syntax

`helptag` [*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 `helptag` command and displays any parser errors that occur. Parser errors are also saved in a file named *volume*`.err`.

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 run-time files.

Any of these options can be either entered on the command line after the *volume* name, or listed in a file named `helptag.opt` located in the current directory:

`onerror` Specifies whether the `helptag` 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 files 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=iso8859-1`.

You can also specify a character set within your help volume by declaring an entity named "LanguageElementDefaultCharset". (This is demonstrated in the `/usr/vhelp/helptag/helplang.ent` file.) See also Chapter 10.

`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` Clears 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" in Chapter 4
- "Gathering Run-Time Help Files" in Chapter 8
- "Viewing a Help Volume" in Chapter 4

Displaying Help Topics ('helpview')

Command Syntax

Here are the various ways to invoke Helpview:

```
helpview  -helpVolume  volume [ -locationId id ] &

helpview -man &

helpview -manPage man &

helpview -file filename &
```

Where:

`-helpVolume` *volume*	Specifies the name of the *volume*`.hv` file you want to view. A path is not required unless the volume is not in the current directory *and* the volume has not been "registered."
`-locationId` *id*	Specifies an ID. Helpview 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 Helpview's app-defaults file (`/usr/lib/X11/app-defaults/Helpview`).

See Also

- "Registering Your Online Help" in Chapter 8
- "Viewing a Help Volume" in Chapter 4

Printing Help Topics ('helpprint' and 'helpprintrst')

The HP Help System uses the Helpprint application to print help topics. Helpprint can be run manually (with the `helpprint` command) or directly from Helpview (by choosing Print from the File menu).

A second printing program is provided for printing help volumes that contain multi-byte characters (such as Japanese or Korean). The `helpprintrst` command operates just like the `helpprint` command except that its output does not depend on printer fonts. Instead, `helpprintrst` creates a page-size graphic image of each help topic.

Both printing programs are in the `/usr/vhelp/bin/` directory. If you are using HP VUE 3.0, they are also in the `/usr/vue/bin/` directory.

Command Syntax

`helpprint -helpVolume` *volume* `[ -locationId` *id* `] [ -R ]`

`helpprintrst -helpVolume` *volume* `[ -locationId` *id* `] [ -R ]`

Where:

`helpVolume` *volume*	Specifies a full path to the help volume (`.hv` file) that contains the topics to be printed. This parameter is required.
`-locationID` *id*	Identifies the topic to print. The default topic is "`_hometopic`"—which applies if this parameter is not specified.
`-R`	Recursively prints all the subtopics that are beneath the topic specified in the `-locationID` parameter. If the `-R` parameter is not used, only the specified topic is printed.

Examples

Each of the following commands prints topics from the HPHelpKit volume (which is the online version of this guide):

To print just the copyright topic:

```
helpprint -helpVolume /etc/vhelp/volumes/C/HPHelpKit.hv -locationId _copyright
```

To print the entire volume:

```
helpprint -helpVolume /etc/vhelp/volumes/C/HPHelpKit.hv -R
```

To print the "Command Summary" section and all its subtopics:

```
helpprint -helpVolume /etc/vhelp/volumes/C/HPHelpKit.hv -locationId CommandSummary -R
```

See Also

- "Processing HelpTag Files ('helptag')"
- "Viewing a Help Volume" in Chapter 4

14

Summary of Application Programmers Interface

The HP Help System's Application Programmers Interface (API) includes the following functions:

- Functions for creating and working with help dialogs:

  "XvhCreateHelpDialog()"
  "XvhCreateQuickHelpDialog()"
  "XvhQuickDialogGetChild()"

- Function for implementing item help mode:

  "XvhReturnSelectedWidgetId()"

- Functions for working directly with help text:

  "XvhGetTopicData()"
  "XvhProcessLinkData()"
  "XvhFreeTopicData()"

- Function for specifying the message catalog for the Xvh library:

  "XvhSetCatalogName()"

XvhCreateHelpDialog()

```
#include <Xvh/Xvh.h>
#include <Xvh/HelpDialog.h>

Widget XvhCreateHelpDialog (
     Widget    parent,
     String    name,
     ArgList   arglist,
     Cardinal  argcount );
```

Where:

parent	Specifies the parent widget ID.
name	Specifies the name of the new help dialog.
arglist	Specifies the argument list.
argcount	Specifies the number of attribute-value pairs in the argument list (*arglist*).

This function creates a new instance of a general help dialog and returns its ID. The widget ID returned is a Dialog Shell widget which serves as the top level child in the created help dialog. Refer to the

OSF/Motif documentation for more information on the Dialog Shell widget and applicable resources.

Resources

The following resources are specific to the help dialog widget.

`XmNcloseCallback`

Specifies the list of callback functions executed when the user chooses Close from the dialog's File menu. (See "Detecting When Help Dialogs are Dismissed" in Chapter 6.)

`XmNcolumns`

Specifies the desired width of the help display area. The `XmNcolumns` resource expects a number (type `Dimension`) value that represents the number of average-width characters (based on the current font). The default is `40`.

`XmNhelpFile`

Specifies the name of a text file to be displayed in the help dialog. This resource is used only when the `XmNhelpType` resource is set to `XvhHELP_TYPE_FILE`.

The `XmNhelpFile` resource expects a string (`char *`) value. Its default is the NULL string.

`XmNhelpOnHelpVolume`

Specifies the help volume used to display "help on help." This volume is accessed if the user requests help while using the help dialog.

The `XmNhelpOnHelpVolume` resource expects a string (`char *`) value. Its default is the NULL string.

`XmNhelpPrint`

Specifies the command used to print help topics. The `XmNhelpPrint` resource expects a string (`char *`) value. Its default is `/usr/vhelp/bin/helpprint`.

`XmNhelpType`

Specifies the type of information to be displayed in the help dialog. Valid values, which are defined in `<Xvh/Xvh.h>`, include the following:

`XvhHELP_TYPE_TOPIC` indicates that the information is a formatted help topic. The displayed topic is located using the ID specified by the `XmNlocationId` resource in the volume specified by `XmNhelpVolume`. (`XvhHELP_TYPE_TOPIC` is the default value for the `XmNhelpType` resource.)

`XvhHELP_TYPE_STRING` indicates that the information is a text string provided in the `XmNstringData` resource. Newline

characters within the string are used to determine line breaks when formatting the string.

`XvhHELP_TYPE_DYNAMIC_STRING` indicates that the information is a text string provided in the `XmNstringData` resource. Newline characters within the string are used to separate paragraphs. Text in the string is automatically wrapped to fit the current size of the window.

`XvhHELP_TYPE_MAN_PAGE` indicates that the information is a man page. The text to be displayed is retrieved internally by executing the `man` command using the value of the `XmNmanPage` resource.

`XvhHELP_TYPE_FILE` indicates that the information is in a text file. The file name is identified using the value of the `XmNhelpFile` resource.

`XmNhelpVolume`

Specifies the help volume to use. This resource is used in conjunction with the `XmNlocationId` resource, which specifies an ID within the volume. The `XmNhelpType` resource must be set to `XvhHELP_TYPE_TOPIC`.

The `XmNhelpVolume` resource expects a string (`char *`) value. Its default is the NULL string.

`XmNhyperLinkCallback`

Specifies the list of callback functions executed when a hyperlink event occurs within the help dialog. Callbacks should be added to the dialog using the standard `XtAddCallback()` function. (See "Responding to Hyperlink Events" in Chapter 6.)

`XmNlocationId`

Specifies the ID string for a help topic. The `XmNhelpVolume` resource must be set to specify the help volume in which the corresponding ID resides, and `XmNhelpType` must be set to `XvhHELP_TYPE_TOPIC`.

The `XmNlocationId` resource expects a string (`char *`) value. Its default is the NULL string.

`XmNmanPage`

Specifies a man page to display. This resource is used when the `XmNhelpType` resource is set to `XvhHELP_TYPE_MAN_PAGE`.

The `XmNmanPage` resource expects a string (`char *`) value. Its default is the NULL string. The string is passed directly to the system `man` command to find and display the man page.

`XmNprinter`

Specifies the printer device name to be used for printing help topics.

The XmNprinter resource expects a string (char *) value. Its default is the NULL string, which causes printed help topics to be directed to the system's default printer.

XmNrows

Specifies the desired height of the help display area in terms of the number of rows of text. The height of each row is determined by the current font scheme in use. The XmNrows resource expects a number (type Dimension) value. The default is 15.

XmNscrollBarDisplayPolicy

Controls the automatic placement of scroll bars around the help topic display area. If set to XvhAS_NEEDED_SCROLLBARS, scroll bars are displayed only if the help text doesn't completely fit within the display area. (XvhAS_NEEDED_SCROLLBARS is the default value for this resource.)

If XmNscrollBarDisplayPolicy is set to XvhSTATIC_SCROLLBARS, the scroll bars are always managed, regardless of the size of the current help topic.

If XmNscrollBarDisplayPolicy is set to XvhNO_SCROLLBARS, the scroll bars are never managed, even if the current help topic is too big to competely fit within the display area.

XmNstringData

Specifies a string of characters to display in the help dialog. This resource is used when the XmNhelpType resource is set to XvhHELP_TYPE_STRING or XvhHELP_TYPE_DYNAMIC_STRING.

If XmNhelpType is set to XvhHELP_TYPE_STRING, newline characters in the string are used to determine the line breaks when formatting the text. If XmNhelpType is set to XvhHELP_TYPE_DYNAMIC_STRING, newline characters are interpreted as paragraph separators and the string is automatically wrapped to fit the current display area.

The XmNstringData resource expects a string (char *) value. Its default is the NULL string.

XmNtopicTitle

Specifies the topic title to be used in conjunction with either the XmNstringData or XmNhelpFile resource. The topic title is used to represent the topic in the History list. This resource is used only when the XmNhelpType is set to XvhHELP_TYPE_STRING, XvhHELP_TYPE_DYNAMIC_STRING, or XvhHELP_TYPE_FILE.

The XmNtopicTitle resource expects a string (char *) value. Its default is the NULL string.

XmNvisiblePathCount

Specifies the height of the Topic Hierarchy area (which is just below the help dialog's menu bar). The XmNvisiblePathCount resource expects an int (integer) value that specifies the number of visible items. The default value is 4.

Callback Information

A pointer to the following structure is passed to each callback.

```
typedef struct {
     int      reason;
     XEvent  *event;
     char    *locationId;
     char    *helpVolume;
     char    *specification;
     int      hyperType;
} XvhHelpDialogCallbackStruct;
```

Where:

reason	This element indicates why the callback was invoked.
event	This element points to the event that triggered the callback.
locationId	This element points to the current topic ID. If the dialog is not displaying a topic, this value is NULL.
helpVolume	This element points to the current help volume. If the dialog is not displaying a formatted help topic, this value is NULL.
specification	This element points to author-supplied data that is used in application-defined hyperlinks. This value is NULL if the event that invoked the callback was not an application-defined hyperlink.
hyperType	This element indicates the type of hyperlink that invoked this callback. The value is one of: XvhLINK_JUMP_NEW, XvhLINK_MAN, or XvhLINK_APP_DEFINE.

Usage Tips

- To destroy an instance of a help dialog, use XtDestroyWidget().
- To display a help dialog, use XtManageChild().
- To hide a help dialog, use XtUnmanageChild().

See Also

- "Displaying Help Topics" in Chapter 6
- "Responding to Hyperlink Events" in Chapter 6
- "Detecting When Help Dialogs are Dismissed" in Chapter 6
- "To create a general help dialog" in Chapter 5
- "Creating a Dialog Cache" in Chapter 5

XvhCreateQuick-HelpDialog()

```
#include <Xvh/Xvh.h>
#include <Xvh/QuickHelpD.h>

Widget XvhCreateQuickHelpDialog (
     Widget    parent,
     String    name,
     ArgList   arglist,
     Cardinal  argcount );
```

Where:

parent	Specifies the parent widget ID.
name	Specifies the name of the new help dialog.
arglist	Specifies the argument list.
argcount	Specifies the number of attribute-value pairs in the argument list (*arglist*).

This function creates a new instance of a general help dialog and returns its ID. The widget ID returned is a Dialog Shell widget which serves as the top level child in the created help dialog. Refer to the OSF/Motif documentation for more information on the Dialog Shell widget and applicable resources.

Resources

The following resources are specific to the quick help dialog widget.

`XmNbackLabelString`

Specifies the string label for the Back button. The `XmNbackLabelString` resource expects a compound string (type `XmString`) value. The default string is "Backtrack."

`XmNcolumns`

Specifies the desired width of the help display area. The `XmNcolumns` resource expects a number (type `Dimension`) value that represents the number of average-width characters (based on the current font). The default is `40`.

`XmNhelpFile`

Specifies the name of a text file to be displayed in the help dialog. This resource is used only when the XmNhelpType resource is set to XvhHELP_TYPE_FILE.

The XmNhelpFile resource expects a string (char *) value. Its default is the NULL string.

XmNhelpPrint

Specifies the command used to print help topics. The XmNhelpPrint resource expects a string (char *) value. Its default is /usr/vhelp/bin/helpprint.

XmNhelpType

Specifies the type of information to be displayed in the quick help dialog. Valid values, which are defined in <Xvh/Xvh.h>, include the following:

XvhHELP_TYPE_TOPIC indicates that the information is a formatted help topic. The topic is located using the ID specified by the XmNlocationId resource in the volume specified by XmNhelpVolume. (XvhHELP_TYPE_TOPIC is the default value for the XmNhelpType resource.)

XvhHELP_TYPE_STRING indicates that the information is the text string in the XmNstringData resource. Newline characters within the string are used to determine line breaks when formatting the string.

XvhHELP_TYPE_DYNAMIC_STRING indicates that the information is the text string in the XmNstringData resource. Newline characters within the string are used to separate paragraphs. Text in the string is automatically wrapped to fit the current size of the window.

XvhHELP_TYPE_MAN_PAGE indicates that the information is a man page. The text to be displayed is retrieved internally by executing the man command using the value of the XmNmanPage resource.

XvhHELP_TYPE_FILE indicates that the information is in a text file. The file name is identified using the value of the XmNhelpFile resource.

XmNhelpVolume

Specifies the help volume to use. This resource is used in conjunction with the XmNlocationId resource, which specifies an ID within the volume. The XmNhelpType resource must be set to XvhHELP_TYPE_TOPIC.

The XmNhelpVolume resource expects a string (char *) value. Its default is the NULL string.

XmNhyperLinkCallback

Specifies the list of callback functions executed when a hyperlink event occurs within the help dialog. Callbacks should be added

to the dialog using the standard `XtAddCallback()` function. (See "Responding to Hyperlink Events" in Chapter 6.)

`XmNlocationId`

Specifies the ID string for a help topic. The `XmNhelpVolume` resource must be set to specify the help volume in which the corresponding ID resides, and `XmNhelpType` must be set to `XvhHELP_TYPE_TOPIC`.

The `XmNlocationId` resource expects a string (`char *`) value. Its default is the NULL string.

`XmNmanPage`

Specifies a man page to display. This resource is used when the `XmNhelpType` resource is set to `XvhHELP_TYPE_MAN_PAGE`.

The `XmNmanPage` resource expects a string (`char *`) value. Its default is the NULL string. The string is passed directly to the system `man` command to find and display the man page.

`XmNminimizeButtons`

Specifies whether the dialog's buttons should be resized so they are all the same width as the widest button and the same height as the tallest button. The `XmNminimizeButtons` resource expects a Boolean value (`True` or `False`). The default is `True`, which makes the buttons all the same size. If this resource is `False`, the button sizes are not altered from their default sizes.

`XmNmoreLabelString`

Specifies the string label for the More button. The `XmNprintLabelString` resource expects a compound string (type `XmString`) value. The default string is "More." If this button is used to display a general help dialog, the recommended label is "Browse ... "

`XmNokCallback`

Specifies the list of callback functions executed when the user chooses the OK button. The callback reason is `XvhCR_OK`. (See "Detecting When Help Dialogs are Dismissed" in Chapter 6.)

`XmNokLabelString`

Specifies the string label for the OK button. The `XmNokLabelString` resource expects a compound string (type `XmString`) value. The default string is "OK."

`XmNprinter`

Specifies the printer device name to be used for printing help topics.

The `XmNprinter` resource expects a string (`char *`) value. Its default is the NULL string, which causes printed help topics to be directed to the system's default printer.

XmNrows

Specifies the desired height of the help display area in terms of the number of rows of text. The height of each row is determined by the currently used font scheme. The XmNrows resource expects a number (type Dimension) value. The default is 15.

XmNscrollBarDisplayPolicy

Controls the automatic placement of scroll bars around the help topic display area. If set to XvhAS_NEEDED_SCROLLBARS, scroll bars are displayed only if the help text doesn't completely fit within the display area. (XvhAS_NEEDED_SCROLLBARS is the default value for this resource.)

If XmNscrollBarDisplayPolicy is set to XvhSTATIC_SCROLLBARS, the scroll bars are always managed, regardless of the size of the current help topic.

If XmNscrollBarDisplayPolicy is set to XvhNO_SCROLLBARS, the scroll bars are never managed, even if the current help topic is too big to competely fit within the display area.

XmNstringData

Specifies a string of characters to display in the help dialog. This resource is used when the XmNhelpType resource is set to XvhHELP_TYPE_STRING or XvhHELP_TYPE_DYNAMIC_STRING.

If XmNhelpType is set to XvhHELP_TYPE_STRING, newline characters in the string are used to determine the line breaks when formatting the text. If XmNhelpType is set to XvhHELP_TYPE_DYNAMIC_STRING, newline characters are interpreted as paragraph separators and the string is automatically wrapped to fit the current window size.

The XmNstringData resource expects a string (char *) value. Its default is the NULL string.

XmNhelpLabelString

Specifies the string label for the Help button. The XmNhelpLabelString resource expects a compound string (type XmString) value. The default string is "Help."

XmNprintLabelString

Specifies the string label for the Print button. The XmNprintLabelString resource expects a compound string (type XmString) value. The default string is "Print ... "

Callback Information

A pointer to the following structure is passed to each callback.

```
typedef struct {
     int      reason;
     XEvent  *event;
```

```
        char    *locationId;
        char    *helpVolume;
        char    *specification;
        int      hyperType;
   } XvhHelpDialogCallbackStruct;
```

`reason`	This element indicates why the callback was invoked.
`event`	This element points to the event that triggered the callback.
`locationId`	This element points to the current topic ID. If the dialog is not displaying a topic, this value is NULL.
`helpVolume`	This element points to the current help volume. If the dialog is not displaying a formatted help topic, this value is NULL.
`specification`	This element points to author-supplied data that is used in application-defined hyperlinks. This value is NULL if the event that invoked the callback was not an application-defined hyperlink.
`hyperType`	This element indicates the type of hyperlink that invoked this callback. The value is one of: `XvhLINK_JUMP_NEW`, `XvhLINK_MAN`, or `XvhLINK_APP_DEFINE`.

Usage Tips

- To destroy an instance of a quick help dialog, use `XtDestroyWidget()`.
- To display a quick help dialog, use `XtManageChild()`.
- To hide a quick help dialog, use `XtUnmanageChild()`.

See Also

- "Displaying Help Topics" in Chapter 6
- "Responding to Hyperlink Events" in Chapter 6
- "Detecting When Help Dialogs are Dismissed" in Chapter 6
- "Using the Application-Defined Button" in Chapter 6
- "To create a quick help dialog" in Chapter 5
- "Creating a Dialog Cache" in Chapter 5

XvhQuickDialogGetChild()

```
#include <Xvh/QuickHelpD.h>

Widget XvhQuickDialogGetChild (
     Widget          widget,
     unsigned char   child );
```

Description

XvhQuickDialogGetChild is used to access a component within a Quick Help Dialog. The parameters given to the function are the Quick Help Dialog widget and a value indicating which child to access.

widget Specifies the widget ID of the quick help dialog.

child Specifies a component within the Quick Help Dialog. The following are legal values for this parameter:

```
XvhDIALOG_OK_BUTTON
XvhDIALOG_PRINT_BUTTON
XvhDIALOG_HELP_BUTTON
XvhDIALOG_SEPARATOR
XvhDIALOG_MORE_BUTTON
XvhDIALOG_BACK_BUTTON
```

Return Value

Returns the widget ID of the specified Quick Help Dialog child. An application should not assume that the returned widget will be of any particular class.

Usage

XvhQuickDialogGetChild() allows developers to create and display Quick Help Dialogs with different button configurations.

XvhReturnSelectedWidgetId()

```
#include <Xvh/HelpUtil.h>

int XvhReturnSelectedWidgetId (
     Widget   parent,
     Cursor   cursor,
     Widget  *widget );
```

Where:

parent Specifies the widget ID to use as the basis of the interaction. This can be any valid widget within the application's widget hierarchy. Usually it is a top-level or application shell widget.

cursor Specifies the shape to be used for the pointer during the interaction. If *cursor* is NULL, the function uses a default pointer shape.

widget The ID of the widget that the user selects. If the value returned is NULL, the function was canceled with the Esc key or an error occurred.

Description

This function temporarily grabs the pointer so the user can select any widget on the screen. The function completes when the user selects a widget or presses Esc to cancel the function.

If a successful selection has been made, the *widget* parameter contains the widget ID of the selected widget.

The function always returns one of the following exit status values:

`XvhSELECT_VALID` Indicates that the selection was successful. The *widget* parameter should contain the ID of the selected widget.

`XvhSELECT_INVALID` Indicates that the user selected an invalid widget outside the scope of the current application's widget hierarchy.

`XvhSELECT_ABORT` Indicates that the user canceled the function by pressing the Esc key.

`XvhSELECT_ERROR` Indicates that the function terminated due to an error.

See Also

- "Supporting Item Help Mode" in Chapter 6 explains how the `XvhReturnSelectedWidgetId()` function is used to implement "item help mode."

XvhGetTopicData()

```
#include <Xvh/FormatTerm.h>

int XvhGetTopicData (
     char            *helpVolume,
     char            *locationId,
     int              maxColumns,
     char          ***helpList,
     XvhHyperLines  **hyperList );
```

Where:

helpVolume Specifies the help volume file. This must be a fully qualified path to a help volume file (*volume*`.hv`).

locationId Specifies the topic ID to search for within the *helpVolume*.

maxColumns Specifies the maximum number of characters per line for formatting the help text.

helpList A returned value containing a pointer to a list of NULL-terminated strings. Each string is a line of help text. Blank lines are allocated with zero-length line. The caller is responsible for freeing the *helpList* information using the `XvhFreeTopicData()` function.

hyperList A returned value containing a pointer to a list of NULL-terminated hyperlink specifications. See below for more information. The caller is responsible for freeing the *hyperList* information using the `XvhFreeTopicData()` function.

The `XvhGetTopicData()` function provides a mechanism for retrieving help text from a help volume. All graphics and special characters are stripped from the data.

Hyperlinks found within the topic are assembled in a list (*hyperList*) and a pointer to that list is returned. It is up to the application to display the list of possible links and provide a user interface for choosing a link. When a link is chosen, the `XvhProcessLinkData()` function is used to follow the link to display the related topic.

If any problems occurred while processing the information, a value of `-1` is returned and the appropriate error message is generated. If the requested information is found and processed with no errors, then this function returns zero (`0`).

The hypertext specifications returned by the `XvhGetTopicData()` function are of type `XvhHyperLines`, which is a structure defined as follows:

```
typedef struct {
     char  *title;
     char  *linkData;
     int    hyperType;
} XvhHyperLines;
```

Where:

`title` This element specifies the title to the hypertext topic that is pointed to by `linkData`.

`linkData` This element specifies a pointer to a string that is the hypertext link information. Depending on the type of hypertext link, this could be a file name, an ID string, or a command to execute.

`hyperType` This element specifies the hyperlink type, which is one of these values: `XvhLINK_JUMP_REUSE`, `XvhLINK_JUMP_NEW`, `XvhLINK_DEFINITION`, `XvhLINK_EXECUTE`, or `XvhLINK_APP_DEFINED`.

XvhProcessLinkData()

```
#include <Xvh/FormatTerm.h>

int XvhProcessLinkData (
     XvhHyperLines   *hyperList,
     char            **helpVolume,
     char            **locationId );
```

Where:

hyperList — Is a pointer to an individual structure in the *hyperList* value returned by the `XvhGetTopicData()` function. The caller is responsible for freeing the *hyperList* information using the `XvhFreeTopicData()` function.

helpVolume — Specifies the help volume file. This must be a fully qualified path to a help volume file (*volume*`.hv`).

locationId — Specifies the topic ID to locate within the *helpVolume*.

The `XvhProcessLinkData()` function provides a mechanism for traversing hyperlinks that occur within a topic retrieved with the `XvhGetTopicData()` function.

If the requested information is found and formatted with no errors, a status of zero (`0`) is returned. If any problems occurred, a value of `-1` is returned.

XvhFreeTopicData()

```
#include <Xvh/FormatTerm.h>

void XvhFreeTopicData (
     char            **helpList;
     XvhHyperLines   *hyperList; );
```

Where:

helpList — Is a pointer to a list of strings to be freed.

HyperList — Is a pointer to a list of hyperlink specifications to be freed.

The `XvhFreeTopicData()` function frees the memory allocated for structures returned by the `XvhGetTopicData()` function.

XvhSetCatalog-Name()

```
#include <Xvh/Xvh.h>

void XvhSetCatalogName (
     char            *catFile; );
```

Where:

catFile The name of the message catalog file.

The `XvhSetCatalogName()` function sets the name of the message catalog used to supply the labels and error messages for help dialogs.

Users of the HP Help System Developer's Kit who ship a translated version of their product must uniquely name their translated version of the `Xvh.cat` message catalog file (for example *appName*`_Xvh.cat`). This ensures that other products will not overwrite the message catalog file for your application's help dialogs .

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 Helpview.)

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**.

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

dialog cache
A list of help dialogs that have 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.

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.

figure
A graphic or illustration that appears in the help information.

general help dialog
A window in which help information is displayed. General help dialogs have a menu bar, a Topic Hierarchy (which displays the current topic location), and a help topic display area. In addition, the dialog has the following subdialogs: History, Keyword Index, and Print. See also **quick help dialog**.

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**.

help browser
A general purpose application for viewing the online help installed on a system. HP VUE 3.0 uses the Helpview program as a "help browser" by displaying a special browser volume that lists the help installed on the system. (HP VUE uses a utility called `helpgen` to create the browser volume.)

help callback
An application function called when the user presses the F1 key.

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 about how to use the help dialog windows. 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.

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
A dialog that shows 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. The History dialog remembers only the 20 most recent 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 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.

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.

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 window. Jump-new-view links are intended for cross-volume links. The user senses a "new context" by a new window being displayed.

keyword index
A list of important words and phrases that appear throughout a help volume. The keyword index, like the index in a book, is an alphabetical list of the words and a list of each important occurrence. The HP Help System presents the keyword index in a dialog when the user chooses Keyword from the Search menu (in a general help dialog).

man page link
A hyperlink which, 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.

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 HP 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.

product family
A set of help volumes that are related to one another because the applications they refer to are related.

quick help dialog
A simple window that displays help information. A quick help dialog has a help topic display area and a few buttons. See also **general help dialog**, 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 `helptag` command. These are the files distributed to users who will use the HP Help System.

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 begin 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*''

stand-alone 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 stand-alone help volume can be displayed using the `helpview` command.

Tagged Image File Format (TIFF)
A standard graphics file format. The HP Help System dialogs support TIFF 5.0 images. TIFF images are identified by the `.tif` filename extension.

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 (`>`).

topic
Information about a specific subject. Usually, this is about 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 (via hyperlinks) to progressively more detailed topics. See also **home topic**.

warning
Help 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` filename extension.

X pixmap
A multi-color image. Pixmap image files are identified by the `.pm` filename 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` filename extension.