A Media-Rich Online Help System

Based on an existing fast and easy-to-use online help system, the CDE help system extends this baseline to provide features that will work across all UNIX® platforms.

by Lori A. Cook, Steven P. Hiebert, and Michael R. Wilson

With the growing demand by users for a unified desktop strategy across all UNIX operating system platforms comes the requirement for a standard help system. Users expect some base level of online help to be provided from within the desktop they are using. They expect online information to be easy to use and graphical, with growing expectations of direct audio and video support and interactive capabilities.

The Common Desktop Environment (CDE) help system provides media-rich, online information that is both fast and easy to use. It has its basis in the standard online help system from HP that is used extensively by HP VUE 3.0 and HP MPower[1] components and by many other HP OSF/Motif-based products.

Background

The CDE 1.0 help system originated with HP VUE. An early version, HP VUEhelp 2.0, satisfied few of the requirements of a modern help system. VUEhelp 2.0 did not allow rich text and graphics, was hard to integrate into an application, lacked authoring tools, and suffered from poor performance. The HP VUE 3.0 help system delivered a complete solution for creating, integrating, and shipping rich online information with an OSF/Motif-based application, while keeping its presence (use of system resources) to a minimum. HP VUE 3.0 walked a very fine line between providing the rich set of features that customers required while maintaining performance. In 95% of the cases where features and performance came into conflict, performance won. The HP VUE 3.0 help system was submitted to the CDE 1.0 partners for consideration. It was the only existing, unencumbered, rich text help system put forward for consideration. After an evaluation period, the HP VUE 3.0 help system was accepted almost as is. It contained all the functionality required, and with a little work, it could use a standard distribution format.

The Help Developer's Kit

The CDE 1.0 developer's kit includes a complete system for developing online help for any OSF/Motif-based application. It allows authors to write online help that includes rich graphics and text formatting, hyperlinks, and communication with the application. It provides a programmer's toolkit that allows developers to integrate this rich online help information with their client applications. The help dialog widget serves as the main display window for the CDE 1.0 help system. A second, lighter-weight help widget, called *quick help dialog*, is also available from the toolkit. Following is the list of components supported within the toolkit:

For authors:

- The CDE 1.0 HelpTag markup language. This is a set of tags used in text files to mark the organization and content of online help. HelpTag is based on SGML (Standard Generalized Markup Language).
- The CDE 1.0 HelpTag software. This is a set of software tools for converting the authored HelpTag files into their run-time equivalents used to display the online help information. CDE 1.0 HelpTag is a superset of the HelpTag used with HP VUE 3.0. HP VUE 3.0 HelpTag source files compile under CDE 1.0 with no modification. One exception to this is if authors use the old HP VUE 3.0 help distribution format.
- The dthelpview application. This program is for displaying online help so it can be read and interacted with in the same manner as end users will use it.

For programmers:

- The DtHelp programming library. This is an application programming interface (API) for integrating help windows (custom OSF/Motif widgets) into an application.
- A demonstration program. This is a simple example that shows how to integrate the CDE 1.0 help system into an OSF/Motif application.

Changed or new features for CDE 1.0:

- Public distribution format based on SGML conforms to standards.
- A new keyword searching capability allows users to search across all volumes on the system.
- A new history dialog allows the user to select previously viewed volumes.
- A different table of contents mechanism allows full browsing of the volume without using hypertext links.

- A richer message formatting capability is provided. (While HelpTag does not allow tables, other documents that do allow tables can be translated into the public distribution format and displayed. Also, the table of contents now allows graphics in the titles.)
- Public text extraction functions are removed, but available for old HP VUE 3.0 users by redefining.

General Packaging Architecture

An online help system needs to feel like it is part of the host application to the end user, not an appendage hanging off to the side. For developers to leverage a third-party help system, it must be delivered in such a way as to provide easy and seamless integration into their applications. Furthermore, the effort and overhead of integrating and redistributing the help system along with their applications must be minimal, while at the same time meeting application and end-user requirements for a help system. Users should feel as if they have never left the application while getting help.

During the initial prototyping of the HP VUE 3.0 help system, two key issues kept occurring: performance and packaging. The help system needed to be light and fast and easy to integrate into any OSF/Motif-based application and redistribute with that application. HP VUE 2.0 help suffered greatly on both these points. VUEhelp 2.0 was server-based, large, slow, and dependent on the HP VUE desktop. Any application using this help service had to run within the HP VUE desktop environment.

These two issues were addressed in HP VUE 3.0 help and we found no reason to change the paradigm in the CDE development. To fix the performance problems (slow startup times), functionality is linked into the client via a library rather than starting up a separate help server process (as was the case for HP VUE 2.0). Since most OSF/Motif applications tend to incorporate the same widgets as those used by the help dialogs, the increase in data and text for attaching a help dialog to an application is minimal.

Fig. 1 represents two different approaches to integrating online help into an application. Our initial prototypes quickly exposed the value of a client-side embedded solution especially with respect to performance (e.g., fast rendering time) and memory use.

The following advantages are gained by using a library-based help system instead of a server-based architecture.

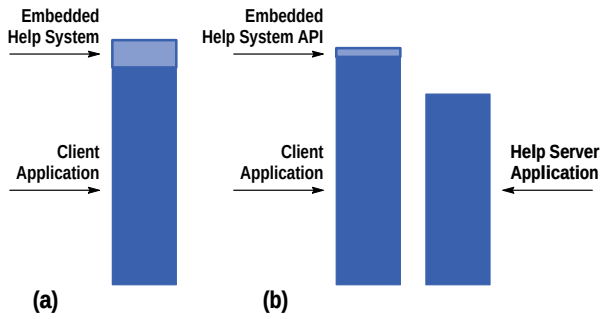
Integration advantages:

- OSF/Motif-based API (simple to use for Motif knowledgeable developers)
- Application control of the help system dialog management (creation, destruction, caching, and reuse)
- Smooth transition into help system via consistent resource settings between the application and the help system (same fonts and color scheme and quick response times)
- Ready support for a tightly coupled application-to-help-system environment (application defined and processed help controls such as hypertext links)
- Application-specific customizing of the help system (dialogs, controls, resources, and localization).

CDE dependencies:

- DtHelp is delivered in shared-library form only.
- Application must link with the CDE libraries DtSvc and tt.
- CDE is not required as long as the libraries it depends on are available.

Fig. 1. *Two different approaches to integrating online help into an application. (a) Client-side embedded implementation. This is the scheme used in CDE 1.0. (b) Server-side implementation of the help system. This was the implementation used in HP VUE 2.0.*



Integration Concepts, Practices, and Mechanisms

The intricacies of how to author online information, or the different ways a developer can integrate help into an application, are described in the CDE 1.0 help system developer's guide. We will describe in this section the general integration concepts and practices for which this help system was intended.

The run-time help facility is made up of a collection of help dialogs (complex, composite widgets), and compiled help volumes (described below). The help widgets are linked directly into the client application via the help library libDtHelp.sl (shared library) and instantiated by the client to display help information. The help dialogs serve only as the vehicle for displaying rich online help information, while standard OSF/Motif, Xlib, and Xt services serve as the glue to integrate them into the application. For this reason, it is necessary to stray from discussing the specifics of the help system and its components, and describe some of the OSF/Motif and toolkit mechanisms that must be used to integrate help into an application.

There are two levels of integration with respect to this help system: integrating help into an application and integrating a help-smart application into CDE.

Integrating Help into an OSF/Motif Application

Developers have many degrees of freedom with respect to how much or how little help they include in their applications. If an application and its help information have very loose ties, there may be only a handful of topics that the application is able to display directly. In contrast, the application could provide specific help for nearly every object and task in the application. This requires more work, but it provides potentially much greater benefits to the end user if done well.

Help menus and buttons in an application serve as the most basic of help entry points for an application. Reference 2 provides more information about integrating help menus and buttons into an application.

Contextual Help. Contextual help provides help information about the item on which the selection cursor* is positioned. Contextual help information provides users with information about a specific item as it is currently used. The information provided is specific to the meaning of the item in its current state.

The OSF/Motif user interface toolkit provides direct support for contextual help entry points via its help callback mechanism. When a valid help callback is added to a widget and the user presses the help key (**F1**) while that widget has the current keyboard focus (selection cursor), the widget's help callback is automatically executed. From within the help callback the application has the opportunity to display some help topic directly based on the selected widget, or the application could dynamically construct some help information based on the current context of the selected item.

Any level of granularity can be applied when adding help callbacks to an application's user interface components. They can be added to all the widgets and gadgets within the application dialogs, to the top-level windows for each of the dialogs, or to any combination in between.

If the user selects **F1** help with the selection cursor over a widget or gadget that has no help callback attached to it, the OSF/Motif help callback mechanism provides a clever fall-back mechanism for providing more general help. The help callback mechanism will jump to the nearest widget ancestor that has a help callback assigned, and invoke that callback. The theory is that if you don't have a specific help on that widget or gadget, then it's better to provide more general help than none at all.

Application developers are responsible for adding their own help callbacks to their user interface components within their application. OSF/Motif sets the help callbacks to NULL by default.

Item Help. Item help allows users to get help on a particular control, such as a button, menu, or window, by selecting it with the pointer. Item help information should describe the purpose of the item for which help is requested and tell users how to interact with that item. This information is typically reference-oriented.

Item help is usually accessed via an application's help under the On Item menu selection. Once selected, the selection cursor is replaced with a question mark cursor and users can choose the item on which they want help by making the selection over that item.

The CDE help system API utility function DtHelpReturnSelectedWidgetId() assists developers in providing item help within their applications. This function provides an interface for selection of a component within an application.

DtHelpReturnSelectedWidgetId() will return the widget ID for any widget in the user interface that the user has selected via the pointer. The application then has the opportunity to display some help information directly on the selected widget or gadget.

* A selection cursor is visual cue that allows users to indicate with the keyboard the item with which they wish to interact. It is typically represented by highlighting the choice with an outline box.

At any point while the question mark cursor is displayed, the user can select the escape key (**ESC**) to abort the function call. If the user selects any item outside the current applications window, the proper error value will be returned.

Once DtHelpReturnSelectedWidgetId() has returned the selected widget, the application can invoke the help callback on the returned widget or gadget to process the selected item.

From within the help callback, the application has the opportunity to display some help topics based on the selected widget, or it could dynamically construct some help information based on the current item selected.

Integrating a Help-Smart Application into the Desktop

There are no restrictions regarding where run-time help files are installed. However, a suggested practice is to set up the help file installation package with the ability to redirect the default help file install location (e.g., put them on a remote help server system). By default, the run-time help files should be installed with the rest of an application's files. Installing the run-time help files in the same location as the application's files gives system administrators more flexibility in managing system resources.

An important step in installing help files is registration. The registration process enables two important features of the CDE 1.0 help system: cross volume hyperlinks and product family browsing.

Registering Help Volumes. After the run-time files have been installed, the volume is registered by running the CDE 1.0 utility, dtappintegrate. This utility registers the application's help volume by creating a symbolic link from where the help volumes are installed to the registry directory /etc/dt/appconfig/help/<$LANG>. By using this utility to register the help volumes, the application can ask for a help volume by its name only and the DtHelp library will search the standard locations for the volume's registry. Then, no matter where the application's help volume ends up, the DtHelp library finds the symbolic link in a standard location and traces the link back to the real location. ***Article 2*** describes registration and the utility dtappintegrate.

If access to an application's help volume is restricted to the application, then the location of the help volume can be hardcoded into the application. The disadvantage of this method occurs when a system administrator moves the application's help volumes to another location. If the application looks in only one place for the help information, moving the help volume causes a serious problem.

Registering a Product Family. When registering a product family, which is a group of help volumes belonging to a particular product, a help family file (product.hf) should be created and installed with the rest of the product's help files. The dtappintegrate utility creates a symbolic link for the product file to a standard location where the browser generator utility, dthelpgen, can find and process it. The dthelpgen utility creates a special help volume that lists the product families and the volumes within each family installed on the system.

Access to Help Volumes

The CDE 1.0 help system has a simple, yet extensible mechanism for transparent access of help volumes installed on the desktop. It supports both local and remote access to help volumes and works with any number of workstations. The only dependencies required are proper help registration (discussed above), NFS services (i.e., remote systems mounted to the local server), and proper configuration of the help environment variables discussed below.

When an application creates an instance of a help widget the DtNhelpVolume resource can be set using one of two formats: a complete path to the volume.sdl file, or if the volume is registered, the base name of the volume. When using the base name, the help system searches several directories for the volume. The search ends when the first matching volume 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).

DTHELPUSERSEARCHPATH. This environment variable contains the user-defined search path for locating help volumes. The default value used when this environment variable is not set is:

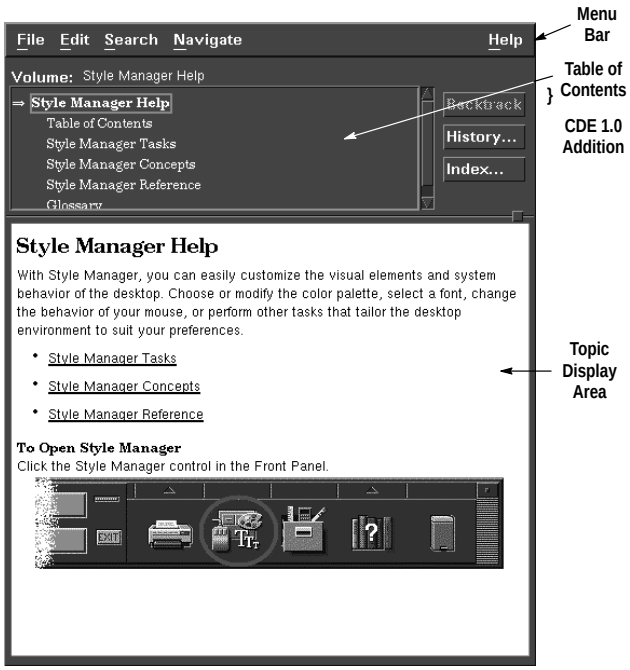
- dt/help/%T/%L/%H: \
- dt/help/%T/%L/%H.sdl:
- dt/help/%T/%L/%H.hv:

where:

- %L is the value of the $LANG environment variable (C is the default)
- %H is the DtNhelpVolume (help volume) resource specified
- %T is the type of file (volume or family) being searched for.

Whenever this resource is set to a relative path, the resource value will be prefixed with the user's default home directory.

Fig. 2. *General help dialog box.*



Examples:

Help volume: /user/vhelp/volumes/C/reboot.hv
Product family: /user/vhelp/family/SharedX.hf.

The .hv and .hf extensions are provided for backwards compatibility with the HP VUE 3.0 help and family volumes.

DTHELPUSERSEARCHPATH supplements any system search path defined. Therefore, the user uses this environment variable to access help volumes that should not be available to everyone.

DTHELPSEARCHPATH. This environment variable specifies the system search path for locating help volumes. The default values used when this environment variable is not set include:

- /etc/dt/appconfig/help/%T/%L/%H: \
- /etc/dt/appconfig/help/%T/%L/%H.sdl: \
- /etc/dt/appconfig/help/%T/%L/%H.hv: \

Where: %L, %H, and %T are the same as above.

When setting either of these environment variables it is a sound practice to append the new values to the current settings.

Help Widgets

The CDE 1.0 help dialog widgets are true OSF/Motif widgets. The syntax and use model for programmers is the same as for every OSF/Motif widget, whether custom or part of the toolkit. This makes it easy for developers familiar with OSF/Motif programming to understand and use help widgets.

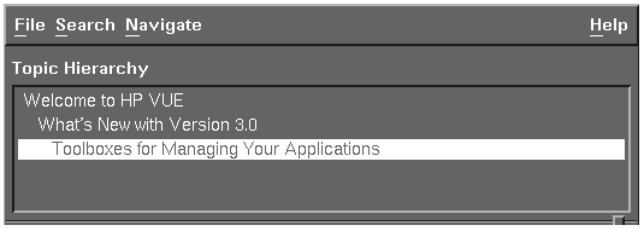
The OSF/Motif-based API directly supports various controls to manage an instance of a help dialog. Through standard OSF/Motif, Xt, and Xlib functions, programmers access help dialog resources and manipulate these values as they see fit. Developers add callbacks (XtAddCallback()), set and modify resources (XtSetValues()), manage and unmanage the dialogs (XtManageChild, and XtUnmanageChild) and free resources (XtDestroyWidget()).

The General Help Widget

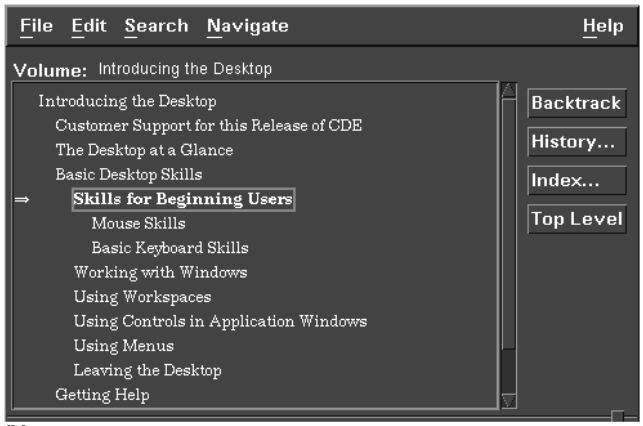
The general help widget for CDE 1.0 is very similar to the general help widget in HP VUE 3.0 help. It contains a menu bar, a table of contents area, and a topic display area. Added for CDE 1.0 are buttons to the right of the table of contents for easy access to menu items. Behavior for the table of contents area and keyword searching has changed for CDE 1.0. Fig. 2 shows a general help widget.

New Buttons. For convenience, buttons for the backtrack functionality, history, and index dialog boxes have been added to the general help dialog widget for CDE 1.0.

Fig. 3. *Help table of contents for (a) HP VUE 3.0 and (b) CDE 1.0.*



(a)



(b)

Table of Contents. The table of contents area changed significantly from its HP VUE 3.0 help system predecessor. The old version was a "here you are" type of listing, and it did not give the user the chance to explore other side topics unless they were direct ancestors of the current topic. Consequently, this type of table of contents required the authors to write help topics with extensive hypertext linking to other topics such as child topics that would be of direct interest to the user or direct descendants of the current topic. Fig. 3 shows the difference between HP VUE 3.0 and CDE 1.0 table of contents boxes.

CDE 1.0 help shows the topic, the subtopics related to the current topic, the ancestor topics, and the ancestor's sibling topics. As the user selects other topics from this table of contents, the listing changes so that it always shows the path to the topic, the siblings of any topic in the path, and the children of the current topic. Thus, the writer only has to include hypertext links to topics of interest if the topic is not a child of the current topic, an ancestor topic, or an ancestor's sibling.

Another change is that the table of contents now displays graphics. Before, the table of contents was limited to text. Now, the table of contents displays multifont and graphical headings.

Searching for a Keyword. The search dialog for CDE 1.0 help has been completely redesigned. A common complaint was that the old HP VUE keyword search dialog displayed or knew about keywords for the current volume only.

Now the user has the option to search all known volumes on the machine for a keyword. Since this can be a time-intensive operation, the search engine allows the user to interrupt a search operation. Additionally, the user can constrain the search to a limited set of volumes. Fig. 4 shows the new CDE 1.0 search dialog box.

Quick Help Dialog

From the programmer's point of view, the quick help dialog is a stripped-down version of the general help dialog. Its functionality is the same as HP VUE 3.0 quick help. It still contains a topic display area and a button box (see Fig. 5).

Display Area

From the developer's perspective the help dialogs are treated as a single widget. They are created, managed, and destroyed as one single object. If one were to look inside one of the help widgets there would be not one monolithic help entity but two separate very distinct components: the text display engine component and the widget component.

The display engine component, as seen by the developer, is the region in the help widget that renders the rich text and graphics and provides hyperlink access and dynamic formatting of the text. The widget component consists of the OSF/Motif code that makes it a widget and the remainder of the user interface, including topic hierarchy, menu bar, and supporting dialogs (print, index search, and history).

Fig. 4. *CDE 1.0 help search dialog box.*

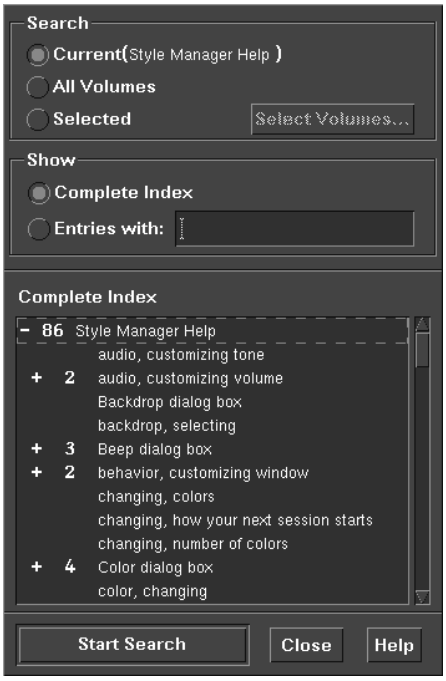
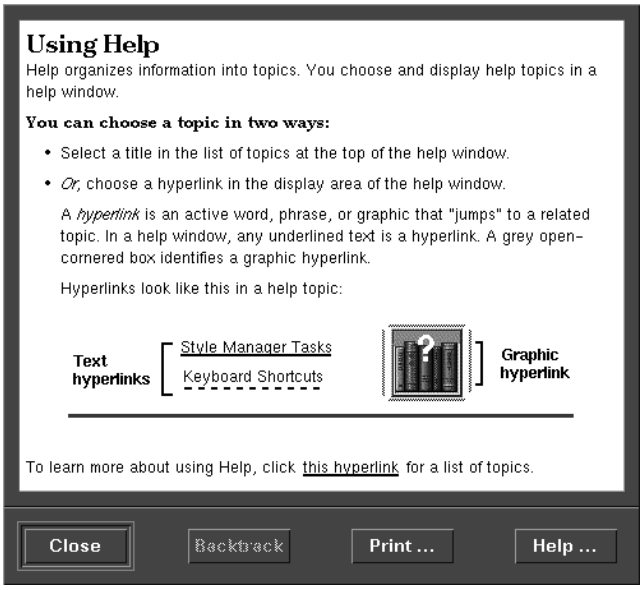


Fig. 5. *Quick help dialog box.*

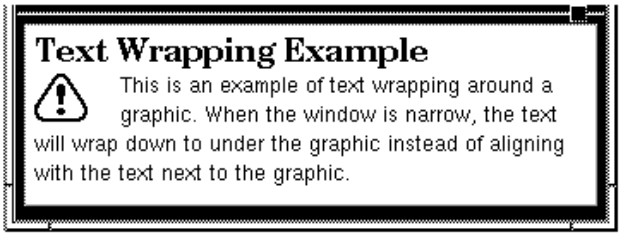


Display Area Text. The display area allows text to be rendered using many different fonts. It also allows for various types of text. The author can specify dynamic text that will reformat according to the window size and static text that will not. For dynamic text, a sequence of two or more spaces is compressed into a single space and internal new lines will be changed into a space. For static text, all spaces and internal new lines are honored.

To Reformat or Scroll? While vertical scrolling is accepted as necessary when help topics are longer than the available screen space, horizontal scrolling is not. Therefore, when users resize help windows, dynamic text is reformatted and figures are centered according to the new window size. This dynamic reformatting on the fly is seen as beneficial by the customer.

Scrolling Supported. Even using these line breaking rules, sometimes the lines are too long to fit in the available space. For this case or when static text pushes the boundary of the display area, the help system gives up and displays a scroll bar so that the user can see the information without having to resize the window.

Fig. 6. *Example of text flowing around a graphic.*



Flowing Text. The display area has the ability to flow text around a graphic. This is seen as a space-saving measure and highly desired functionality. The graphic can be placed on the left side or the right side of the display area and the text occupies the space to the side of the graphic (see Fig. 6). If the text is too long and does not fit completely in the space beside the graphic, the text wraps to below the graphic.

European (One-Byte) Rules. For most European languages (including English), breaking a line of text at spaces is sufficient. The only other line-breaking rule applied is to hyphens. If a word begins or ends with a hyphen, the hyphen is considered a part of the word. If the hyphen has a nonspace before and after it, it is considered a line breakable character and anything after it is considered safe to place on the next line.

Spaces and hyphens can be tagged by the author as nonbreaking characters. This allows the added flexibility for an author to demand that a word or phrase not be broken.

Asian (Multibyte) Rules. For Asian language support, breaking a line of text on a space is unacceptable since some multibyte languages do not break their words with spaces (e.g., Japanese and Chinese but not Korean). With the Japanese language, the characters are placed one after another without any word-breaking character, since each character is considered a word. There is also the concept that certain characters cannot be the first character on a line or the last character on a line. English (one-byte) characters can be mixed with multibyte characters.

Given these considerations, line breaking for multibyte languages boils down to the following rules:

1. Break on a one-byte space.
2. Break on a hyphen if the character before and after the hyphen is not a space.
3. Break on a one-byte character if it is followed by a multibyte character.
4. Break on a multibyte character if it is followed by a one-byte character.
5. Break between two multibyte characters if the first character can be the last character on a line, and the other character can be the first character on a line.

Rather than hard code the values of those Japanese characters that can't start or end a line into the CDE help system, the message catalogue system is used. This provides a general mechanism for any multibyte language. All the localizers are required to do is determine which characters in their language cannot start or end a line and localize the appropriate NLS file. The file /usr/dt/lib/nls/%l/%t/fmt_tbl.cat contains the values of those characters that are used for rule 5 of the line-breaking rules. If this file does not exist for a given language, rule 5 is ignored and line breaking will occur between any two multibyte characters.

One Library for All Locales. The help system uses the same library for processing one-byte or multibyte documents. It internalizes how many of the rules to use based on the $LANG environment variable and the character set used in the document. If a document specifies an ISO-LATIN1 character set, the display engine does not bother looking at Rules 3, 4, and 5 or using multibyte system routines. Only when the document and the $LANG environment variable specify a multibyte character set are these rules and routines used. This impacts the access and rendering time (minimally) but allows the same library to be used in the United States, Europe, and Asia.

Color Degradation. A feature of HP VUE 3.0 that was strongly desired was the ability to degrade color images. Having to provide three different images for each of three types of files is not feasible because of disk use. The color degradation algorithms used by the CDE 1.0 help system are the same as used by the HP VUE 3.0 help system. For TIFF (.tif) images, the HP image library forces the image to the proper color set depending on the display type. For X pixmaps (.xpm), the Xlib routines take the same approach depending on the display. This leaves the X window (.xwd) files to manipulate.

The task is to reduce an .xwd file from a full-color image to a grayscale image containing no more than the maximum number of gray colors available. The first step in this process maps each of the X Window colors that the image uses to a grayscale luminosity value.

This is done using the NTSC (National Television Standards Committee) formula for converting RGB values into a corresponding grayscale value:

$$\text{luminosity} = 0.299 \times \text{red} + 0.587 \times \text{green} + 0.114 \times \text{blue}$$

where red, green, and blue are the X window color values with a range of 0 to 255.

The next step is to count the number of distinct luminosity values used by the image. The help system then determines how many luminosity values to assign to each grayscale. For example, if there are 21 distinct luminosity values and eight gray colors, then the first five gray colors will represent three luminosity values, and the last three gray colors will represent two luminosity values. Next, a gray color is assigned to the luminosity values it will represent. The last step is to change the original color pixel to the gray color its luminosity value indicates.

If the number of distinct luminosity colors is less than the number of gray colors being used, then the help system spreads out the color use. For example, if there are three distinct luminosity values and eight gray colors, then the first, third, and sixth gray colors are used in the image, instead of using the first three gray colors. This provides a rich contrast to the image.

If the system has to degrade the image to black and white, it first calculates the grayscale colors using the luminosity calculation given above. Then the image is dithered using a Floyd-Steinberg error diffusion algorithm,[3] which incorporates a Stucki error filter.

Color Use. The user can force the help system to use grayscale or black and white by setting the HelpColorUse resource. The CDE 1.0 help system will also degrade an image if it cannot allocate enough color cells for the graphic. For example, if the image uses 31 unique colors, but there are only 30 color cells left, the help system will try to display it in grayscale and if that fails, the image will be dithered to use black and white.

Run-Time Help Volumes

The flexibility and power of this help system are largely placed in the author's hands. With the CDE HelpTag markup language and a creative author, very different and interesting approaches can be taken with respect to presenting information to the end user. Documents can be organized in either a hierarchy with hyperlinks referencing the children at any given level or in the form of a network or web, with a linear collection of topics connected via hyperlinks to related topics. It is up to the author to explore the many capabilities with respect to authoring online help for this system.

Help Volume Structure. A help volume is a collection of related topics that form an online book. Normally, the topics within a volume are arranged in a hierarchy, and when developing application help, there is one help volume per application. However, for complex applications or a collection of related applications, several help volumes might be developed.

Topics within a help volume can be referenced by unique location identifiers that are assigned by the author. Through these location identifiers help information is referenced in the run-time environment.

Help Volume Authoring. The authoring language for the CDE 1.0 help system is HelpTag. This authoring or markup language conforms to a variant of the Standard Generalized Markup Language (SGML (ISO 8879:1986)), which is a simple language consisting of about fifty keywords, or tags.

SGML is a metalanguage used to describe the syntax of markup languages. In a sense, SGML is very similar to the UNIX utility YACC in which the syntax of programming languages such as C is described in a formal, machine readable format. SGML itself does not provide a method for attaching semantics to markup constructs. That is, the equivalents of the YACC actions are not contained in or described by SGML. An SGML syntax description is known as a *document type definition* (DTD).

Other examples of markup languages described via SGML include HTML (HyperText Markup Language), which is used for documents on the World-Wide Web (WWW), DocBook, which is used for documentation of computer software, and PCIS (Pinnacles Component Information Standard), which is used for the exchange of semiconductor data sheets.

SGML is a very powerful markup description language and, as such, allows a great variety in the markup languages to be described. However, in a typical application SGML is used to

describe a highly structured markup language with the concept of containment playing a large role.* The concept of containment works very well for applying style to text because text styles can be pushed and popped on entry to and exit from a container.

SGML containers are known as elements. SGML elements are demarcated by the keywords or tags using the form:

```
<keyword> text..... text..... </keyword>
```

where keyword is replaced by the tag name. In SGML terms, the keyword making up the tag name is known as the generic identifier or GI. The form <keyword> is known as the start-tag, and the form </keyword> is known as the end-tag.

* Containment refers to the hiearchy of an item. For example, a book can contain chapters, chapters can contain sections, and sections can contain paragraphs, lists, and tables. Paragraphs can contain lists and tables, strings can be marked up as proper names, and so on.

An SGML element consists of the start-tag and all the text or contained markup up to and including the end-tag. For example, a simple paragraph would be marked up as:

```
<P> This is a paragraph with P being the generic identifier</P>
```

The syntax of SGML is itself mutable. SGML-conforming documents all start with an explicit or implicit SGML declaration in which SGML features are enabled or disabled for the document, the document character set is specified, and various parts of the SGML syntax are modified for the duration of the document. For example, the default form of an SGML end-tag is left angle bracket, slash, generic identifier, and right angle bracket (<, /, GI, >). For historical reasons, the form of an end-tag in HelpTag is left angle bracket, backslash, generic identifier, right angle bracket (<, \, GI, >). The change of slash to backslash in the end-tag opener is specified in the SGML declaration for HelpTag.

As implied by a previous paragraph, SGML elements may themselves contain other SGML elements. Depending upon the markup described in the document type definition, elements can be recursive.

Further, start-tags can contain attribute and value pairs to parameterize the start-tag. For example, the HelpTag element <list> has an attribute to determine if the list is to be an ordered (number or letter label) list or an unordered (no label) list. Creating an unordered list in HelpTag is done with the following markup:

```
<list>
* item 1
* item 2
* item 3
<\list>
```

which generates:

- item 1
- item 2
- item 3

To create an ordered list (starting with Arabic numerals, by default), one would enter:

```
<list type=order>
* item 1
* item 2
* item 3
<\list>
```

which generates:

1. item 1
2. item 2
3. item 3

To create an ordered list starting with uppercase Roman numerals in the labels, one would enter:

```
<list type=order order=uroman>
* item 1
* item 2
* item 3
<\list>
```

which generates:

I. item 1
II. item 2
III. item 3

Note that in the markup described above for list, the individual list items are initiated with the asterisk symbol. In this case, the asterisk has been defined as shorthand or, in SGML terms, a short reference for the full list item markup <item>. Without using short references, the unordered list markup given above as the first example would look like:

```
<list>
<item>item 1<\item>
<item>item 2<\item>
<item>item 3<\item>
<\list>
```

The short reference map of the HelpTag document type definition states that within a list element the asterisk character at the beginning of a line is shorthand for the list item start-tag, <item>, and for second and subsequent list items, also serves as the end-tag for the previous list item.

In HelpTag, the generic identifiers and attribute names are case-insensitive. Attribute values are case-sensitive for strings, but the enumerated values of those attributes with a fixed set of possible values are also case-insensitive.

To reduce typing when creating a help volume, the SGML features known as SHORTTAG and OMITTAG are set to the value yes for the classic HelpTag document type description. The most noticeable result of allowing these features is the ability to leave off the attribute names when specifying attributes in a start-tag. For example, the markup:

```
<list type=order>
```

is equivalent to:

```
<list order>
```

According to the SGML standard, the enumerated values (i.e., order and uroman), must be unique within an element's start-tag so the attribute name can be omitted without creating ambiguity.

Creating a HelpTag Source File. Currently there are no SGML tools for authoring HelpTag documents. To date, all HelpTag authoring has been done using the common UNIX text editors such as emacs and vi. The concept of short references has been used heavily to simplify the process of authoring the HelpTag source and minimizing the amount of typing necessary.

One hindrance to using SGML tools to author HelpTag source code is that the markup language does not adhere strictly to the SGML standard. In particular, short references have been used aggressively to the point of requiring extensions to the SGML syntax. Certain changes in the SGML declaration, while perfectly acceptable according to the standard, are not supported by any SGML tool vendor.

To enhance the possibility of future use of SGML tools for authoring HelpTag source code, a second version of the HelpTag document type definition (DTD) has been created. This second version of the DTD follows the SGML syntax to the letter and makes no use of esoteric features of SGML. This canonical form of the DTD is referred to as the formal HelpTag DTD.

The tools described here for processing HelpTag source code will accept documents conforming to both the classic HelpTag DTD and the formal HelpTag DTD. A switch to the dthelptag script -formal determines whether a document is to be processed according to the classic or formal HelpTag DTD.

The Semantic Delivery Language. The distribution format* chosen for CDE 1.0 changed from the format used in HP VUE 3.0. The old format could not be used for CDE 1.0 because:

- The distribution format was known only within HP and then only by some members of one division. The specification of this distribution format was never published or intended for publication.
- The potential for growth using this distribution format was severely restricted.
- The help volume existed in several files, resulting in problems such as losing one or more of the files during installation.

The run-time format of the CDE 1.0 help system is known as the Semantic Delivery Language, or SDL. SDL conforms to the ISO 8879: 1986 SGML standard. The benefits derived by moving to this distribution format are:

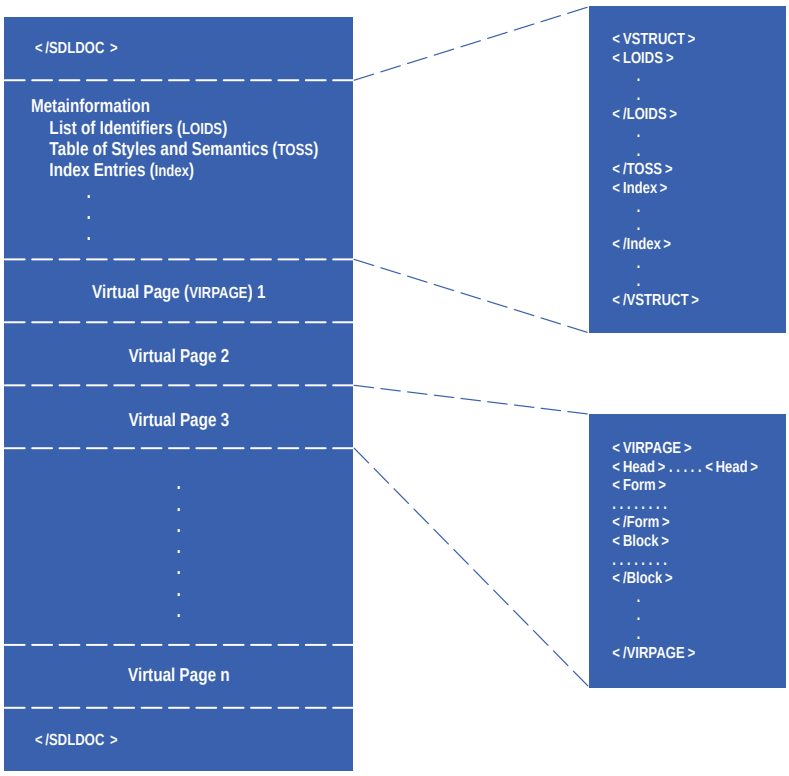
- SDL is based on SGML, which is a standard that is strongly supported and recognized in the desktop publishing arena.
- The format is publicly available. Anyone can create a parser to produce SDL.
- The growth potential of this public distribution format is unbounded.
- The resulting SDL exists in one file, reducing installation problems for developers.

The SDL language can be thought of as a halfway point between the typical SGML application, which ignores formatting in favor of describing the semantic content of a document (from which formatting is to be derived), and the typical page description language such as PostScript™ or nroff, which ignores the semantic content of a document in favor of rigorously describing its appearance.

Unlike typical SGML applications that break a document into specific pieces such as chapters, sections, and subsections, SDL breaks a document into generic pieces known as blocks and forms. SDL blocks contain zero or more paragraphs and SDL forms contain zero or more blocks or other forms (recursively). The SDL block is the basic unit of formatting, and the SDL form is a two-dimensional array of blocks and forms. Fig. 7 shows the structure and elements that make up an SDL volume, which is also a help volume.

* The file format of the help files delivered to customers.

Fig. 7. *The structure and elements of an SDL volume.*



Most elements in SDL have an attribute known as the *source semantic identifier* (SSI), which is used to indicate the meaning in the original source document (HelpTag in this case) of a particular construct that got converted into SDL markup. It is also used to help in the run-time formatting process by enabling searches into an SDL style sheet.

The SDL style sheet mechanism, known as the *table of semantics and styles* (TOSS), is also an element within the SDL document. SDL blocks or forms and their constituent parts can contain a source semantic identifier attribute and other attributes such as CLASS and LEVEL, which are used to match similar attributes on individual style elements in the table of semantics and styles. When the attributes of an element in the body of the document match a style element in the table of semantics and styles, the style specification in that style element is applied to the element in the document proper. That style specification is then inherited by all subelements where appropriate until overridden by another style specification.

Groups of SDL blocks and forms are collected in the SDL construct known as the virtual page (VIRPAGE). Each virtual page corresponds to an individual help topic. For example, the HelpTag elements chapter and s1 through s9 (subchapter levels 1 through 9) would each begin a new virtual page. An SDL virtual page is self-contained in that all the information needed to format that page is contained in the page. There is no need to format the pages preceding a virtual page to set a context for formatting the current page. Fig. 8 shows an example of a VIRPAGE.

For rapid access to help topics, an SDL file also contains an element known as the *list of identifiers* (LOIDS). When an SDL file is accessed and the virtual page containing a specific identifier is requested, the run-time help viewer can scan the list of identifiers to find the requested identifier. Each list entry contains a reference to the identifier it describes and the absolute byte offset from the beginning of the volume to the virtual page containing that identifier.

Since SDL virtual pages can be formatted independently of the preceding pages, the information in a list-of-identifiers entry can be used to get directly to the desired page and formatting can begin at that point.

Each of the entries in the list of identifiers also contains an indicator that tells whether the element containing the identifier is a paragraph, block, form, virtual page, or any of the other elements that can carry identifiers. In the case of virtual pages, the list-of-identifier entries also carry an indication of the page's semantic level (e.g., is the virtual page a representation of a chapter, subchapter, etc.).

Duplicating the level and type information of an element in the list of identifiers is a performance enhancement. Knowing which identifiers in the list correspond to virtual pages and knowing the semantic level of those pages allows a human-readable table of contents to be generated for a document by processing only the list of identifiers, avoiding reading and parsing the full document.

Fig. 8. *(a) An SDL statement (using SGML syntax) that defines a paragraph with a book title in it. (b) A VIRPAGE representation of the paragraph in (a).*

```
<P>This is a paragraph with a
<KEY CLASS="BOOK"
SSI="TITLE">Book Title
</KEY> in it.</P>
```

(a)

```
<VIRPAGE ID="MYCHAPTER" LEVEL="1"
 DOC-ID="MYDOC">
<HEAD>An Example Chapter</HEAD>
<BLOCK>
<P>This is a paragraph with
<KEY CLASS="BOOK"
SSI="TITLE"> Book Title
</KEY> in it.</P>
</BLOCK>
</VIRPAGE>
```

(b)

Processing a HelpTag Source File. The dthelptag compilation process performs a number of different tasks in generating the compiled run-time help volume:

- Syntax validation
- Conversion from the authored format to run-time format
- Location identifier map generation
- Topic compression.

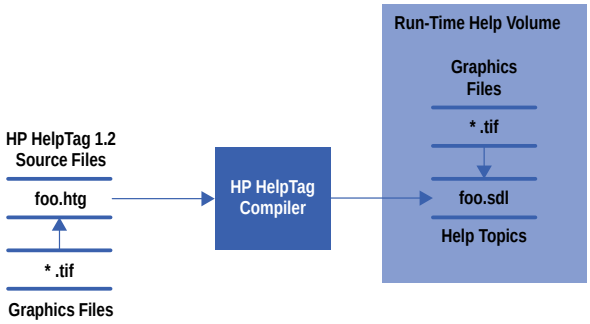
While designing and implementing the help volume compilation process, techniques for improving performance emerged. The objectives were to create a file that supported quick access and a small overall disk footprint. Fig. 9 shows the components involved in the HelpTag compilation process.

For run-time display, a HelpTag source file must be converted into a form usable by the CDE 1.0 help system viewer. This conversion process is often referred to as compiling or translating the HelpTag source file. The output of the conversion process is a single file containing all the information (except the graphics) necessary to display a help topic. The graphics associated with a topic are left in their own files and are referenced from within the CDE 1.0 help system run-time format, the Semantic Delivery Language.

The dthelptag utility, which converts HelpTag source code to SDL, is a shell script driver for the multiple passes of the conversion process. The conversion takes place in three major steps:

1. The HelpTag source is read in and the first step in conversion to SDL is made. During this step side buffers are created to hold references to graphics and hyperlinks outside of the current document and keyword entries that will later be used to enable keyword searches of the document. The keyword entries correspond to an index in a printed book. Forward cross-reference entries may cause this step to be executed twice for complete resolution.

Fig. 9. *The HelpTag compilation process.*



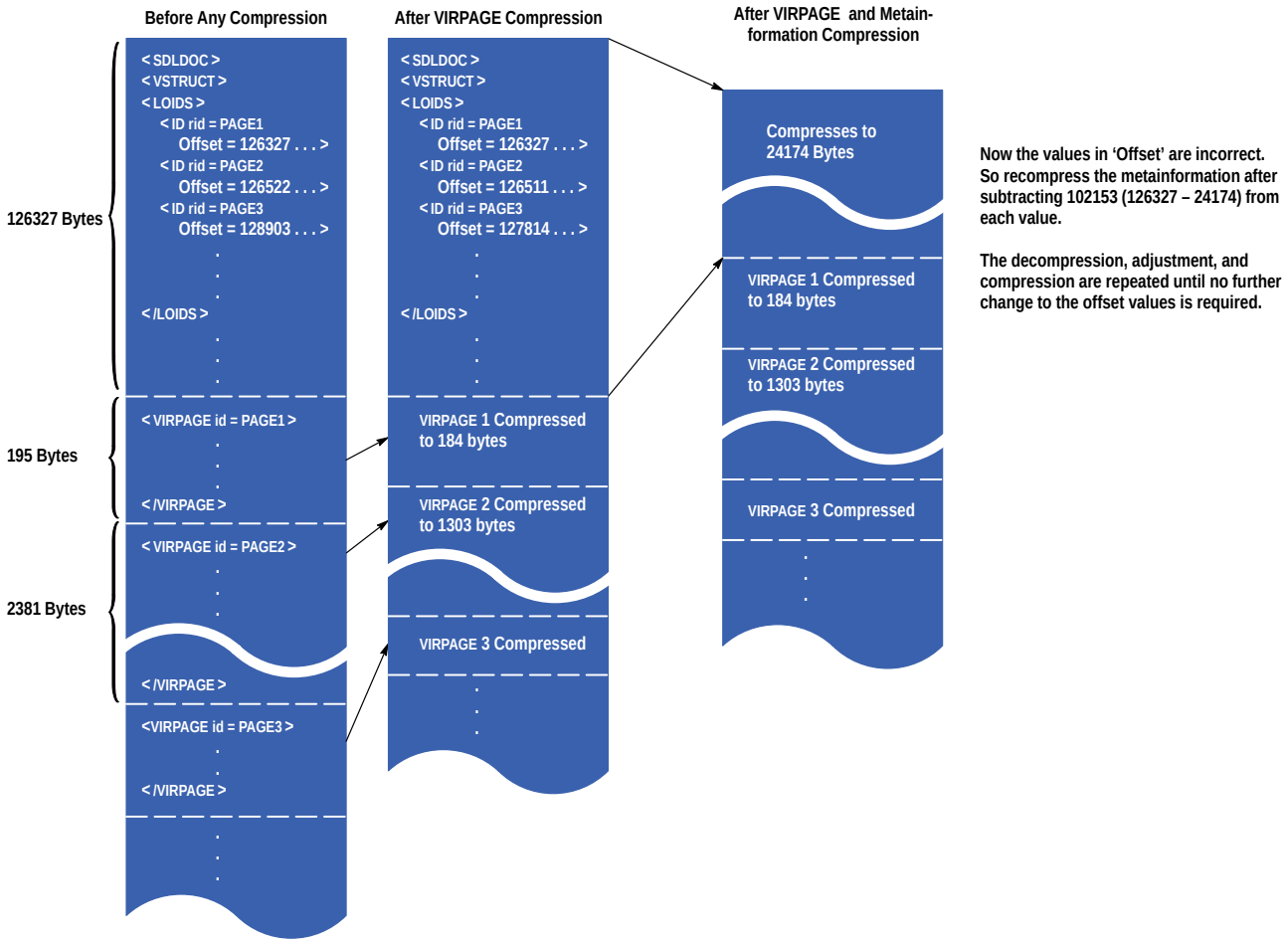
2. The keyword side buffer is sorted using the collation sequence of the locale of the document. Duplicate keyword entries are merged at this time.
3. The SDL output of the first step is reparsed and examined for possible optimizations. Then the optimizations are performed, the keyword list created in step two and the external reference information from step one are merged in, and the SDL file is preprocessed to facilitate fast run-time parsing and display. Finally, the SDL file is compressed and written to disk.

The optimization possibilities mentioned in step three are created when the first pass must make worst-case assumptions about text to follow or be contained in an element when the start-tag for that element is encountered. If the worst-case scenario does not manifest itself, the resulting SDL code will be suboptimal. The equivalent of a peephole optimizer* is used to detect the suboptimal SDL and to replace, where possible, suboptimal SDL with equivalent but simpler constructs.

The compression mentioned in step three uses the UNIX compress(1) utility, which is a modified version of the LZW (Lempel-Ziv and Welch) compression algorithm. The run-time decompression of the help topic is performed via a library version of the LZW algorithm to avoid requiring the creation of an extra process with its attendant time and memory penalties.

To preserve the random-access nature of the help topics within the help volume, the volume is compressed on a per-virtual-page basis (Fig. 10). That is, each virtual page is compressed and prefaced with a single null byte followed by three bytes containing the size of the virtual page in bytes. After compressing the virtual pages, the list of identifiers must be updated to reflect that all virtual pages following the compressed page are now at a new, lower offset into the file. When the run-time help viewer reads a virtual page, the reader looks at the first byte of the virtual page and if it is a null byte, decompresses the virtual page starting four bytes into the page (the null byte and three bytes for length occupy the first four bytes of the page).

Fig. 10. *The compression sequence for an SDL file.*



* In this application, the peephole optimizer reads a small subsection of a document, such as a chapter or paragraph, and works on that portion in isolation from the rest of the document.

Finally, after all the virtual pages have been compressed, the metainformation in the SDL file, including the list of identifiers, is compressed. Since for performance reasons all the metainformation is at the front of the SDL file, compressing that information must necessarily be an iterative process. After compressing the metainformation, the offsets to the virtual pages in the list of identifiers must be updated to reflect that they all are now at new addresses. When the offsets in the list of identifiers have been updated, the metainformation must be recompressed resulting in new values for the offsets in the list of identifiers.

At some point while iterating through this compression and update cycle, the result of compression will be no reduction in size or, worse, an increase in size. If the result is no change, we are done and we can write out the final SDL file. If the result is an increase in size, a padding factor is added after the end of the compressed metainformation and that factor is added to the offsets in the list of identifiers. The iteration then continues, increasing the padding factor on each pass until the size of the compressed metainformation plus all or part of the padding factor stabilizes. The first pass in which the size of the compressed metainformation plus zero or more bytes of the added padding equals the most recently computed offset for the first virtual page terminates the iteration. The compressed metainformation plus as much padding as is necessary is then written to the output SDL file and all the compressed virtual pages follow.

Graphic File Formats. Complaints about the text-only nature of HP VUEhelp 2.0 strongly demonstrated the truth of the adage that "one picture is worth a thousand words." The CDE 1.0 Help System supports the following graphic formats:

- X Bitmaps
- .xwd Files
- .xpm Files
- TIFF 5.0

Graphic Compression. While JPEG compression schemes are common for use with TIFF files, no compression was being used with the X graphical formats. After several complaints from authors about how much space .xwd files require, the help system was modified to find and access compressed files. The author can use the UNIX compress(1) command to compress the graphic files. The help system decompresses the graphic file into a temporary file and then reads the file as usual.

Using compression on X graphic format files can impact access time. For very large graphic images or for a topic that uses many graphics, this impact can be noticeable. The trade-off between speed and disk space is one that the help volume author and application engineer must address. In most cases the best results, both for performance and disk usage, are gained by using JPEG-compressed TIFF images.

Printing

Currently, CDE 1.0 DtHelp renders only text to hard copy devices. The printing solution for the CDE 1.0 help system allows the user to print a comprehensive table of contents and index, complete with page numbers. When an entire help volume is printed, each page is numbered allowing easy cross reference from the table of contents or index. This functionality did not exist in the HP VUE 3.0 help system. It was developed for CDE 1.0.

Localization

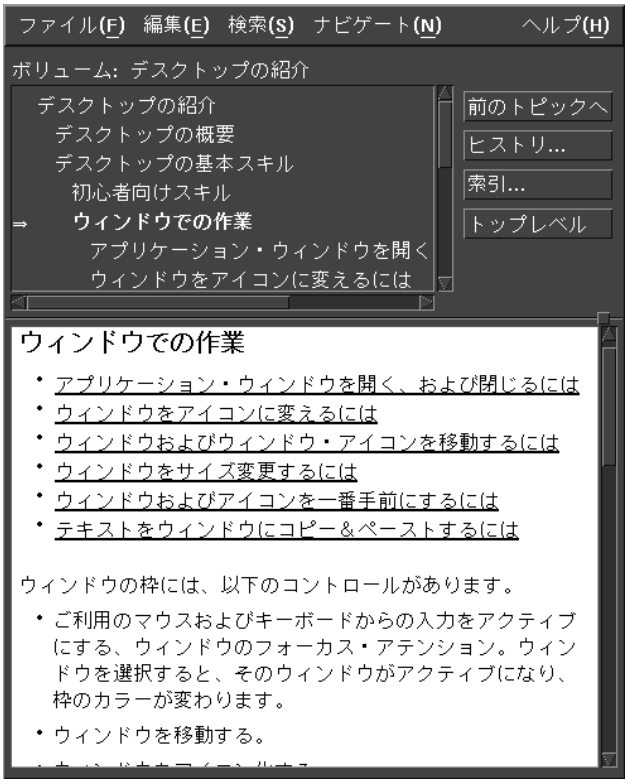
The CDE 1.0 help system supports authoring and displaying of online help in virtually any language. The online help information can be authored and translated in either single-byte or multibyte character sets, and all the components within the developer's kit are multibyte-smart and can parse and display the localized information.

The help widgets use the user's $LANG environment variable to determine what language directory to retrieve the requested help volume from. If $LANG=japanese when the request to display help occurs, the widget code will attempt to open the Japanese localized version of that help volume. If one does not exist, then the default version, which is English, will be used.

When an authored help volume is compiled via HelpTag, the author sets the proper character set option. The character set information is used at run time to determine the proper fonts to use for displaying the localized help text. The HelpTag compiler assumes a default locale ($LANG=C). Currently, because of the complexities involved, only one multibyte character set per volume is supported (e.g., a Japanese-to-Korean dictionary cannot be displayed). Fig. 11 shows a sample of a localized window.

Parsing Multibyte Characters. To make dthelptag work for single and multibyte character sets without constantly checking for the length of the current character, all characters are converted to wide characters (wchar_t) on input. This input conversion is driven by a command line option, a HelpTag entity file, or the current setting of the locale. All internal processing of characters is done on wide characters with those wide characters being converted back to multibyte characters on output. Single-byte character sets are treated just like multibyte character sets in that the conversions in and out always take place.

Fig. 11. *Sample localized help window.*



This scheme of doing all internal processing on wide characters has proven to be a very effective means for making one tool work for all languages. The scheme did require implementation of wide character versions of most of the string functions (e.g., strcpy, strlen), but those functions were all quite straightforward to create.

Localizing User Interface Components. The menus, buttons, labels, and error messages that appear in help dialogs also support full localization to native languages. The help dialogs read these strings from a message catalog named DtHelp.cat. Various localized versions are supported by default and included with the developer's kit product. For languages not supplied, the developer must translate the message catalog /usr/dthelp/nls/C/DtHelp.msg and then use the gencat command to create the needed run-time message catalog file.

Conclusion

Building upon the strong foundation provided by the HP VUE 3.0 help system, the CDE 1.0 help system (DtHelp) has become the standard help system for the desktop of choice for the providers of a majority of UNIX systems across the world. As more companies provide the CDE desktop, this help system will become even more pervasive.

References

1. *Hewlett-Packard Journal*, Vol. 45, no. 2, April 1994.
2. *OSF/Motif Style Guide Release 2.1*, Prentice Hall, 1993.
3. R. Ulichney, *Digital Halftoning*, MIT Press, 1988, Chapter 8.

▶ Go to Article 6
▶ Go to Table of Contents
▶ Go to HP Journal Home Page