BROWN UNIVERSITY
Department of Computer Science
Master's Thesis
CS-89-M12

"A System Model Which Accounts for Previous Experience: A Combined Interface and Help System"

by
Margaret D. Stone

A System Model Which Accounts for Previous Experience:
A Combined Interface and Help System

by

Margaret Dexter Stone

B.S., Southeastern Massachusetts University, 1988

Approved:

Thesis

Submitted in partial fulfillment of the requirements for the Degree of Master of Science in the Department of Computer Science at Brown University

May, 1990

Abstract

A user's ability to generalize his experiences with a computer system to new tasks is directly related to the robustness of the mental model he forms while learning to use the system. If designers don't account for the variety of user goals and preconceptions that will be brought to software when it is first learned, the system model built into the software will be ineffective. This paper proposes as a solution to this problem, a "bootstrapping" interface: an integrated interface and on-line help system which identifies user preconceptions and goals and encourages users to develop a model of the system which coherently incorporates these preconceptions and goals. This paper outlines the design methodology used to develop a prototype of a "bootstrapping" word processor. Nine subjects tested the prototype for usability, successfully completing assigned tasks despite a variety of backgrounds and incomplete instruction on use of the word processor. Insight was gained into user preconceptions in the domain of word processing, and the prototype design was improved.

Problem Statement

It is widely believed that users form a mental model of a system as they learn to use it (diSessa, 1986; Lewis, 1986; Norman, 1986; Owen, 1986; Riley, 1986). This model is used to predict the behavior of the system as the user attempts to carry out new tasks. The user's ability to generalize his experiences to new tasks is directly related to the robustness of his mental model of the software.

Designers of software systems are faced with the task of developing a system model which projects a consistent, coherent image from which the user will construct his mental model. Designers have interpreted this task as meaning they must create a system which projects a single system model, the one which is most appropriate to the software and the average user. Norman (1986) and Owen (1986) have pointed to the learning period of system use as the stage in which a mental model is the most useful. A designer's model, no matter how coherent, will not assist the user in learning the system if it differs widely from the user's preconceived ideas about the software and the software domain.

Different users prefer different software systems for performing similar tasks, use the same software toward different ends, and approach a new software package with personal sets of preconceptions developed in years of experience with the world. All of these factors must be considered in designing a system model, or the model will be ineffective in the most crucial period of software use.

The field of human-computer interaction will benefit from the development of a design methodology which uses the study of user preconceptions and their manifestations to develop interfaces designed specifically to aid users in the early stages of software use. Such an interface could be referred to as a "bootstrapping" interface, as the uninitiated user

will be able to "pull himself up by the bootstraps" in the learning phase of software use, requiring little or no formal instruction to successfully learn the software.

This paper outlines a design methodology for the development of a "bootstrapping" interface, and describes the early phases of the development of a "bootstrapping" word processor. The chosen user audience is college students who are taking a composition course. College students taking a composition course are an interesting audience for several reasons. First, they are trying to learn the difficult task of writing. They are neither experts nor novices in this domain, but they have preconceptions about their task, as they had writing assignments in secondary school. They have preconceptions about the tools used for the task, whether they have used pen and paper, the typewriter, a word processor, or some combination or subset of these tools. Word processing is an interesting domain because the word processor is the most crucial software tool used by the chosen audience. The majority of students in college today are required to write with a word processor, despite the possibility of little or no previous experience with the tool and little time to learn. There are many word processors available for study and for example of interface and functionality choices.

There have been numerous studies on the effects of computer writing tools on user writing attitudes, writing ability, and revision strategies (Baer, 1988; Bridwell, Sirc, and Brooke, 1985; Hansen and Haas, 1988; Harris, 1985; Hawisher, 1987; Lutz, 1987). These studies conclude that while users enjoy using the word processor and feel that it improves their writing, there is little or no change in writing ability and revision strategy. The users studied generally adhered to old strategies and adopted them to the software they were using. These findings point to writing as a challenging domain for the development of a "bootstrapping" interface, as users can be expected to hold fast to their individual preconceptions.

Thesis

Design Models for a "Bootstrapping" Interface

In order to develop a "bootstrapping" interface, a designer must be able to answer the following questions:

1. What are the possible backgrounds that users might have?
2. What models do these backgrounds bring to the software domain?
3. Can each model be integrated into the design, and if not, what is the best possible model to use in interacting with the user who has this background and model?
4. How are the backgrounds and models recognizable in task execution?

The answers to these questions provide the designer with the raw information from which the software model, the user model, and a system-user interaction strategy can be forged. In examining user backgrounds, the designer learns terminologies and methodologies that are familiar to users of different backgrounds. By studying the models of users with these backgrounds, the designer gains insight into user expectations and strategies. This insight is used to develop an interface which projects multiple system models, each appropriate to a different user background. The software must have the ability to recognize a user's background in order to interact with him in a suitable manner. Help strategies are designed to reinforce the user's model and teach task-achievement methods which are in concordance with this model. If it is impossible for the designer to accommodate a user's background, there must be a scheme for guiding the user to a different, but similar, model.

The developed system projects multiple system models and has strategies for recognizing the user's mental model of the system, and for tutoring users with different models of the system. In order to have an interface / help system which classifies and helps the user according to preconceptions and goals, a set of user models and tutoring strategies must be maintained in the system, in addition to a profile of the current user.

The only way for the system to non-intrusively discover the current user's mental model is through his use of the system. It is therefore appropriate to model the user as his method of interaction with the system, and assume that this is an approximation of his mental model of the system. If the user is a novice with the software, his interaction with the system is an approximation of his preconceptions, background, and expectations.

Precisely how the "user's interaction" is to be represented depends on the software domain. Generally, the user's interaction can be represented as a set of smaller interactions, each denoting a task which can be achieved in the domain. These interactions are characterized by the task to be achieved, the method of task achievement, general strategies which affect the achievement of the task, and the language used to describe and conceptualize the task.

A Methodology for the Development of a "Bootstrapping" Interface

The four questions above can be answered only through study of the domain and the target population. Potential task-achievement methods and their associated terminology and strategies must be abstracted from the study of the domain. As much of this user interaction data as possible must be incorporated into the design, without "cluttering" the design, and without causing obvious conflicts. Naturally, the issues of "clutter" and "conflicts" are subjective. The designer has to use his best judgement, and then allow testing to illuminate problem areas. The following method for developing a

"bootstrapping" interface will guide the designer in discovering appropriate user-interaction data and successfully incorporating these data into a design.

1. Study the domain, focusing on the previous tools and methods used to perform tasks in the domain, language used to describe tasks in the domain, and general strategies employed to solve problems in the domain.
2. Develop prototypical situations on the target computer. Central to each situation is a common task to be performed on the computer, for which several methods of task execution have been implemented, each corresponding to a different user model.
3. Execute usability testing on the prototypical situation, testing several users with a variety of background experiences.
4. Determine if the implemented methods and their corresponding models were appropriate.
5. Record any methods attempted by the users which were not provided in the prototype.
6. Resolve ambiguities in the user's methods, if possible, and add the user's methods to the interface and help system, including recognition and tutoring schemes. If this is not possible, add a recognition scheme, and assign the user an approximate model for tutoring.
7. Develop a fully functional system based on the prototypes and repeat steps 3 through 6 to test and improve this system.
8. Execute advanced testing on the final system, studying the effectiveness of the system after long periods of use and comparing the system to other tools in the domain.

The Prototype for a "Bootstrapping" Word Processor

For this project, the prototype of a "bootstrapping" word processor was developed and tested, fulfilling steps one through six of the methodology described above. The

purpose of this project was to demonstrate that "bootstrapping" software is a practical, and potentially superior, alternative to software which forces the user to adopt the designer's domain model. The prototype, therefore, conforms to the concepts described above, but does not incorporate pioneer methods in the style of on-line help provided.

The Domain Study

The purpose of the domain study is to learn about previous tools and methods used to perform tasks in the domain, language used to describe tasks in the domain, and general strategies employed to solve problems in the domain. This information is used to develop an interface which projects multiple system models, and a help system which recognizes a user by his method of interaction with the word processor, and interacts with the user accordingly.

In the domain of writing, the most common tools are pen or pencil and paper, the typewriter, and the word processor. These tools are used individually or in combination. There is a large variety of word processors and typewriters, offering a variety of task-achievement procedures, language, and strategies for the achievement of similar tasks. When correcting drafts on paper, an assortment of symbols are used to indicate changes to be made to texts. Some of these symbols are universal, others are unique to an individual or group. A multitude of task-achievement data can be gleaned from the study of these areas.

To study task-achievement methods in writing, tasks were divided into two writing categories: editing and formatting, and three word processing categories: filing, selecting, and moving through a document. For each task in the writing categories, four or more methods were listed: the Macintosh word processor method, the "other" word processor method (meaning popular word processors, such as Microsoft Word for the IBM PC), the

typewriter method, and the pen / pencil and paper method. For example, for the editing task of moving a block of text, the Macintosh word processor method is to select, cut, move the insertion point, and paste. The "other" word processor method is a similar cut and paste operation, a copy / delete and paste operation, a move operation, or the user could delete and retype with any word processor. There is no typewriter method, and with pen and paper, the "user" might draw a circle around the block to be moved, and an arrow to its new location (a typewriter user might also use this method, before retyping). For the tasks in the word processing categories, mostly word processors were considered, although icons and commands representing events that might occur with pen and paper or a typewriter (such as turning a page) were also considered.

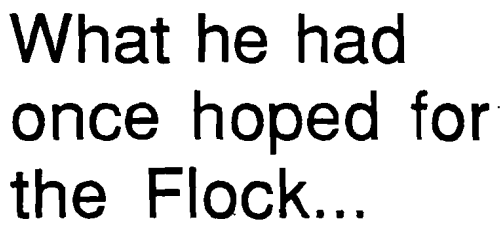
A questionnaire (Appendix A) was designed to discover user backgrounds and the associated language used in describing tasks. The questionnaire consists of two sections: one multiple-choice and fill-in-the-blank section to discover the user's background (users were divided into the following backgrounds: pen / pencil and papers users, pen / pencil and paper and typewriter users, typewriter users, pen / pencil and paper and word processor users, word processor users, and typewriter and word processor users), and the other consisting of before and after "pictures" of text-editing and text-formatting situations, under which the user was instructed to describe what had happened between the pictures. The questionnaire was completed by 179 college and high-school students, most of whom had little experience with word processors.

Seven of the twenty-five before / after questions were studied for language used to describe word-processing tasks (these were the most relevant questions in light of the reduced functionality of the prototype as compared with a fully-functional word processor). Although there were some trends which linked background with language, this occurred mainly with the computer users. The computer users and non-computer users alike

generally used individual and unique language. For each question, two issues were considered: the verb used to describe the transformation, and the noun used to describe the block of text. Following is a brief summary of the data gathered from the seven questions studied.

Question six consists of the following graphic:

What he had once hoped for the Flock...



For the verb, or transformational word, one hundred and forty-four responses included the word enlarge, large, or big. Thirty-two responses included "word processor" words such as style, bold, font, and size (the majority of these were pen / pencil and paper and word processor users, some of the "bold" and "size" responses were from non-computer users). The most commonly used nouns were type, letters, sentence, print, or words (one hundred and thirty-one responses).

Question nine consists of the following graphic:

What he had ocne... What he had once...

One hundred and thirty-three responses mentioned spelling or correction (about a third of these were from students who had used a word processor previously, so no trends linking background and language were discovered here). Seven responses used the word "switch." For nouns, thirty-one used "word," six used "letter," and three used "mistake." The remaining responses did not include a noun, mentioned "once" explicitly, or used the "before" column or the "after" column as the noun.

Question eleven consists of the following graphic:

What he had once hoped for the Flock...

<u>What he had once hoped for the Flock...</u>

One hundred and seventy-one responses indicated that the fragment had been underlined. Of the ninety-nine responses which included a noun, eighty-eight used one of the following nouns: phrase, fragment, sentence, word, passage, or statement. Only four responses included the noun "text," two of these responses were from word processor users, and two were from pen / pencil and paper and word processor users.

Question twenty-four consists of the following graphic:

What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid.

What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid.

Although many of the responses described only the size change, one hundred and forty-seven described the change in typeface. Of those, the verb was usually "change," and therefore not very interesting. The noun was usually print (fifty-four responses), type (twenty-nine responses), or the response described the new typeface as "lighter" (twenty responses), italics (eighteen responses), or boldface (seven responses). Eleven responses used the word font. Of these, four were strict word-processor users (out of six total strict word-processor users), five were pen / pencil and paper and word processor users (out of thirty-nine total), one was a typewriter and word processor user (the only one), and the last was a pen / pencil and paper and typewriter user who had previously used a word processor. Many of the responses were phrased badly, indicating that the student had no language available for describing a typeface change. The following are some examples:

"form of writing changed"

"the words in the second form were lightened in color, and were also fitted together to make a smaller paragraph"

"the first one is regular writing the second they put in fancy writing"

"it's in a different way of print"

Question twenty-six consists of the following graphic:

What he had once hoped for the Flock, he now got for only himself;	What he had once hoped for the Flock, he now gained for himself alone;

One hundred and two responses used the nouns "word" or "sentence." Five others used one of the following nouns: statement, excerpt, or paragraph. The remaining responses described the transformation using the exact words which were changed, "before" column and "after" column, or used no noun at all. Forty-three responses used the verb "change," thirty-seven used "add," and sixteen used one of the following verbs: correct, replace, or reword. Nineteen responses indicated that the wording had been improved. One response used the verb "edit" (a pen / pencil and paper and word processor user).

Question twenty-seven consists of the following graphic:

What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid. Jonathan Seagull discovered that boredom and fear and anger are the reasons that a gull's life is so short, and with these gone from his thought, he lived a long fine life indeed.	Jonathan Seagull discovered that boredom and fear and anger are the reasons that a gull's life is so short, and with these gone from his thought, he lived a long fine life indeed. What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid.

One hundred and twenty-two responses indicated switching; Sixty-nine used the verbs "switch" or "swap," and fifty-three used one of the following verbs: flip, rearrange, revise, or order. These responses were from a homogeneous mixture of students who had and had not used a word processor previously. Two responses used the verb "move," these were both pen / pencil and paper and word processor users. Three responses used "cut and paste," these were either pen / pencil and paper and word processor users, or strict word processor users. Of the one hundred and forty-five responses which included a noun, one hundred and thirty-five used "paragraph," five used "sentence," four used "section," and one used "stanza."

Question twenty-eight consists of the following graphic:

What he had once hoped
for the Flock, he now
gained for himself alone;
he learned to fly, and
was not sorry for the
price that he paid.

What he had once hoped
for the Flock, he now
gained for himself alone;

Ninety-two responses contained a verb describing the transformation. Of these, forty used one of the following verbs, indicating that part of the text had been deleted: deleted, dropped, cut, eliminated, omitted, excluded, removed. One response used the word "erase," and twenty-seven wrote that some of the text had been left out, taken out, or taken away. Seventeen used the following verbs which indicated revision: shortened, condensed, revised. One hundred and fourteen responses included a noun. Fifty-three of these used the noun "paragraph," and twenty-five used the noun "sentence." Twenty-seven used one of the following nouns: line, statement, phrase, word, or passage. Five used the noun "text," all of whom were strict word processor users, or pen / pencil and paper and word processor users. Four used either "information" or "material."

The Prototype Design

The functionality of a word processor was divided into five categories: editing, formatting, filing, selecting, and viewing the text. Two of these categories, editing and formatting, are familiar to users of all backgrounds (even if, to a pen / pencil and paper user, formatting means drawing a line under a title). Filing, viewing, and selecting are tasks that are familiar primarily to word processor users. Methods were prototyped for the word processor users, and an attempt was made to accommodate users who will be unfamiliar not only with the procedures to be followed, but more importantly with the concepts of filing, viewing, and selecting. Experimentation is necessary to discover what methods the non-word processor users find most intuitive.

In the prototype, editing commands on the level of cut and paste are provided (cut, copy, paste, and keyboard editing). No advanced editing commands, such as global search and replace and spell-checking, were prototyped. In order to accommodate the different user backgrounds considered, a variety of commands were prototyped that aren't available in standard Macintosh word processors. The following procedures were developed for the "other" word processor users, the typewriter users, and the pen / pencil and paper users: a move procedure, a duplicate procedure, a replace procedure, and a delete procedure. Thus there are several methods available for performing each goal-level task, such as moving, replacing, and revising.

As with editing, a sharply reduced functionality is provided for formatting in the prototype. The prototype has the capability to change character size, typeface, and style; leaving margins, tabs, line spacing, paragraphs, alignment, and special graphics capabilities unimplemented. To change the typeface, style, or size (all of which require the same method), four methods are provided. The first is a standard Macintosh method: select the desired style, size, or font from the appropriate menu (usually there is a Font

menu and a Style menu, in this word processor there is a Set Type Style menu), then type the text. The other standard Macintosh method is to select the text, then choose the style, size, or font. If the user has not demonstrated the meta-skill (or strategy) of selecting text before choosing an operation, he will be guided through the operation, allowing both methods described above.

Selecting is one of the categories that, conceptually as well as methodologically, will be novel to users unfamiliar with a word processor. For word processor users, standard Macintosh selection is provided. This includes pressing and dragging with the mouse, double-clicking (to select a word), and shift-clicking (to extend a selection from the previous selection or from the insertion point). New users invariably have difficulty learning to select text (personal experience), sometimes because they don't understand to select text before choosing an operation, and sometimes because they aren't yet accustomed to the mouse. A second method is provided for the non-word processor users, but is strictly experimental as users will have few preconceptions about a task which they have never been required to perform. This second method is a Select menu, with items which allow selection of the whole document, a word, a sentence, a paragraph, or "some text."

Although viewing will be familiar to users with all backgrounds, only word processor users will be comfortable with the notion of a screen of text (as opposed to a page or line). Three methods were prototyped for viewing different parts of the text: the standard Macintosh scroll bar, the View menu, and the keyboard equivalents for the View menu. The View menu consists of four commands: Previous Screen, Next Screen, Previous Line, and Next Line. The prototype does not provide the ability to see what text will constitute a page when printed, so no page-viewing commands were prototyped. Also provided is the standard scrolling which occurs on the Macintosh when you type above or below the screen, and when you select above or below the screen. The scroll bar was

provided for Macintosh users, and the View menu and keyboard equivalents were provided for "other" word processor users, who might be more familiar with "Page up," "Screen up," and similar commands. In order to determine what would be preferable to non-computer users, more testing has to occur.

It has been observed that when learning the Macintosh (personal experience), users frequently "lose" their text (what they have typed is no longer visible on the screen), through accidental use of the scroll bar, pressing the return key, closing the document, or moving the window. Carroll and Carrithers found that creating a training system, in which certain error states are unreachable, resulted in faster learning and improved comprehension (Carroll and Carrithers, 1984). An attempt was made to alleviate the window-moving problem by reducing the functionality of the word processor. The prototype does not allow movement or resizing of the window by the user. It is possible that the other problems could be alleviated by including a help choice "What if I lose my text?" This requires more study of beginner users and the questions they ask when they lose their text, and what they respond to in order to find their text.

Filing tasks were virtually ignored for this prototype; only one method of saving, opening, closing, and getting a new window was implemented: the standard Macintosh method. A fully-functional word processor was designed which provided multiple methods of accomplishing these tasks, but the fully functional word processor was not prototyped. Formatting and editing tasks are more interesting because they are familiar to users with a variety of backgrounds, and because of limited time and a limited number of test subjects, emphasis had to be placed on these areas in the prototype.

The prototype screen consists of three windows (one with a scroll bar), a menu bar, and a palette of command icons (see Figures 1 and 2). The first window is the text-editing

window, into which the user may enter and edit text. This window has a standard Macintosh scroll bar which consists of a bar with a "thumb" indicating the position of the current screen in relation to the entire document. The scroll bar allows continuous scrolling (by pressing in the arrows at the top and bottom), next and previous line scrolling (by clicking in the arrows at the top and bottom), next and previous screen scrolling (by clicking in the gray area above or below the thumb), and thumb-dragging to reach a screen in the document immediately. The text-editing window title is currently "UntitledX," or the name of the document if it has been saved to disk (named). In a fully-functional version of the prototype, the window title would contain information about the last time the document was written to disk.

The Clipboard window is a temporary holding place for text which has been Cut, Copied, Duplicated, or Moved. The Clipboard window appears whenever a new item is written to it during one of these procedures, or if the user explicitly requests to see the Clipboard by choosing "Show Clipboard" from the Edit menu. Once displayed, the Clipboard remains displayed until the user chooses "Hide Clipboard" from the Edit menu. The Clipboard window title changes in accordance with its contents. If text has been Cut to the Clipboard, the Clipboard window title is "Clipboard: Cut Text." The title changes similarly for copied, duplicated, and moved text. The decision to display the Clipboard after each Clipboard-affecting operation was made as an attempt to attenuate new user misconceptions about the Clipboard. The Clipboard concept is a word processing standard which has no meaning in the domain of writing. In his study of student modeling of word processors, R.L. Upchurch found that users, even after a semester of writing on the word processor, do not have robust models of the Clipboard and the operations that can be performed on it (personal communication, July, 1989).

The Instructions window is a small window which appears at the top of the screen, accompanied by a menu bar flash and a system beep, whenever the user chooses a command which consists of several steps. The OK and Cancel buttons appear at the top of the control panel and remain visible while the Instructions window is visible. The Instructions window disappears after the user has completed the task for which he has been given instructions. The user signifies completion by pressing in either the OK or Cancel buttons. For example, the Move command requires that text be selected, then a place chosen for the new text. If Move is chosen with text selected, the Instructions window instructs the user to click where the text should be moved, then to click in the OK button. If Move is chosen without text selected, the Instructions window asks the user to first select text, then click OK, then click where the text should be, and click OK. Clicking in the Cancel button cancels the operation.

With any software package, task accomplishment consists of selecting an object and choosing an operation to perform on the object. In a word processor, objects are blocks of text, and operations are editing, formatting, moving, selecting, and filing commands. In the "bootstrapping" prototype, there are two ways to select an object; using standard Macintosh text-editing techniques, and the Select menu. There are also two ways to choose operations. Most operations can be chosen from the palette, or from a menu. Viewing options can be chosen from the menu, or using the scroll bar. Language issues shaped the wording of the Select menu, and the decision to have a graphical palette. The Select menu provides selection of words, sentences, and paragraphs, because these are the units into which questionnaire responses decomposed blocks of text (the noun responses), regardless of background. Although certain verbs were used exclusively by word processor users, most language was individual. A graphical palette does not attempt to describe graphical operations with short bursts of language, but with icons representing the operations to be accomplished. Care was taken to use icons which would be universally familiar, and many

word processor users were polled during the design of the palette to determine the most preferred and recognizable icons.

One general goal-achievement strategy in the domain of word processing was identified; the preference of when to choose the operation in relation to the choice of object. Some users are more comfortable with choosing the object first (the standard Macintosh method), others prefer to choose the operation first. Both methods are implemented in the prototype, the operation-before-object method being facilitated by the Instructions window (see Figure 2). The non-computer users were considered in implementing the operation-before-object method, as it has the potential to help them adjust to the unfamiliar notion of selecting text.

In order to help users based on their preferred methods and strategies, a record must be maintained of the current user and the methods and strategies employed while using the word processor. To teach this user task-achievement methods which are in concordance with his model, information must be available which links his methods and strategies to methods and strategies that can be used to solve problems he hasn't yet encountered.

For the prototype, a list of word processor "events" (WP events) was created (See Table 1 for the list of WP events). Each WP event is a unit that can be used in the construction of one or more task-achievement methods. A list of upper-level tasks was created, and for each upper-level task, task achievement methods, composed of WP events, were listed (see Table 2 for the list of task-achievement methods). For example, there are eight methods of replacing text. One of these methods is to press the delete key, and then retype. Another method is to select text, then choose Replace (from the Edit menu or the palette), then type the new text, and press OK.

When the user completes a task, the method of task achievement is recorded in the current user help profile. For the simple purposes of the prototype, user preferences were considered to be the number of times a given task achievement method was used, compared with the total times that task was accomplished. In the future it would be wiser to have task achievement methods weighted and losing weight over time, so that as the user learns, he isn't plagued by his early conceptions.

The user can request help by choosing from the Help menu, or by clicking in the Help button in the palette. Choosing Help invokes a series of dialog boxes: modal windows which require a response before the user can continue using the software. The dialog boxes force the user to specify what task he needs help accomplishing, then a dialog box appears, describing a method of task accomplishment. Cutting edge help techniques were not considered for the prototype, as the purpose of the prototype is to demonstrate the viability of "bootstrapping" software.

When the user specifies the task he needs help accomplishing, the help procedure checks to see what tasks the user has accomplished previously. Tasks, and in some cases specific task-achievement methods, are linked into logical units by similarities in achievement methods, results, and purpose. For example, if the user requests help switching passages, and he hasn't moved text previously, the help procedure will check to see if he's duplicated. Duplication has the same achievement method as moving, and a similar purpose, but different results.

The task-achievement help message has five parts: reminder, first, next, finally, and results. The reminder part of the message reminds the user of the task he's accomplished which is similar, and explains, in general terms, what the difference is between the task previously accomplished, and the task about which help was requested. If the user hasn't

performed a task even remotely similar to the task about which help was requested, the reminder will simply say "You can use the following method to [accomplish task]." The first, next, and finally parts of the message describe the steps for accomplishing the task. Some methods don't require this many steps for explanation. The results part of the message describes what will happen after "finally." This provides the user with the opportunity to determine if he is reading the right help screen for his task, and the tools to determine if he's followed the procedure correctly, once he tries to follow the help advice.

In two instances, there will be ambiguity about command meaning, depending on the user's general strategy of object / operation selection timing. These instances are when the user is Pasting (does he expect the contents of the Clipboard to be pasted at the insertion point when he chooses Paste, or does he expect to choose Paste and then click where he wants the contents of the Clipboard) and when the user is choosing a type font, style, or size without previously selecting text (does he expect to begin typing in the new style, or does he expect to be able to select text after choosing). The prototype maintains this information about the current user. The user is assumed to have the operation-before-object strategy unless he selects text and chooses a palette or menu operation. The first design of the prototype did not consider selecting text before choosing a font, size, or style as contributing toward this strategy information, as selected text can be a side effect of other operations (e.g. Copy), and therefore the user's intentions were ambiguous. If the user has the operation-before-object strategy, whenever Paste or a font, size, or style is chosen, the Instructions window appears, and then user must select the object and press the OK button. If the user has the object-before-operation strategy, the Instructions window does not appear on typeface or Paste choices.

In order to study user interactions with prototyped situations, a recording facility must be available to the designer and invisible to the user. Because of time limitations, a

very simple recording facility was built into the prototype. The prototype records every WP event, and saves this list of events, as well as the user help profile, to disk. To supplement this meager recording scheme, observations were made and recorded by hand as the users tested the prototype.

The majority of Macintosh software is event-driven. The prototype for the "bootstrapping" word processor is no exception. The main procedure is a loop which reads events from the Event Queue and responds accordingly. After every system event, a parser-like procedure is called to determine if a WP event occurred, updating the help profile if necessary. The prototype was developed using THINK Pascal version 2.01. The event-response code, filing code, and utilities code was based on a text editor released as an example of Macintosh programming with THINK LightSpeed Pascal version 1.0, although there are few similarities between the THINK text editor and the "bootstrapping" word processor prototype. The code for the "bootstrapping" word processor prototype is provided as Appendix C.

Usability Testing

Usability testing allows a designer to examine the viability of his design, exposing flaws which can be changed rapidly. In usability testing, the designer prototypes aspects of his design which he wants to test, then develops tasks which are realistic to the software domain, and will force the questionable portions of the design to be exercised extensively by his target audience. Several members of the target audience execute the task, then complete a questionnaire designed to draw out their perceptions of the software. In testing the "bootstrapping" word processor prototype, perceived flaws were corrected or circumvented between testing situations in order to maximize the value of testing.

In testing the "bootstrapping" word processor prototype, a test was designed to force the user to perform high-level editing and formatting tasks. Performance of the editing and formatting tasks could lead to implicit viewing and selecting tasks, but these were not included explicitly as they are not writing goals.

The user was seated in front of a Macintosh IIcx with the word processor running and a short paper typed into the text-editing window. The user was instructed to read a one and one-half page description of the word processor. This description includes a paragraph explaining that the software is being tested, not the user, a paragraph describing the mouse, three paragraphs describing how to choose from the menu bar and the palette, and two additional paragraphs on asking for help (all testing materials are included as Appendix B). After reading the description, the user was given four tasks to perform, one at a time. The tasks were presented in a manner similar to the language questionnaire, as objects to be transformed. Language was not emphasized in presenting tasks to users, so as not to bias them toward choosing particular commands. The first task was to underline text, the second task was to switch two paragraphs, the third task was to emphasize a line, and the fourth task was to revise a badly worded phrase.

As the users performed the tasks, facial expressions, comments, and mouse movements were recorded by the observer. After performing the tasks, the users completed a questionnaire. The questionnaire asked for the user's computer background, a description of the tasks performed (again gathering language), a description of unexpected word processor actions, and the user's general impressions of the word processor. The most meaningful data, in light of the design goals, was the observation of unfulfilled expectations, and subsequent description thereof on the questionnaire.

The first two users tested, Shannon and Alex, had not used a word processor before. Both had used the Macintosh a little bit for "class assignments." For the first task, both users immediately clicked in the underline box in the palette, invoking the Instructions window (as no text had been selected). Both were considerably confused about the message, because it instructed them to "click where you want to begin typing in the new style, or select text to be changed to the new style." Both clicked, then pressed the OK button, and were surprised when the text did not change to the chosen style. After several tries, the observer pointed out the Select menu. Both tried the Select menu, and had no further problems with task 1. Shannon seemed upset that the text was still selected after it was underlined, but she clicked in the window to de-select, and then was happy.

For the second task, both users were stuck until the observer reminded them of the help function. Shannon chose Help->Editing->Switching passages, then effectively followed the instructions. For the remaining two tasks, Shannon chose Help before attempting the task, then followed the instructions successfully (or botched the instructions and asked for repair help and followed those instructions successfully). While using the word processor, Shannon accidentally selected text. Instead of being surprised by the inverted text (many new users are surprised and upset by selected text), she realized that she'd discovered a new method, and thereafter used the click-and-drag method of text selection. Alex believed for a short time that clicking would select for him (probably from his experience with the Select menu), then apparently remembered that he needed to choose from the Select menu first. He performed the second task improperly, thinking that he only had to switch the words at the beginning of the paragraph. He therefore selected the words, then used the Delete rectangle in the palette, and retyped the words (the observer did not point out his error to him).

For the third task, Alex chose Select Some Text from the Select menu, but anticipated the instructions, clicking at the start, then the end of the text he wanted to select (instead of clicking at the start, then the OK button, after which he would be instructed to click at the end). He eventually got it right, and clicked in the Shadow rectangle in the palette. When the selected text changed to Shadow, he said that it looked awful. Later, when he de-selected the text, he said "That's the way I wanted it to look."

For the fourth task, Alex chose Replace from the palette, but he had text selected from the style change, so the replace didn't work. He cancelled, and used the Delete command, then retyped.

For all of the tasks, both Shannon and Alex used the menu bar only to Select text. For all other commands, they used the palette. Shannon requested help, and was instructed to use the palette because she used the palette to underline text. Alex picked out commands from the palette independently of the help, and never chose a command he didn't intend.

On the questionnaire, Shannon indicated that the word processor did nothing she didn't expect. Alex didn't understand why he had to choose from the Select menu after choosing all his other commands from the palette. Both wrote that they could use the word processor in the "impressions" section of the questionnaire. Alex wrote "Even though I struggled for the first few minutes, I picked up on it, and it became easy to work with. This was in a short period of time, so I don't think it would take very long to be able to use this effectively." Shannon liked the help, and wrote "The directions given by the computer help you to know what to do."

After Shannon and Alex, the design was improved in two ways. First, the style change message was reworded to be more explicit about when to select, and when to click.

Second, the selecting "after-effect" was removed. If text is still selected after an operation (such as Copy, or a style change) it is de-selected immediately by the word processor.

The third subject had used an IBM PS2 extensively, and had used the Macintosh some for word processing. Robert completed the four tasks quickly, probably because of his experience with selecting and his facility with the mouse. For the first task, he first clicked in the underline rectangle in the palette, invoking the Instructions window. After reading the message, he clicked at the beginning, and then the end, of the text to be underlined, then clicked in the OK button. After a moment of confusion, he dragged over the text, then chose Underline, and clicked in the OK button. He said that he knew to drag from "previous experience."

For the second task, Robert clicked in the Move rectangle in the palette. When the Instructions window appeared, he selected the text he wanted to move, then clicked where he wanted it to go, anticipating the next set of instructions. He then re-read the instructions, and re-selected the text, following the Move procedure effectively.

For the third task, Robert clicked in the Shadow rectangle in the palette, then selected the text, then clicked in the OK button. For the fourth task, he couldn't decide between the strike-out rectangle (Delete) and the scissors rectangle (Cut) in the palette. He finally decided on the strike-out, and used that command, as well as the Replace command (from the palette) to reword the passage. Like Shannon and Alex, Robert used the palette exclusively for his editing and formatting tasks. Robert wrote that the word processor did nothing he didn't expect, and that it was "simple, completely laid out and direct. Very forward in approach." No changes were made to the prototype at this point.

The fourth subject, Bill F., had used a VAX 11/780, an IBM mainframe, and an IBM PC, but had never used a Macintosh. For the first task, he had an almost identical experience to Shannon and Alex, but preferred the Select Some Text option, where they chose Select a Word or Select a Sentence. For the second task, Bill first selected the text (using the Select menu), then chose Cut from the Edit menu. He then selected and chose Move from the Edit menu, thus losing the Cut text. He realized immediately that the Clipboard had been overwritten (the Clipboard was visible after the Cut operation), and repaired his mistake by retyping the text . Bill informed the observer that he had used a system in which "Move" indicated the same operation as Macintosh's "Paste."

For the third and fourth tasks, Bill easily selected text and chose operations that he wanted to perform on the selected text. Unlike Alex, Shannon, and Robert, Bill preferred the menu bar to the palette, once he noticed it (the observer indicated the menu bar to Bill when he was stuck over selecting text). He admitted that the menu bar was described in the word processor description, but he forgot about it. Bill wrote that he "expected some problems selecting different options. Once I became familiar with how the system worked, it was more obvious and pretty easy." He also wrote that the user should be precautioned about what operations will change the contents of the Clipboard. No changes were made to the prototype at this point.

The fifth subject, James, is an expert computer and Macintosh user, and had no difficulty executing any of the tasks. James selected using the mouse, and chose all of his commands from the palette. He didn't like that the word processor "helped" him when he performed the first task (underlining), because he already knew to select before choosing a style. He wrote that the word processor was "fun to use." Four other subjects were tested the same day, so the prototype could not be changed at this time.

The sixth subject, Jeff, is a beginner on an IBM PC compatible word processor, and had never used any other computer. For the first task, Jeff misunderstood the instructions, and spent some effort changing a paragraph from plain text to boldface. Before he realized his mistake, Jeff tried the help function, which told him to use the palette (the default when the user hasn't actually done anything). Jeff tried to click in the "Change to" rectangle of the palette, which is an ornament and not a command. After realizing his mistake, Jeff chose underline from the Set Type Style menu, selected the text to be underlined, and clicked in the OK button. Surprisingly, Jeff had no difficulty selecting with the mouse. When questioned by the observer, Jeff said that it was "common sense."

For the second task, Jeff chose Move from the Edit menu. His mouse-selecting procedure didn't work well for text longer than one line, because he didn't realize he could drag beyond one line (Shannon did the same thing, and operated on lines one at a time). Because of his difficulties with the mouse, he chose Help (from the palette), and asked for help switching passages. After reading the Help, he successfully switched the passages. The Help told him to use the Move rectangle in the palette (because he hadn't done a similar operation previously), so he used the palette.

For the third task, Jeff had no trouble emphasizing the phrase, choosing Bold from the Set Type Style menu. For the fourth task, he tried the Clip->Text rectangle in the palette, realized his mistake, and tried the Text->Clip rectangle. He then cancelled that (the Instructions message made him realize he wanted to Cut, not Copy), and chose the scissors, successfully rewording the passage. In writing about his expectations, Jeff wrote "the computer did what I told it to do." His impression was "I believe that after working with it for a short period, I would be comfortable using it. It seems to take a lot of the thinking out of how to write something, in other words, it makes organizing your work easier."

Jeff clearly preferred text-oriented commands to the graphical palette, either because of a natural inclination, or because of his previous word processor experience. It was unfortunate that the help directed him to use the palette, although he had few difficulties adjusting to the palette. He hypothesized later that the palette was supposed to be used for editing commands.

The seventh subject, Bill B., had previously used an IBM PC and an Amiga for word processing, but had never used a Macintosh. For the first task, Bill clicked in the Underline rectangle in the palette, read the instructions, selected the text, then clicked elsewhere in the window (de-selecting the text) and clicked in the OK button. He followed the procedure a second time but forgot to click in the OK button. He chose Help, realized his mistake (saying "duh"), and finished the task.

For the second task, Bill used the Move rectangle in the palette to successfully switch the passages. For the third task, Bill used the Bold rectangle in the palette to successfully emphasize the sentences. For the fourth task, Bill chose Help / Edit / Reword, read the screen, then chose Help / Edit / Delete, and successfully used Delete and Move to reword the passage. Unfortunately, Bill didn't realize that he could type on the keyboard, so he spent a considerable amount of effort getting the spacing perfect in his revision. In the expectations portion of the questionnaire, Bill criticized the word processor because of these problems moving words (the Move is not implemented "intelligently," leaving spaces where it should take them). His impressions were that "it worked pretty well, and really didn't give me any more trouble than most. I was a bit unsure about one of the symbols (the delete symbol) on the palette, but it really was no big problem."

The eighth subject, Brian, had previously used an IBM PC Compatible, a Macintosh, and an Apple IIe for a variety of purposes, but didn't consider himself an expert on any of these systems. Nevertheless, he had no problems with the tasks, using the drag method of selecting text, preferring the palette to the menus, and using the Cut and Paste commands rather than the Move and Replace commands. Like James, he didn't like the Instructions message that appeared after a formatting command, telling him to select or click after he had already selected. His general impressions were: "I thought it was good. However, I think the palette needs to be clarified (functions listed in the instructions). Of course, this may have been in the 'Help' section of the program but I didn't use it."

The ninth subject, Sarah, is a frequent MacWrite user, but has never used any other computer, and has had little contact with other software for the Macintosh. She, like Brian and James, had no difficulty accomplishing the tasks. Sarah used the menu bar for the first two tasks (preferring, like Brian, the Cut and Paste commands to the unfamiliar Move command), but she used the palette for the third task, and used only the keyboard and mouse for her revision of the sentence in the fourth task (her revision was the most extensive of all the users tested). Sarah, like Brian and James, wrote that she did not expect "to have to hit O.K. after changing word styles."

After James, Jeff, Bill, Brian, and Sarah, three changes were made to the prototype. Because the select "side-effects" were removed, there was no longer any reasonable ambiguity about the user's intentions when selecting text and choosing a text style. Therefore, selecting text prior to choosing a text style is considered valid information about the user's general strategy of object and operation selection timing. Thus, the Instructions window no longer appears if the user selects text before choosing a text style.

Two intrusive dialog boxes were added. Several of the users tested tried to choose commands while the Instructions window was still up (i.e. they forgot to click OK or Cancel). One message was added instructing the user to click in the OK button or the Cancel button before choosing the new operation. A second message was added because of Jeff's confusion over changing type styles in the palette. The message appears if the user attempts to click in the "Change to" ornament, instructing the user that it is not a command. This is not intended as a fix to the problem, but more of a bypass. Since everything else in the palette is a command, the "Change To" ornament should be removed.

Conclusions

Despite a variety of backgrounds, none of the users required more than a half hour to complete the four tasks, and most used much less time. Each of the nine users developed his own strategy for solving problems with the word processor, and no strategy or method was preferred by a majority of the users.

Users who had previously used the Macintosh, but were neither novices nor experts, seemed most comfortable with the familiar commands. The Macintosh expert explored the new commands provided by the prototype. The novices preferred the palette, whereas the previous computer users tended to use the menu bar. The users familiar with the Macintosh chose the object before the operation, the other users tended to choose the operation before the object.

From this data, it can be concluded that the design provided options which were comfortable to a variety of users, without being "cluttered." Conflicts arose only when the help system provided the user with advice which forced him to act against his natural inclinations. Only one user, Brian, used the word processor in a manner that seemed to be contrary to his preconceptions. Brian used the palette versions of the standard Macintosh

commands and criticized that the meaning of the palette options wasn't clear. Why didn't Brian use the menu bar, where the commands are named in the same way that they are in all Macintosh word processors? It is possible that the palette is distracting, causing users to forget the less flamboyant menu bar. Another user, Bill F., didn't notice the menu bar, even though it was described in the word processor description. It is also possible that Brian assumed that different commands would be provided in the palette and the menu bar (a natural assumption), and the palette looked like it provided the Cut, Copy, and Paste commands.

All users were able to choose the operations they wanted, but the non-computer users had difficulty selecting objects, as was expected. Can the "bootstrapping" methodology help the user who has no preconceptions about objects, operations or strategies? The non-computer users were able to select once the Select menu was revealed to them, but the notion of selecting was not as natural to them as it was to previous computer users. The "bootstrapping" methodology needs to be better developed to effectively treat areas in which preconceptions and previous experience may be weak or missing.

The help portion of the prototype demonstrated that confirmation of a user's preconceptions assists the user in developing a consistent mental model of the software, through both positive and negative examples. Bill B. was pleased to discover that the procedure he was following was the correct one, when he asked for help completing a task with which he was having difficulty. He indicated that he had missed something obvious, but to another user, his method would not have been obvious at all. Jeff had the unfortunate experience of receiving advice which was counter-intuitive, and subsequently developed an inconsistent model of the prototype (editing commands must be chosen from the palette, even though they seem to be contained in the Edit menu, but formatting commands may be chosen from the Set Type Style menu).

A design methodology which accounts for user preconceptions and tutors problem-solving based on preconceptions appears to be a viable alternative to single-model interface design, at this early stage of development. The users tested developed their own methods for achieving tasks, and all successfully achieved the assigned tasks. More work must be done to improve the methodology itself, in order to handle areas in which preconceptions are weak or missing, and to tutor users who haven't used the system much, but request help.

Future Directions for This Research

Improvements to the Prototype

Before a final design is developed and tested, more usability testing has to occur. A different version of the prototype should be created and tested with the following features:

- Method and task refinements

Some of the tasks should be altered, and the methods for accomplishing the tasks should be refined. For example, There should be several different Replace Text methods, instead of one: Replace some text, Replace a lot of text, Replace while you're typing, Replace text that is far from the insertion point. The user may backspace frequently while typing, to replace small errors, but this is not an appropriate method for replacing a large block of text. The help system will tutor this method, however, because the user employs it frequently, even though the user may wish to learn a method appropriate to a different kind of replacing.

- "Intelligent" Move and Replace commands

Move and Replace currently move and replace the selected text, instead of checking to see if the selection includes spaces, tabs, and carriage returns which should be part of the move or replace operation.

- Palette selection

For the users who use the palette exclusively, a select option should be included on the palette. Research should be done to determine the exact nature of this option.

- Help changes

In the help procedures, the user's growth should be considered. A button could be provided to teach an alternate method to the method the help system thinks the user should learn. Task achievement methods should be weighted and losing weight over time, so that as the user learns, he isn't plagued by his early conceptions.

The help windows should include options not only about the commands available with the word processor, but also commands for getting started, for typing text so that it appears as desired, and for managing invisible characters.

Help currently uses as a last resort (if the user has tried nothing relevant), how the user chose Help. If the user chose from the menu bar, he is instructed to follow a procedure using commands from the menu bar. Similarly, if he clicked in the Help button, he receives help suggesting he try commands from the palette. Help should be changed to look for a general trend in palette / menu preference instead of looking at how the user chose Help, as evidenced by Jeff's unfortunate conclusion that he should use the palette for editing tasks.

Help should be improved to consider incomplete procedures, especially incomplete procedures before choosing help, to emphasize to the user that the incomplete procedure, or a repaired version of the incomplete procedure, can lead to a successful completion of the task. Bill B. was relieved to learn that his method of changing the type style, which he had tried once without success, was the proper method. One way to find incomplete procedures is to record what happens before the press of the Cancel button.

The Final Design

The final design will contain the interface, user-computer interaction method, and user model developed through the usability study of the user audience on the prototypes. With more time and resources, experimental help techniques could be substituted for the method currently in use. There are many possibilities for the help system which could improve user comprehension and satisfaction with the help provided.

Research indicates that instructional events usually fail because the instructor has made an assumption about the user's knowledge of prerequisite information (Hill, 1989), or because users are unable to formulate a follow-up question (Moore, 1989). A solution to these problems might be a user-computer interaction strategy in which dialogues are composed of brief explanations created from hypertext or offering hypertext-like choices. Hypertext offers the user the possibility of definition of unfamiliar terms, or description of unfamiliar procedures which appear in a help dialogue.

Carroll et al. recommend using task-oriented language in instructional design in order to coordinate learner's goals with procedure descriptions (Carroll, Smith-Kerker, Ford, Mazur-Rimetz, 1987). Carroll and Mack et al. have found that keeping suggestions incomplete improves user competence in learning and task performance (Carroll, Mack, Lewis, Grischkowsky, Robertson, 1985). They hypothesize that if suggestions are

incomplete, following instructions does not replace the user's original task-oriented goal. Instead of modal help windows, incomplete suggestions could be provided in non-disappearing windows, so the user could refer to the help as he attempted to carry out the task.

In order to study user interactions with prototyped situations, a recording facility superior to the current method must be available to the designer and invisible to the user. The recording facility will save on disk keystroke data for situation sessions, available for "play back" on the software. Necessary features, such as "pause," will be identified and provided.

References

Baer, V.E.H. (1988). Computers as composition tools: A case study of student attitudes. Journal of Computer-Based Instruction, 15(4), 144-148.

Bridwell, L., Sirc, G., & Brooke, R. (1985). Revising and computing: Case studies of student writers. In S.W. Freedman (Ed.), The acquisition of written language: Response and revision (pp. 172-193). Norwood, New Jersey: Ablex.

Carroll, J.M, & Carrithers, C. (1984). Training wheels in a user interface. Communications of the ACM, 27, 800-806.

Carroll, J.M., Mack, R.L., Lewis, C.H., Grischkowsky, N.L., & Robertson, S.R. (1985). Exploring exploring a word processor. Human-Computer Interaction, 1, 283-307.

Carroll, J.M., Smith-Kerker, P.L., Ford, J.R., & Mazur-Rimetz, S.A. (1987). The minimal manual. Human-Computer Interaction, 3, 123-153.

diSessa, A.A. (1986). Models of computation. In D.A. Norman and S.W. Draper (Eds.), User centered system design (pp. 201-218). Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers.

Hansen, W.J., & Haas, C. (1988). Reading and writing with computers: A framework for explaining differences in performance. Communications of the ACM, 31(9), 1080-1089.

Harris, J. (1985). Student writers and word processing: A preliminary evaluation. College Composition and Communication, 36(3), 323-330.

Hawisher, G.E. (1987). The effects of word processing on the revision strategies of college freshmen. Research in the Teaching of English, 21(2), 145-159.

Hill, W.C. (1989). How some advice fails. Human Factors in Computing Systems Proceedings (pp. 85-90).

Lewis, C. (1986). Understanding what's happening in system interactions. In D.A. Norman and S.W. Draper (Eds.), User centered system design (pp. 171-185). Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers.

Lutz, J.A. (1987). A study of professional and experienced writers revising and editing at the computer and with pen and paper. Research in the Teaching of English, 21(4), 398-421.

Moore, J.D. (1989). Responding to "Huh?": Answering vaguely articulated follow-up questions. Human Factors in Computing Systems Proceedings (pp. 91-96).

Norman, D.A. (1986). Cognitive engineering. In D.A. Norman and S.W. Draper (Eds.), User centered system design (pp. 67-85). Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers.

Owen, D. (1986). Naive theories of computation. In D.A. Norman and S.W. Draper (Eds.), User centered system design (pp. 187-200). Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers.

Riley, M.S. (1986). User understanding. In D.A. Norman and S.W. Draper (Eds.), User centered system design (pp. 157-169). Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers.

FIGURES

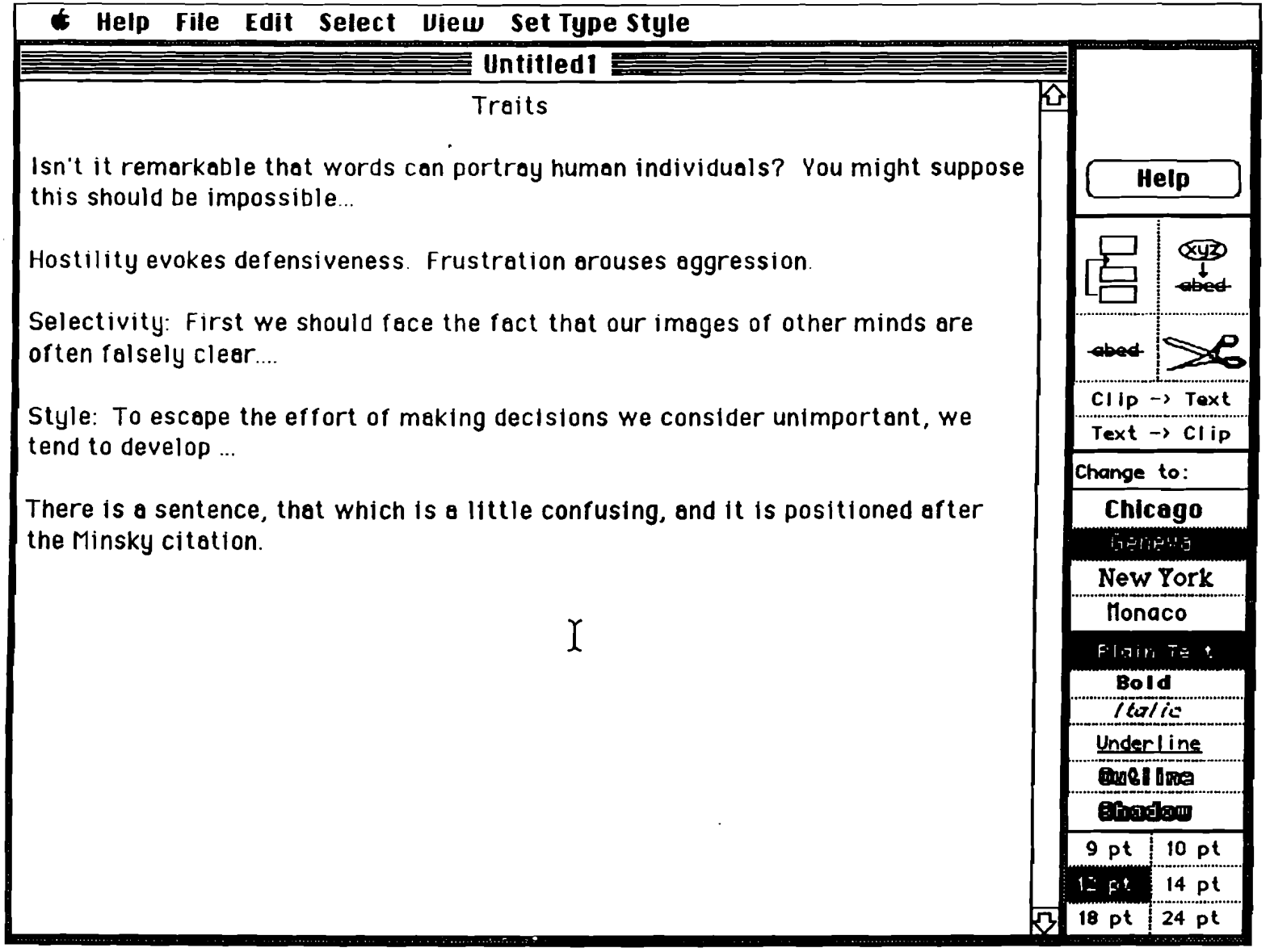


Figure 1: The Word Processor as it Appears at the Beginning of the Testing Task

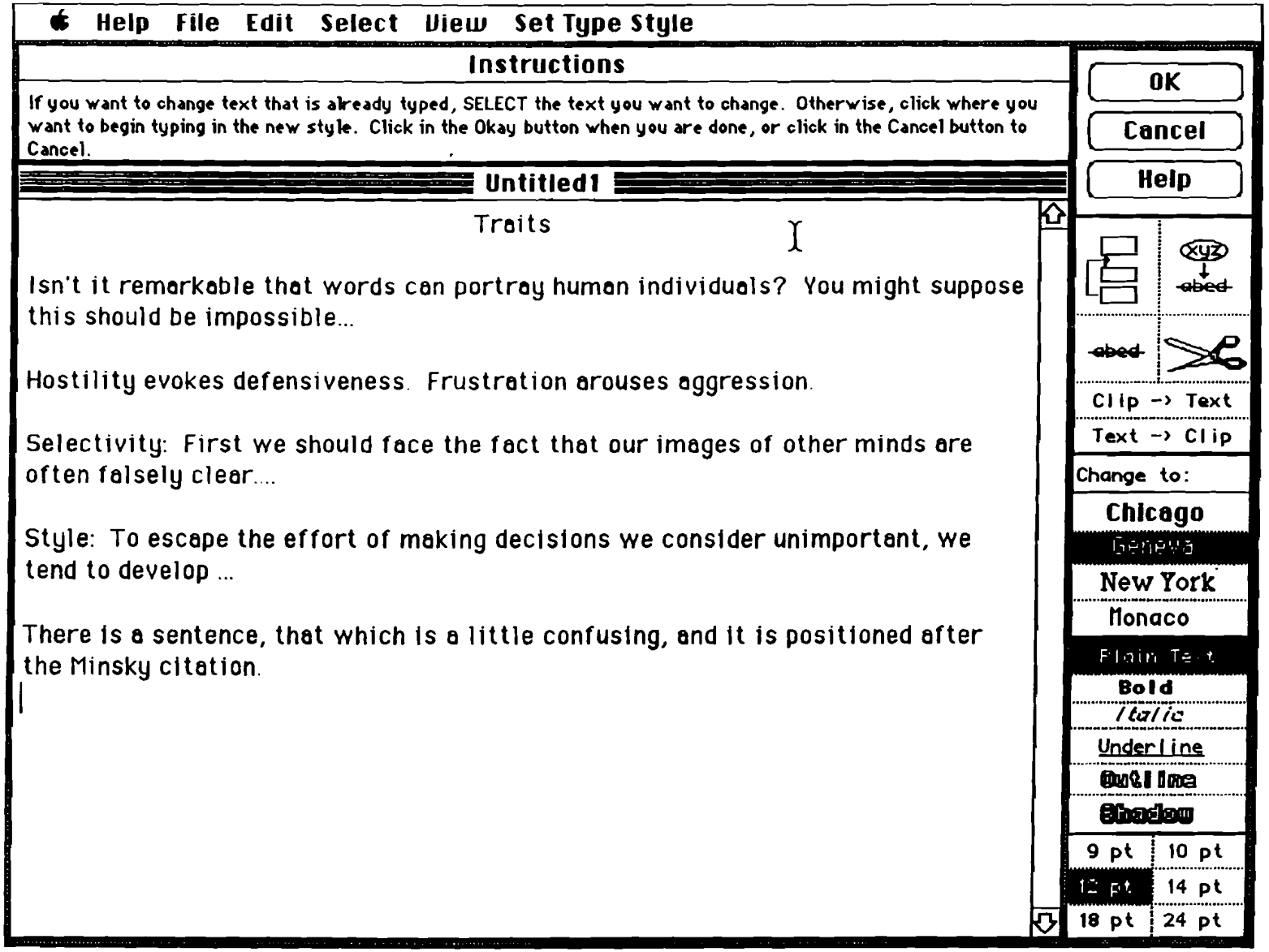


Figure 2: A Style is Chosen From the Palette Without Text Selected

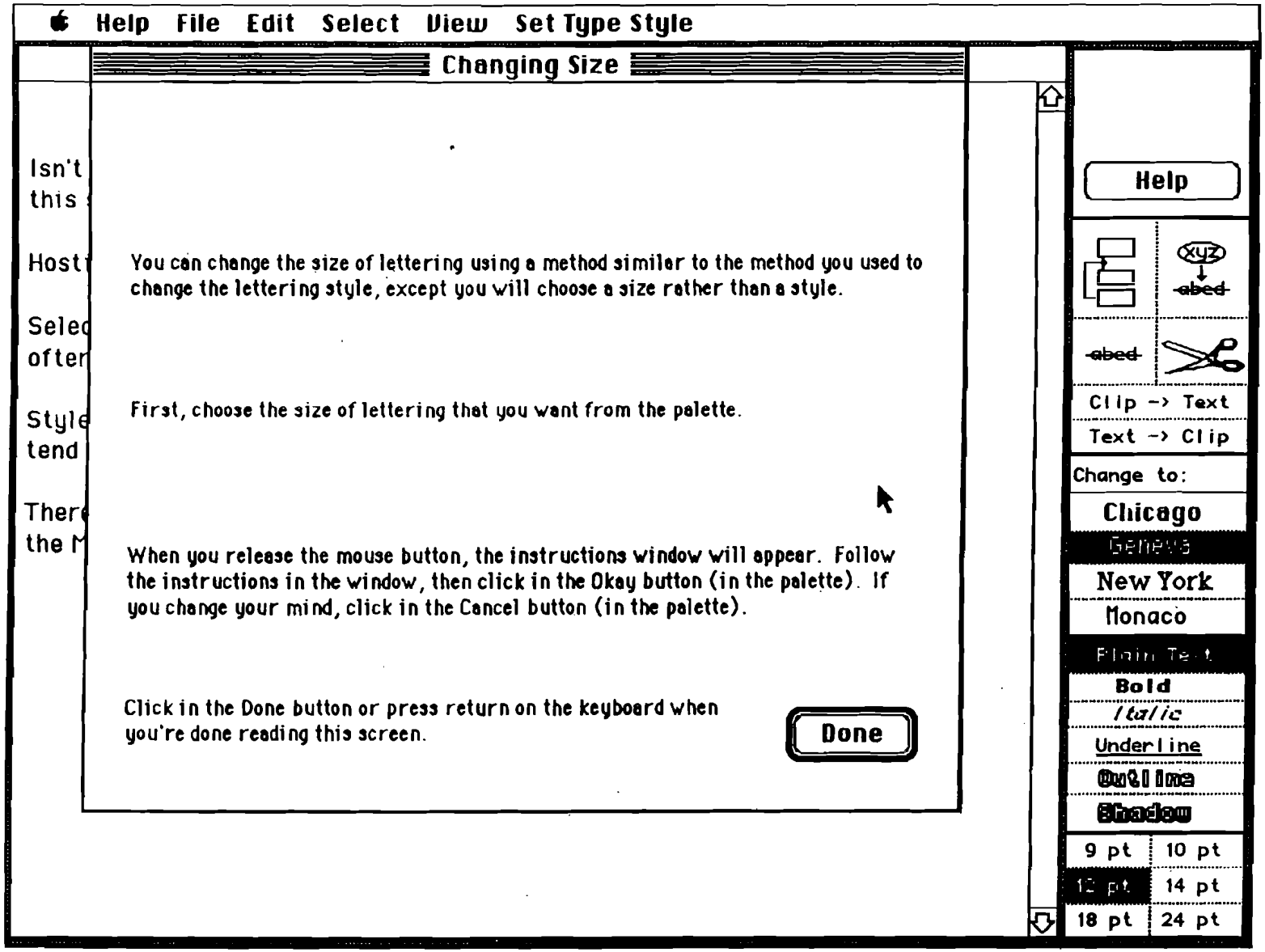


Figure 3: After Help:Formatting:Changing Size Has Been Chosen

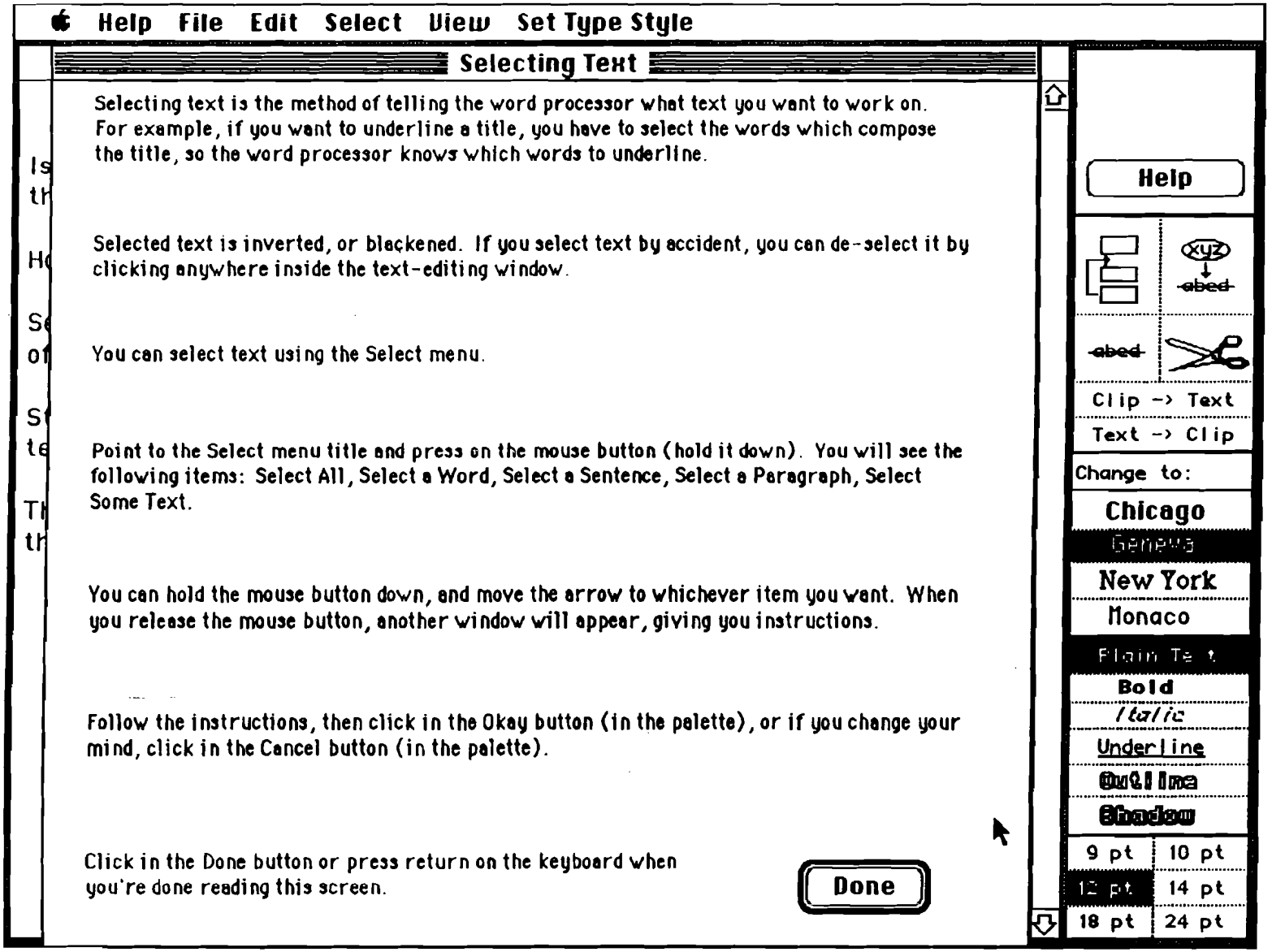


Figure 4: After Help:Selecting is Chosen, User Has Not Previously Selected

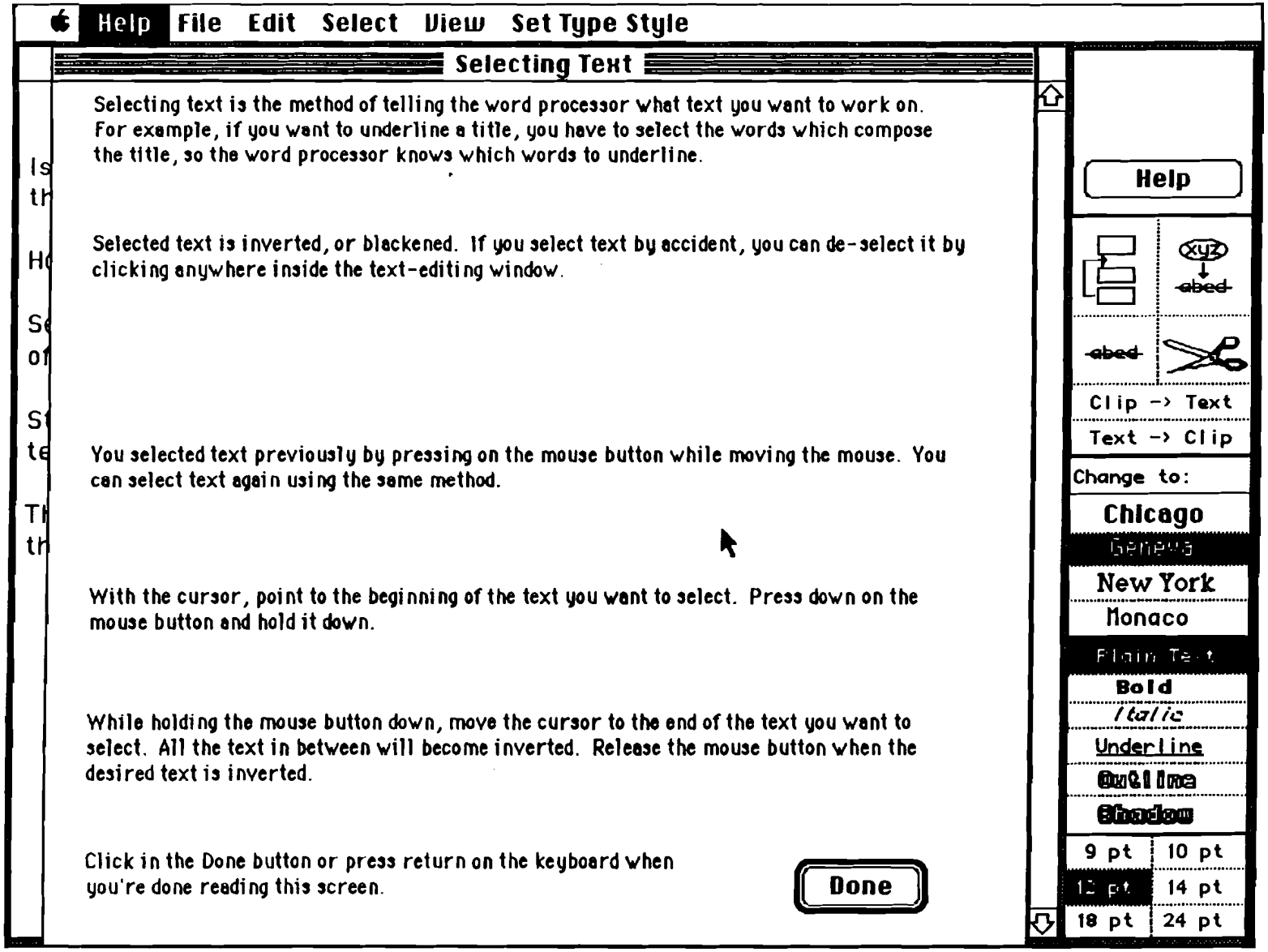


Figure 5: After Help:Selecting is Chosen, User Has Previously Selected

TABLES

WP Event Type	From	Means
nullEvt	-	no event took place
Select	menu	chose Select from menu
	mouse	press&drag or double-click
	key	shift key to extend selection
ClickDnArrow	-	clicked in scroll bar down arrow
ClickUpArrow	-	clicked in scroll bar up arrow
ClickDnArrowRepeat	-	repeatedly clicked in scroll bar dn arrow
ClickUpArrowRepeat	-	repeatedly clicked in scroll bar up arrow
PressDnArrow	-	pressed in scroll bar down arrow
PressUpArrow	-	pressed in scroll bar up arrow
DragThumb	-	press in thumb area of scroll bar
ScreenP	menu	chose Previous Screen from View menu
	key	use cmd-key equivalent to menu
	mouse	clicked in ScreenP area of scroll bar
ScreenN	menu	chose Next Screen from View menu
	key	use cmd-key equivalent to menu
	mouse	clicked in ScreenN area of scroll bar
LineP	menu	chose Previous Line from the View menu
	key	use cmd-key equivalent to menu
LineN	menu	chose Next Line from the View menu
	key	use cmd-key equivalent to menu
PressOK	-	press in OK button
PressCancel	-	press in Cancel button
SizeSet	menu	w/out text selected, chose size from menu
	palette	w/out text selected, chose size from pal.
SizeChange	menu	with text selected, chose size from menu
	palette	with text selected, chose size from pal.
FontSet	menu	w/out text selected, chose font from menu
	palette	w/out text selected, chose font from pal.
FontChange	menu	with text selected, chose font from menu
	palette	with text selected, chose font from pal.
StyleSet	menu	w/out text selected,chose style from menu
	palette	w/out text selected, chose style from pal.
StyleChange	menu	with text selected, chose style from menu
	palette	with text selected, chose style from pal.
Replace	menu	chose Replace from the Edit menu
	palette	chose Replace from the palette
Delete	menu	chose Delete from the Edit menu
	palette	chose Delete from the palette
Move	menu	chose Move from the Edit menu
	palette	chose Move from the palette
Duplicate	-	chose Duplicate from the Edit menu
Copy	menu	chose Copy from the Edit menu
	key	use cmd-key equivalent to menu
	palette	chose Copy from the palette
Cut	menu	chose Cut from the Edit menu
	key	use cmd-key equivalent to menu
	palette	chose Cut from the palette
Paste	menu	chose Paste from the Edit menu
	key	use cmd-key equivalent to menu
	palette	chose Paste from the palette
UserType	-	typed a character key
DeleteType	-	pressed the delete key
MoveIP	-	clicked in the text edit window to move IP

Table 1: Word Processor Event Types

Goal	Method
Delete Text	(1) Put IP (insertion point) at end, press the delete key
	(2) Select, press the delete key
	(3) Select, choose Delete
	(4) Choose Delete, select
Cut Text	(1) Select, choose Cut
	(2) Choose Cut, select
Paste Text	(1) Choose Paste
	(2) Novice Paste (Instructions window)
Copy Text	(1) Select, choose Copy
	(2) Choose Copy, select
Duplicate Text	(1) Select, choose Duplicate
	(2) Choose Duplicate, select
	(3) TOC (text on the clipboard), Move IP (moveip, type, etc), Paste (1)
	(4) TOC, Move IP, Paste (2)
Replace Text	(1) Delete (1) and retype
	(2) Delete (2) and retype
	(3) Delete (3) and retype
	(4) Delete (4) and retype
	(5) TOC:Cut and retype
	(6) Select and retype
	(7) Select and choose Replace, then type, etc
	(8) Choose Replace, then select, etc
Move Text	(1) TOC:Cut, move IP, paste (1)
	(2) TOC:Cut, move IP, paste (2)
	(3) TOC:Copy, delete (2), Move IP, paste (1)
	(4) TOC:Copy, delete (2), Move IP, paste (2)
	(5) TOC:Copy, delete (3), Move IP, paste (1)
	(6) TOC:Copy, delete (3), Move IP, paste (2)
	(7) Select and choose Move
	(8) Choose Move and select, etc.
View	(1) ScreenP, ScreenN
	(2) ClickUpArrow, ClickDnArrow
	(3) ClickUpArrowRepeat, ClickDnArrowRepeat
	(4) PressUpArrow, PressDnArrow
	(5) DragThumb
	(6) LineP, LineN
Change Text	(1) Select and choose Size
	(2) Select and choose Font
	(3) Select and choose Style
	(4) Choose Size and type
	(5) Choose Font and type
	(6) Choose Style and type
	(7) Novice choose Size, select or click (Instructions)
	(8) Novice choose Font, select or click (Instructions)
	(9) Novice choose Style, select or click (Instructions)

Table 2: Methods for Task-Achievement

APPENDIX A

Questionnaire

Questions 1,2, and 3 pertain to how you write most of your assignments, letters, etc.

1. Approximately how many drafts do you write?

2. What do you use to write? (check one)

- ☐ Only pen / pencil and paper
- ☐ Pen / pencil and paper and a typewriter
- ☐ Only a typewriter
- ☐ Pen / pencil and paper and a word processor
- ☐ Only word processor
- ☐ Typewriter and a word processor
- ☐ Other _____

3. Briefly describe your method (process) of writing:

_____

_____

How long have you been writing this way?

Answer questions 4 and 5 only if you have used a word processor.

4. What word processors have you used? (check as many as apply)

- ☐ Macintosh - MacWrite
- ☐ Macintosh - MS Word
- ☐ Macintosh - other — What one? _____
- ☐ IBM PC - MS Word
- ☐ IBM PC - HBJ Writer
- ☐ IBM PC - other — What one? _____
- ☐ VAX - EDT
- ☐ VAX - other — What one? _____
- ☐ Other

What computer? _____

What word processor? _____

5. What word processor do you use most often? (check one)

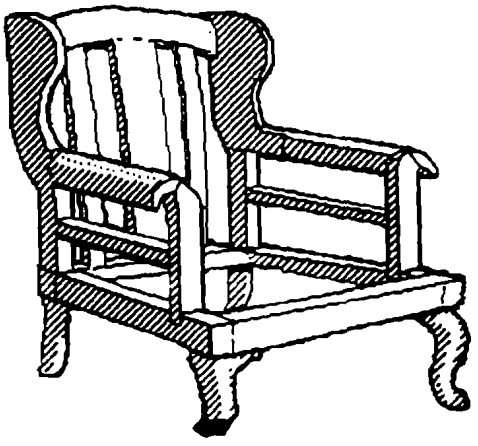
- ☐ Macintosh - MacWrite
- ☐ Macintosh - MS Word
- ☐ Macintosh - other — What one? _____
- ☐ IBM PC - MS Word
- ☐ IBM PC - HBJ Writer
- ☐ IBM PC - other — What one? _____
- ☐ VAX - EDT
- ☐ VAX - other — What one? _____
- ☐ Other

What computer? _____

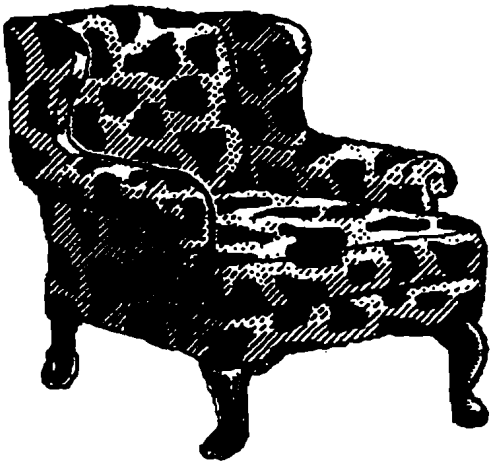
What word processor? _____

The remaining questions (6 - 31) all have the same format. There will be two pictures, one in the "Before" column, and one in the "After" column. Below the pictures there will be lines on which you should write what was done to make the "Before" picture look like the "After" picture. For example, look at the two pictures below:

BEFORE



AFTER



If this were part of the questionnaire, I might write: "The chair has been upholstered."

Now consider the following example:

BEFORE

If there is one thing I can't stand...

AFTER

If there was one thing I can't stand...

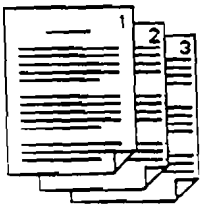
If this were part of the questionnaire, I might write: "The word 'is' has been replaced." I might also write "the sentence has been changed, incorrectly, from present to past tense." Although this is correct, I do not want grammatical comments. I prefer comments of the first kind, describing what has changed in the "Before" picture to make it look like the "After" picture.

6.

What he had once hoped for the Flock...

What he had once hoped for the Flock...

7.



8.

What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid. Jonathan Seagull discovered that boredom and fear and anger are the reasons that a gull's life is so short, and with these gone from his thought, he lived a long time life indeed.

What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid.

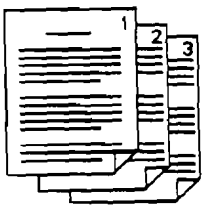
Jonathan Seagull discovered that boredom and fear and anger are the reasons that a gull's life is so short, and with these gone from his thought, he lived a long time life indeed.

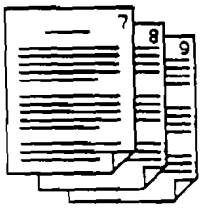
9.

What he had once...

What he had once...

10.



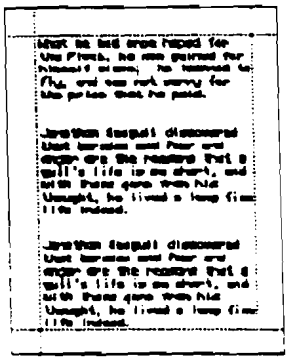


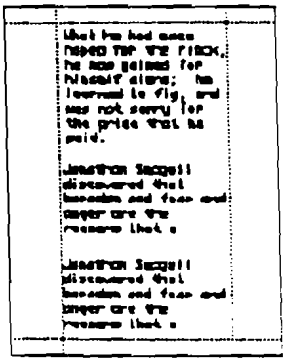
11.

What he had once hoped for the Flock...

<u>What he had once hoped for the Flock...</u>

12.





13.

What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid.

Jonathan Seagull discovered that boredom and fear and anger are the reasons that a gull's life is so short, and with these gone from his thought, he lived a long fine life indeed.

What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid. Jonathan Seagull discovered that boredom and fear and anger are the reasons that a gull's life is so short, and with these gone from his thought, he lived a long fine life indeed.

14.

What he had hoped for the Flock...

What he had once hoped for the Flock...

15.

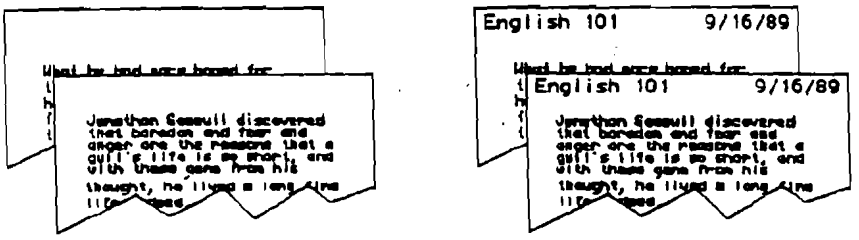
What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid.

Jonathan Seagull discovered that boredom and fear and anger are the reasons that a gull's life is so short, and with these gone from his thought, he lived a long fine life indeed.

What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid.

Jonathan Seagull discovered that boredom and fear and anger are the reasons that a gull's life is so short, and with these gone from his thought, he lived a long fine life indeed.

16.



17.

What he had once hoped for the Flock, he now gained for himself alone...

What he had once hoped for the Flock, he now gained for himself alone...

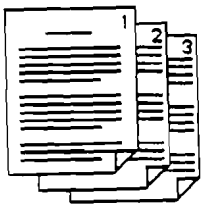
18.

What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid.

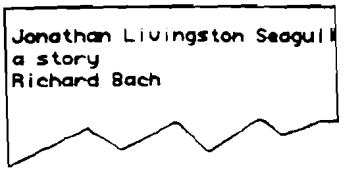
What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid.

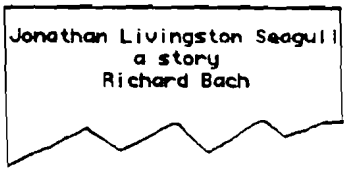
Jona...

19.



20.



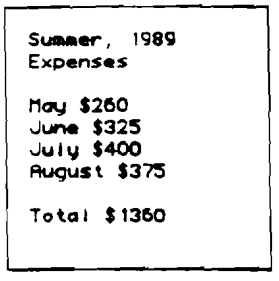


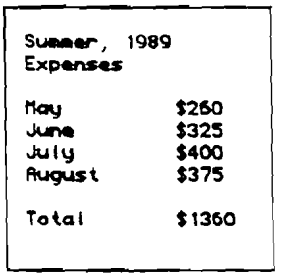
21.

What he had once hoped for

What he had once hoped for th...

22.





23.

Waht he had once...

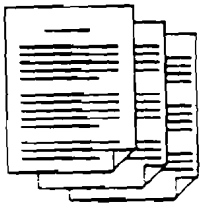
What he had once...

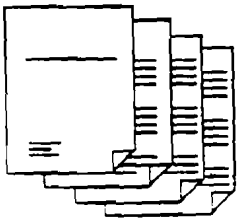
24.

What he had once hoped for the Flock, he now gained for himself alone: he learned to fly, and was not sorry for the price that he paid.

What he had once hoped for the flock, he now gained for himself alone: he learned to fly, and was not sorry for the price that he paid.

25.





26.

What he had once hoped for the Flock, he now got for only himself;

What he had once hoped for the Flock, he now gained for himself alone;

27.

What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid.

Jonathan Seagull discovered that boredom and fear and anger are the reasons that a gull's life is so short, and with these gone from his thought, he lived a long fine life indeed.

Jonathan Seagull discovered that boredom and fear and anger are the reasons that a gull's life is so short, and with these gone from his thought, he lived a long fine life indeed.

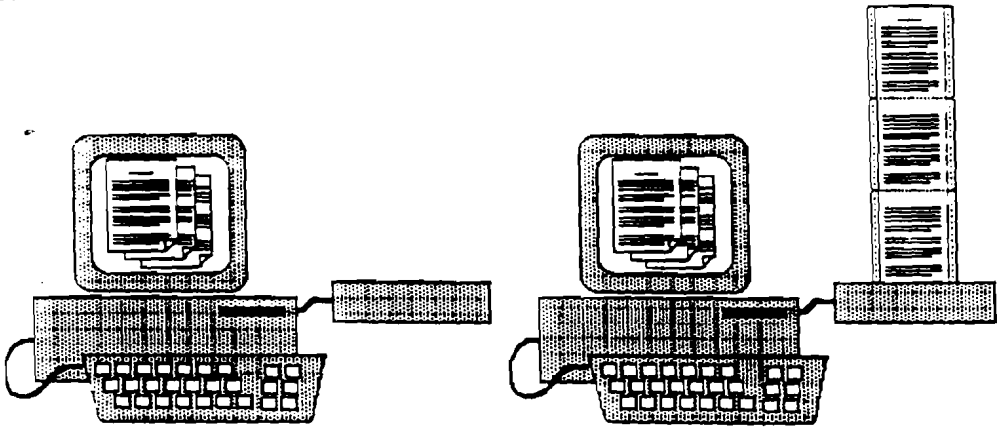
What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid.

28.

What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid.

What he had once hoped for the Flock, he now gained for himself alone;

29.



30.

What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid.

What she had once hoped for the Flock, she now gained for himself alone; she learned to fly, and was not sorry for the price that she paid.

31.

What he had once hoped for the Flock, he now gained for himself alone; he learned to fly, and was not sorry for the price that he paid.

he learned to fly, What he had once hoped for the Flock, he now gained for himself alone; and was not sorry for the price that he paid.

Please don't read this page until you've finished the questionnaire

Thank you for filling out my questionnaire. It is designed to discover the kind of language you use for text-manipulation tasks. I am developing, for my Master's thesis, a word processor which will be easy for people to learn, especially people who are not particularly fond of computers. If I can discover the kind of language different people use to refer to text-manipulation tasks, then my word-processor can use that language to communicate with different people.

After I have developed the word processor (November or December), I will test it. This will entail approximately one hour of learning / using the word processor for each person in my study. If you might be interested in using the word processor I develop, or if you have any questions about this study, leave your name and phone number below,

Name: _____

Phone number: (____)_____

Please do not call before: _____

Please do not call after: _____

and check one or both of the following:

I have questions about your study: ____

I might be interested in participating further in your study: ____

If you have any comments about this questionnaire or the nature of my study, please include them below and continue on the back of this page.

_____

_____

_____

_____

_____

_____

_____

APPENDIX B

The Word Processor

This is a word processor which is designed to be easier for the beginner to learn than commercially available word processors. It is the word processor that is being tested, NOT you. You were chosen to test the word processor because you are a beginner. Don't feel bad if you don't know how to do something, or can't figure out how to do something.

On the screen, you will see a "window" which contains an essay. Using the word processor, you will be able to change the appearance and text of the essay.

The Mouse

The device next to the keyboard is called a "mouse." When you move the mouse, the cursor moves on the screen. You can use the mouse to choose commands and to move the text cursor which is inside the window. You will use the text cursor to indicate what text you want to perform operations on.

Giving Commands

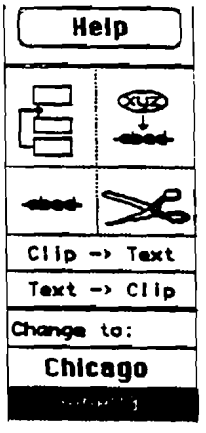
Menus

Across the top of the screen is the menu bar. The words in the menu bar are menu titles. Menu titles describe the items found in menus. For example, items in the Edit menu are editing operations; items in the View menu are options for viewing your document.

To choose from a menu, superimpose the arrow over the menu title and press down on the mouse button. The menu is "pulled down" and you can see the menu items. Hold the mouse button down while moving the pointer down to the menu item you want. When the item is displayed in inverse, release the mouse button.

The Palette

The palette also contains commands. Point to the command that you want in the palette, and click to choose it. The palette contains buttons as well as commands (the rectangle with the word Help is a button). To "press" a button, point to the button and click.



Getting Help

If at any time while you are using the word processor, you feel you need help, choose from the Help menu or click in the Help button in the palette. Choose the kind of help you want, then read the screens that follow. After you are done reading, you can use the word processor again.

Since the screens cannot be up while you are using the word processor, feel free to take notes if you find it necessary.

Instructions

1. At the top of the screen, you will see the heading "Traits," like this:

Traits

Isn't it remarkable that words can portray human individuals? You might suppose this should be impossible...

Change the heading so it looks like this:

Traits

Isn't it remarkable that words can portray human individuals? You might suppose this should be impossible...

Instructions

2. In the essay, there is a section which begins with "Selectivity:" and another section which begins with "Style:". They are as follows:

Selectivity: First we should face the fact that our images of other minds are often falsely clear....

Style: To escape the effort of making decisions we consider unimportant, we tend to develop ...

Change them so they are like this:

Style: To escape the effort of making decisions we consider unimportant, we tend to develop ...

Selectivity: First we should face the fact that our images of other minds are often falsely clear....

Instructions

3. The essay contains the following line:

Hostility evokes defensiveness. Frustration arouses aggression.

Make the line stand out more.

Instructions

4. The last sentence in the text is badly worded. It is as follows:

There is a sentence, that which is a little confusing, and it is positioned after the Minsky citation.

Reword the sentence so it is easier to read.

Questionnaire

If you have used computers before, list the computers you've used, what you've used them for, and rate your proficiency on that computer. List only the computers that you use the most.

Computer _____
What you use it for _____
Proficiency rating (circle one)
Beginner 1 2 3 4 5 Expert

Computer _____
What you use it for _____
Proficiency rating (circle one)
Beginner 1 2 3 4 5 Expert

Computer _____
What you use it for _____
Proficiency rating (circle one)
Beginner 1 2 3 4 5 Expert

Briefly describe the four tasks that you performed on the word processor today, telling what you were supposed to do, not how you did it:

_____
_____
_____
_____
_____

When you tried to accomplish the four tasks on the computer, did the computer behave in ways you did not expect? Describe what you expected the computer to do, compared to what it actually did:

_____
_____
_____
_____
_____

Briefly describe your impressions of the word processor:

_____
_____
_____
_____
_____

APPENDIX C

Options	File (by segment)	Size
	Interface.lib	8104
	Runtime.lib	18258
D N V R	Quickdraw.p	0
D N V R	ToolIntf.p	0
D N V R	Editor Globals	0
D N V R	Editor Main	66
	Segment 1	26432
D N V R	Open Palette	1710
D N V R	Editor Init	3960
	Segment 2	5670
D N V R	Editor Utilities	4884
D N V R	Help Filing	12544
D N V R	Editor Help 3	10390
	Segment 3	27818
D N V R	Editor Help 2	16634
D N V R	Editor Help	8202
	Segment 4	24836
D N V R	Help Messages	3012
	Segment 5	3012
D N V R	Help Bests	23702
D N V R	help bests 2	676
	Segment 6	24378
D N V R	Editor Help Init	3016
D N V R	Format Help	11626
D N V R	Format Help 2	11668
	Segment 7	26310
D N V R	Editing Help	4580
D N V R	Editing Help 2	19220
	Segment 8	23800
D N V R	Show Edit	1120
D N V R	Edit Records	1310
D N V R	Change Font	5688
D N V R	Editor TopLevel	22630
	Segment 9	30748
D N V R	Format Help 3	11670
	Segment 10	11670
D N V R	Editing Help 3	12134
D N V R	Editing Help 4	3874
D N V R	Editing Help 5	6804
	Segment 11	22812

Options	File (by build order)	Size
	Interface.lib	8104
	Runtime.lib	18258
D N V R	Quickdraw.p	0
D N V R	ToolIntf.p	0
D N V R	Editor Globals	0
D N V R	Open Palette	1710
D N V R	Editor Utilities	4884
D N V R	Editor Help Init	3016
D N V R	Help Filing	12544
D N V R	Help Bests	23702
D N V R	help bests 2	676
D N V R	Editing Help	4580
D N V R	Editing Help 2	19220
D N V R	Editing Help 3	12134
D N V R	Editing Help 4	3874
D N V R	Editing Help 5	6804
D N V R	Format Help	11626
D N V R	Format Help 2	11668
D N V R	Format Help 3	11670
D N V R	Help Messages	3012
D N V R	Editor Help 2	16634
D N V R	Editor Help 3	10390
D N V R	Editor Help	8202
D N V R	Show Edit	1120
D N V R	Edit Records	1310
D N V R	Change Font	5688
D N V R	Editor Init	3960
D N V R	Editor TopLevel	22630
D N V R	Editor Main	66

```
{-------------------------------------------------------------------------}
{ Margaret Stone                                                          }
{ Sc.M.  Project, Brown University                                        }
{                                                                         }
{ This unit contains all global data for the editor (but not for the help functions).  }
{-------------------------------------------------------------------------}

unit EditorGlobals;

{-------------------------------------------------------------------------}
{ Interface Section                                                       }
{-------------------------------------------------------------------------}

interface

   const
      longHeight = 440;
      longScrollUnits = 22;
      shortHeight = 380;
      shortScrollUnits = 18;
      shortestHeight = 300;
      shortestScrollUnits = 15;

      SP = ' ';
      Null = '';
      QuitOp = 'quitting';
      CloseOp = 'closing';

      SBarWidth = 16;
      HorizMax = 1024;
      PgIconSize = 20;

      FontDLOG = 100;
      AboutDLOG = 101;
      SavDiscDLOG = 102;
      SavDiscCancDLOG = 103;
      ErrorDLOG = 104;
      DuplicateDLOG = 105;


      BeamCURS = 1;
      WatchCURS = 4;

      AppleMenuId = 100;
      AboutItem = 1;

      HelpMenuId = 101;
      FormatItem = 1;
      EditItem = 2;
      SelectItem = 3;
      ViewItem = 4;

      FileMenuId = 102;
      NewItem = 1;
      OpenItem = 2;
      SaveItem = 4;
      CloseItem = 6;
      QuitItem = 7;

      EditMenuId = 103;
      CutItem = 1;
      CopyItem = 2;
      PasteItem = 3;
      MoveItem = 5;
      ReplaceItem = 6;
      DeleteItem = 7;
      DuplicateItem = 8;
      ClipEdItem = 10;

      SelectMenuId = 104;
      SelAllItem = 1;
      SelWordItem = 2;
      SelSentenceItem = 3;
      SelParaItem = 4;
      SelDocItem = 5;

      ViewMenuId = 106;
      NextLItem = 1;
      PrevLItem = 2;
      NextSItem = 4;
      PrevSItem = 5;

      TextMenuId = 107;
      ChicagoItem = 1;
      GenevaItem = 2;
      MonacoItem = 3;
      NewYorkItem = 4;
      PlainItem = 6;
      BoldItem = 7;
      ItalicItem = 8;
      UnderlineItem = 9;
      OutlineItem = 10;
      ShadowItem = 11;
      NineItem = 13;
      TenItem = 14;
      TwelveItem = 15;
      FourteenItem = 16;
      EighteenItem = 17;
      TwentyFourItem = 18;

      MessageMenuId = 108;

      Active = 0;
      Inactive = 255;
      WindowList = $9D6;

   type
{Word processor event types: }
```

```
    FromType = (nullFrom, menu, mouse, key, palette);
    WPEventType = (nullEvt, Select, ClickDnArrow, ClickUpArrow, ClickDnArrowRepeat, ClickUpArrowRepeat, ScreenP, ScreenN, LineP, LineN,
PressDnArrow, PressUpArrow, PressOK, PressCancel, DragThumb, SizeSet, SizeChange, FontSet, FontChange, StyleSet, StyleChange, Replace, Delete, Move,
Duplicate, Copy, TextClip, Cut, Paste, ClipText, UserType, DeleteType, MoveIP);
    WPERHandle = ^WPERPtr;
    WPERPtr = ^WPEventsRecord;
    WPEventsRecord = record
        theEvent: WPEventType;
        from: FromType;
        textSelected: boolean;
        numChars: integer;
        next: WPERPtr
      end;

{Select types: }
    SelectionMethods = (Normal, Word, Sentence, Paragraph, UnusualStart, Unusual);

{Font types: }
    CheckStyles = set of 1..20;

{Window types: }
    WindType = (wEdit, wClipEd, wPalette, wInstructions);

    TextInfoPtr = ^TextInfo;
    TextInfo = record
        IP: boolean;
        teh: TEHandle;
        lineHeight: integer;
        nextTI: TextInfoPtr
      end;

    WindoHandle = ^WindoPtr;
    WindoPtr = ^WindowInfo;
    WindowInfo = record
        textLines: integer;
        height: integer;
        scrollUnits: integer;
        wType: WindType;
        vScrollBar: ControlHandle;
        fileName: string;
        vFileRefN: integer;
        textDirty: boolean;
        theText: TextInfoPtr
      end;

  var
{Cursors: }
    Beam, Watch: CursHandle;

{Menus: }
    AppleMenu, HelpMenu, FileMenu, EditMenu, SelectMenu, ViewMenu, TextMenu: MenuHandle;
    MessageBar: Boolean;

{Windows: }
    TheWindo, EditWindo, ClipWindo, CantWindo, InstructionsWindo: WindoHandle;
    ClipEdStorage, utilStorage, CantW, InstructionsW: WindowRecord;
    TheWindow, EditWindow, ClipEdWindow, CantPtr, InstructionsPtr: WindowPtr;
    ClipEdRect, utilRect, utilClipRect, CantRect, InstructionsRect, utilInstructionsRect, utilInstClipRect: Rect;
    FileCount, OldStorageCount, UntitledNum: integer;
    EditText, inWindow, FileOpen, ShowClip, WindowOpen, ShowInstructions: boolean;

{Text Styles: }
    FontNum, FontSize, CheckFont, CheckSize: integer;
    FontStyle: Style;
    CheckStyle: CheckStyles;
    TheFontRect, TheSizeRect: Rect

{Buttons: }
    OkRect, CancelRect, HelpRect: Rect;
    OkButton, CancelButton, HelpButton: ControlHandle;

{Select Menu: }
    SelectMethod: SelectionMethods;
    OldSelectStart: integer;
    CR, BS, Tab, Space, Comma, ExclPt, Quote, SQuote, OpenDQuote, CloseDQuote, OpenParen, CloseParen, Period, Colon, Semi, QMark, OpenQuote,
CloseQuote, OpenBrack, CloseBrack: char;

{Events: }
    FirstClick: longint;
    Event: EventRecord;
    SafetyHandle: Handle;
    FilePtr: Ptr;
    FileLength: LONGINT;
    Done: boolean;
    KeyStrokes: integer;
{Word processor events: }
    WPEvents, WPER: WPERHandle;
    H_MenuKey, H_Palette: boolean;
    WPEventJournal: TEHandle;
{Instructional messaging: }
    OKPress, messageCame, textSMessage: boolean;
    OldMessageLength: integer;
    OldMessage: str255;
    isSelecting, isEditing, isReplacing, isClicking, SelectInstruct, SelectChoice: boolean;

{Control panel: }
    MoveImageTop: array[1..31, 1..2] of integer;
    MoveImageBottom: array[1..9, 1..2] of integer;
    ReplaceImage: array[1..23, 1..2] of integer;
    DeleteImage: array[1..8, 1..2] of integer;
    CutImage: array[1..19, 1..4] of integer;
    MoveMapTop, MoveMapBottom, ReplaceMap, DeleteMap, CutMap: BitMap;
    MoveRect, ReplaceRect, DeleteRect, CutRect, ClipToTextRect, TextToClipRect: Rect;
    ChicagoRect, GenevaRect, NewYorkRect, MonacoRect, PlainRect, BoldRect, ItalicRect, UnderlineRect, OutlineRect, ShadowRect: Rect;
    NineRect, TenRect, TwelveRect, FourteenRect, EighteenRect, TwentyfourRect: Rect;

implementation
end.
```

```
{-------------------------------------------------------------------------}
{  Margaret Stone                                                         }
{  Sc.M. Project, Brown University                                        }
{                                                                         }
{  This unit contains the code to initialize the palette.                 }
{-------------------------------------------------------------------------}

unit OpenPalette;

{-------------------------------------------------------------------------}
{  Interface Section                                                      }
{-------------------------------------------------------------------------}

interface

   uses
      EditorGlobals;

   procedure CreatePalette;

{-------------------------------------------------------------------------}
{  Implementation Section                                                 }
{-------------------------------------------------------------------------}

implementation

{-------------------------------------------------------------------------}
{  PaletteContents draws the bitmaps and QuickDraw parts of the palette.  }
{-------------------------------------------------------------------------}

   procedure PaletteContents;
      var
         TempRect: rect;
   begin
      SetPort(CemPtr);

  {Gray Lines:}
        PenPat(gray);
        MoveTo(0, 136);
        LineTo(91, 136);
        MoveTo(0, 169);
        LineTo(91, 169);
        MoveTo(0, 187);
        LineTo(91, 187);
        MoveTo(0, 246);
        LineTo(91, 246);
        MoveTo(0, 261);
        LineTo(91, 261);
        MoveTo(0, 279);
        LineTo(91, 279);
        MoveTo(0, 316);
        LineTo(91, 316);
        MoveTo(0, 331);
        LineTo(91, 331);
        MoveTo(0, 344);
        LineTo(91, 344);
        MoveTo(0, 362);
        LineTo(91, 362);
        MoveTo(0, 378);
        LineTo(91, 378);
        MoveTo(0, 416);
        LineTo(91, 416);
        MoveTo(0, 433);
        LineTo(91, 433);
        MoveTo(43, 450);
        LineTo(43, 368);
        MoveTo(43, 85);
        LineTo(43, 189);

  {Black Lines:}
        PenPat(black);
        MoveTo(0, 84);
        LineTo(91, 84);
        MoveTo(0, 85);
        LineTo(91, 85);
        MoveTo(0, 206);
        LineTo(91, 206);
        MoveTo(0, 208);
        LineTo(91, 208);
        MoveTo(0, 229);
        LineTo(91, 229);
        MoveTo(0, 297);
        LineTo(91, 297);
        MoveTo(0, 298);
        LineTo(91, 298);
        MoveTo(0, 397);
        LineTo(91, 397);
        MoveTo(0, 398);
        LineTo(91, 398);

  {BitMaps:}
        SetRect(TempRect, 550, 174, 582, 181);
        CopyBits(DeleteMap, screenBits, DeleteMap.bounds, TempRect, srcCopy, nil);
        SetRect(TempRect, 546, 119, 578, 150);
        CopyBits(MoveMapTop, screenBits, MoveMapTop.bounds, TempRect, srcCopy, nil);
        SetRect(TempRect, 546, 150, 578, 154);
        CopyBits(MoveMapBottom, screenBits, MoveMapBottom.bounds, TempRect, srcCopy, nil);
        SetRect(TempRect, 580, 170, 634, 188);
        CopyBits(CutMap, screenBits, CutMap.bounds, TempRect, srcCopy, nil);
        SetRect(TempRect, 585, 121, 627, 148);
        CopyBits(ReplaceMap, screenBits, ReplaceMap.bounds, TempRect, srcCopy, nil);

  {Text:}
        TextFont(monaco);
        TextSize(9);
        TextFace([]);
        MoveTo(10, 182);
        DrawString('Clip -> Text');
```

```
    MoveTo(10, 200);
    DrawString('Text -> Clip');
    MoveTo(2, 229);
    DrawString('Change to:');
    MoveTo(10, 311);
    DrawString('Plain Text');
    MoveTo(10, 410);
    DrawString('9 pt');
    MoveTo(4, 428);
    DrawString('12 pt');
    MoveTo(6, 445);
    DrawString('18 pt');
    MoveTo(50, 410);
    DrawString('16 pt');
    MoveTo(50, 428);
    DrawString('14 pt');
    MoveTo(50, 445);
    DrawString('24 pt');

    TextFace([bold]);
    MoveTo(28, 327);
    DrawString('Bold');

    TextFace([italic]);
    MoveTo(22, 341);
    DrawString('Italic');

    TextFace([underline]);
    MoveTo(16, 357);
    DrawString('Underline');

    TextFace([outline]);
    MoveTo(17, 374);
    DrawString('Outline');

    TextFace([shadow]);
    MoveTo(17, 391);
    DrawString('Shadow');

    TextFace([]);
    TextSize(12);
    MoveTo(19, 292);
    DrawString('Monaco');

    TextFont(systemFont);
    MoveTo(17, 340);
    DrawString('Chicago');

    TextFont(newYork);
    MoveTo(16, 275);
    DrawString('New York');

    TextFont(Geneva);
    MoveTo(20, 257);
    DrawString('Geneva');

    InvertRect(PlainRect);
    InvertRect(GenevaRect);
    InvertRect(TwelveRect);
  end;

{.....}
{CreatePalette creates the control panel palette window, the palette contents, and draws the palette}
{contents.                                                                          }
{.....}

  procedure CreatePalette;
  begin
    Dummy := NewWindow(@CardW, CardRect, 'Background', TRUE, plainDBox, nil, FALSE, 0);
    CardWrite := WriteWindowStructure(GetCWindow(Window));
    CardWrite^.theText := nil;
    SetWRect(CardCardPtr, CardCardWrite);
    CardWrite^.vScrollBar := nil;
    CardWrite^.textLines := 0;
    CardWrite^.textOnly := FALSE;
    CardWrite^.fileName := 'BackGroundWind';
    CardWrite^.vRefNum := 3;
    CardWrite^.wType := wPalette;
    OkButton := NewControl(CardPtr, OkRect, 'OK', FALSE, 0, 0, 0, pushButProc, 0);
    CancelButton := NewControl(CardPtr, CancelRect, 'Cancel', FALSE, 0, 0, 0, pushButProc, 0);
    HelpButton := NewControl(CardPtr, HelpRect, 'Help', TRUE, 0, 0, 0, pushButProc, 0);
    PaletteCreated;
  end;

end.
```

```
{-----------------------------------------------------------------------}
{ Margaret Storo                                                        }
{ Sc.M. Project, Brown University                                       }
{                                                                       }
{ This unit contains various utilities used by the top level and file I/O code of the editor. }
{-----------------------------------------------------------------------}

unit EditorUtilities;

{-----------------------------------------------------------------------}
{ Interface Section                                                     }
{-----------------------------------------------------------------------}

interface


  uses
    EditorGlobals;

  type
    StandardType = (StandardDat, StandardPut);

  procedure HandleUpdate;
  procedure MyErrorAlert (msg: Str255);
  procedure ReadDataStruct;
  procedure WriteDataStruct;
  procedure UpdateCBuDat (ResetUpdate: boolean);
  function HourGlass: Str255;
  function DuplicateFile (fName: Str255): boolean;
  function DuplicateName (fName: Str255): Str255;
  function StandardFile (opCode: StandardType; edName: Str255; fType: OSType; var vRef: integer): Str255;
  function ReadFile (fName: Str255; vRef: integer): boolean;
  function WriteFile (fName: Str255; vRef: integer; teh: TEHandle): boolean;
  function AboFile (title: string): integer;
  procedure ChangeFile (window: WindowPtr; fName: Str255);
  procedure DoubleFile (window: WindowPtr);
  procedure TestPrint;
  procedure FrameDBase (dLog: DialogPtr; fNum: integer);
  procedure DoAbout;
  procedure DoAgain (item: integer);
  function FindWordStart: integer;
  function FindWordFinish: integer;
  function FindParaStart: integer;
  function FindParaFinish: integer;
  function FindSentenceStart: integer;
  function FindSentenceFinish: integer;

{-----------------------------------------------------------------------}
{ Implementation Section                                                }
{-----------------------------------------------------------------------}

implementation

{-----------------------------------------------------------------------}
{ The FindStart/Finish functions find the starts and finishes of the words, sentences, and }
{ paragraphs which contain the insertion point (or selection range).    }
{-----------------------------------------------------------------------}

  function FindWordStart:      {integer}
    var
      txt: CharsHandle;
      insP: integer;
      fdound: boolean;
      letter: char;
  begin
    fdound := FALSE;
    with EdWinRec^, theText^, txH^^ do
      begin
        txt := ^CharsHandle(hTxt);
        insP := selStart;
        while ((insP >= 0) and not fdound do
          begin
            letter := txt^^[insP];
            if (letter = SQuote) or (letter = OpenSQuote) or (letter = ClosedSQuote) then
              begin
                letter := txt^^[insP + 1];
                if ((letter <= 'Z') and (letter >= 'A')) or ((letter <= 'z') and (letter >= 'a')) then
                  insP := insP - 1
                else
                  fdound := TRUE
              end
            else if (letter = Quote) or (letter = OpenQuote) or (letter = Comma) or (letter = CR) or (letter = Space) or (letter = ExclPt) or (letter =
OpenParen or (letter = ClosedParen) or (letter = Period) or (letter = Colon or (letter = Semi) or (letter = QMark) or (letter = ClosedQuote) or (letter =
OpenBrack) or (letter = ClosedBrack) then
              fdound := TRUE
            else
              insP := insP - 1;
          end
      end;
    if insP <> 0 then
      insP := insP + 1;
    FindWordStart := insP;
  end;

  function FindWordFinish:      {integer}
    var
      txt: CharsHandle;
      insP: integer;
      fdound: boolean;
      letter: char;
  begin
    fdound := FALSE;
    with EdWinRec^, theText^, txH^^ do
      begin
        txt := CharsHandle(hTxt);
        insP := selEnd;
        while ((insP <= teLength) and not fdound) do
          begin
```

```
                letter := txt^^[insP];
                if (letter = SQuote) or (letter = OpenSQuote) or (letter = CloseSQuote) then
                  begin
                    letter := txt^^[insP + 1];
                    if ((letter <= 'Z') and (letter >= 'A')) or ((letter <= 'z') and (letter >= 'a')) then
                      insP := insP + 1
                    else
                      Sfound := TRUE
                  end
                else if (letter = Comma) or (letter = Period) or (letter = ExclPt) or (letter = QMark) or (letter = CR) or (letter = Space) or (letter = Colon)
or (letter = Semi) or (letter = CloseBrack) or (letter = CloseQuote) then
                  Sfound := TRUE
                else
                  insP := insP + 1;
              end
          end;
        FindWordFinish := insP;
      end;

      function FindSentenceStart:      integer)
        var
          txt: CharsHandle;
          insP: integer;
          Sfound: boolean;
          letter: char;
      begin
        Sfound := FALSE;
        with EdWrite^^, theText^, txt^^ do
          begin
            txt := CharsHandle(theText);
            insP := selStart;
            while ((insP >= 0) and not Sfound) do
              begin
                letter := txt^^[insP];
                if (letter = CR) or (letter = ExclPt) or (letter = Period) or (letter = QMark) then
                  Sfound := TRUE
                else
                  insP := insP - 1;
              end
          end;
        repeat
          insP := insP + 1;
          letter := txt^^[insP];
          if (letter = Quote) then
            begin
              letter := txt^^[insP + 1];
              if not (((letter >= 'A') and (letter <= 'Z')) or ((letter >= 'a') and (letter <= 'z')) or (letter = OpenSQuote) or (letter = SQuote) or (letter =
OpenBrack) or (letter = OpenParen)) then
                insP := insP + 1
              else
                letter := txt^^[insP];
            end;
        until ((letter >= 'A') and (letter <= 'Z')) or ((letter >= 'a') and (letter <= 'z')) or (letter = OpenSQuote) or (letter = SQuote) or (letter = OpenQuote) or
(letter = Quote) or (letter = OpenBrack) or (letter = OpenParen);
        FindSentenceStart := insP;
      end;

      function FindSentenceFinish:      integer)
        var
          txt: CharsHandle;
          insP: integer;
          Sfound: boolean;
          letter: char;
      begin
        Sfound := FALSE;
        with EdWrite^^, theText^, txt^^ do
          begin
            txt := CharsHandle(theText);
            insP := selEnd;
            while ((insP <= theLength) and not Sfound) do
              begin
                letter := txt^^[insP];
                if (letter = CR) or (letter = ExclPt) or (letter = Period) or (letter = QMark) then
                  Sfound := TRUE
                else
                  insP := insP + 1;
              end
          end;
        letter := txt^^[insP + 1];
        if (letter = Quote) or (letter = CloseQuote) or (letter = SQuote) or (letter = CloseSQuote) or (letter = CloseParen) or (letter = CloseBrack) then
          insP := insP + 1;
        FindSentenceFinish := insP + 1;
      end;

      function FindParaStart:      integer)
        var
          txt: CharsHandle;
          insP: integer;
          Sfound: boolean;
          letter: char;
      begin
        Sfound := FALSE;
        with EdWrite^^, theText^, txt^^ do
          begin
            txt := CharsHandle(theText);
            insP := selStart;
            while ((insP >= 0) and not Sfound) do
              begin
                letter := txt^^[insP];
                if (letter = CR) then
                  Sfound := TRUE
                else
                  insP := insP - 1;
              end
          end;
        if insP <> 0 then
          insP := insP + 1;
        FindParaStart := insP;
```

```
end;

function FindParaFinish:        {:integer}
   var
      txt: CharsHandle;
      insP: integer;
      SFound: boolean;
      letter: char;
begin
   SFound := FALSE;
   with EditWinRec^, theText^, teh^^ do
      begin
         txt := CharsHandle(hText);
         insP := selEnd;
         while ((insP <= teLength) and not SFound) do
            begin
               letter := txt^^[insP];
               if (letter = CR) then
                  SFound := TRUE
               else
                  insP := insP + 1;
            end
      end;
   FindParaFinish := insP;
end;


{-----------------------------------------------------------------------}
{ DoApple is the code for the "Apple" menu. Either the "About" box is   }
{ displayed or a Desk Accessory is opened.                              }
{-----------------------------------------------------------------------}

   procedure DoApple;        {(item: integer)}
      var
         accName: Str255;
         accNumber: integer;
   begin
      if item = AboutItem then
         DoAbout
      else
         begin
            GetItem(AppleMenu, item, accName);
            accNumber := OpenDeskAcc(accName)
         end
   end;


{-----------------------------------------------------------------------}
{ HandleUpdate is the Main Event Loop code to update windows. It is     }
{ defined in this unit so that routines may use it to force window      }
{ updates.  The controls, Grow icon, and text of the widow in the       }
{ global "Event" are updated.                                           }
{-----------------------------------------------------------------------}

   procedure HandleUpdate;
      var
         oldPort: GrafPtr;
         wrwin: WindowHandle;
         window: WindowPtr;
   begin
      window := WindowPtr(Event.message);
      wrwin := WindowHandle(GetWRefCon(window));
      if (window <> ConPtr) then
         begin
            GetPort(oldPort);
            SetPort(window);
            BeginUpDate(window);
            EraseRect(window^.portRect);
            if (window = EditWindow) then
               DrawControls(window);
            HLock(Handle(wrwin));
            TEUpdate(wrwin^^.theText^.teh^^.viewRect, wrwin^^.theText^.teh);
            HUnlock(Handle(wrwin));
            EndUpdate(window);
            SetPort(oldPort)
         end
      else
         begin
            GetPort(oldPort);
            SetPort(window);
            BeginUpDate(window);
            DrawControls(window);
            EndUpdate(window);
            SetPort(oldPort)
         end
   end;

{-----------------------------------------------------------------------}
{ MyErrorAlert sets up a general error reporting dialog. A dialog is    }
{ used instead of an alert so that the required storage can be          }
{ allocated statically instead of on the Heap. Note that any windows    }
{ that may be open are updated before the dialog is drawn.              }
{-----------------------------------------------------------------------}

   procedure MyErrorAlert( (msg : Str255) );
      var
         item: integer;
         oldPort: GrafPtr;
         eDialog: DialogPtr;
         errorStorage: DialogRecord;
   begin
      InitCursor;
      while GetNextEvent(updateMask, Event) do
         HandleUpdate;
      GetPort(oldPort);
      ParamText(msg, Null, Null, Null);
      eDialog := GetNewDialog(ErrorDLOG, @errorStorage, WindowPtr(-1));
      SetPort(eDialog);
      FrameDItem(eDialog, OK);
      ModalDialog(nil, item);
```

```
    CloseDialog(eDialog);
    SetPort(oldPort)
  end;


{-------------------------------------------------------------------------}
{ Num2Str is a functional interface to the built in procedure NumToString }
{-------------------------------------------------------------------------}

  function Num2Str (num: integer): Str255;
    var
      str: Str255;
  begin
    NumToString(num, str);
    Num2Str := str
  end;


{-------------------------------------------------------------------------}
{ UpdateClipBd tests if the TE scrap has changed. If so, the Clipboard window is updated to }
{ show the current TE scrap. Note that only the teLength of the Clipboard window needs to   }
{ be updated since the hText field is initialize to be the TE Scrap Handle.                 }
{-------------------------------------------------------------------------}

  procedure UpdateClipBd; { (forceUpdate : boolean) }
    var
      oldPort: GrafPtr;
      wrote: WindowHandle;
      scrapInfo: PScrapStuff;
  begin
    if forceUpdate then {or (OldScrapCount <> scrapInfo^.scrapCount) then}
      begin
        GetPort(oldPort);
        SetPort(ClipBdWindow);
        wrote := ClipWrite;
        HLock(Handle(wrote));
        with wrote^^.theText^ do
          begin
            TECalText(teh);
            InvalRect(teh^^.viewRect)
          end;
        HUnlock(Handle(wrote));
        SetPort(oldPort)
      end
  end;

{-------------------------------------------------------------------------}
{ DuplicateFile tests if the name "fName" is the name of a window. It returns TRUE if so. }
{-------------------------------------------------------------------------}

  function DuplicateFile; { (fName : Str255) : boolean }
    var
      i: integer;
      title: Str255;
      found: boolean;
  begin
    i := 0;
    found := FALSE;

    while (i < MaxFiles) and not found do
      begin
        i := i + 1;
        with Files[i] do
          if isOpen then
            begin
              GetWTitle(@isStorage, title);
              found := (fName = title
            end
      end;

    DuplicateFile := found
  end;

{-------------------------------------------------------------------------}
{ NewName generates a FileWindow name of the form "UntitledXX" where XX is an integer.  }
{ After generating the name, the integer is incremented so the same name will not be    }
{ generated again. Note that the name may not be the same as that of a window.          }
{-------------------------------------------------------------------------}

  function NewName; { : Str255 }
    var
      name: Str255;
  begin
    repeat
      name := Concat('Untitled', Num2Str(UntitledNum));
      UntitledNum := UntitledNum + 1
    until not DuplicateFile(name);
    NewName := name
  end;

{-------------------------------------------------------------------------}
{ DuplicateName is called if the above procedure, "DuplicateFile", returns TRUE. It gets an }
{ "UntitledXX" name and prompts the user if he/she wants to open the file "fName" using     }
{ the untitled name. If so, the untitled name is returned as result. Otherwise, the null string }
{ is returned as result and the integer used to generate the untitled name is decremented so }
{ that the untitled name can be re-used. Note that any windows that may be open are        }
{ updated before the dialog is drawn.                                                      }
{-------------------------------------------------------------------------}

  function DuplicateName; { (fName : Str255) : Str255 }
    var
      item: integer;
      dName: Str255;
      oldPort: GrafPtr;
      dupDialog: DialogPtr;
      dupStorage: DialogRecord;
  begin
```

```
    InitCursor;
    while GetNextEvent(updateMask, Event) do
        HandleUpdate;
    GetPort(oldPort);
    dName := NewName;
    ParamText(dName, dName, Null, Null);
    dupDialog := GetNewDialog(DuplicateDLOG, @dupStorage, WindowPtr(-1));
    SetPort(dupDialog);
    FrameDItem(dupDialog, Ok);
    ModalDialog(nil, item);
    CloseDialog(dupDialog);
    SetPort(oldPort);
    if item = Ok then
        DuplicateName := dName
    else
        begin
            DuplicateName := Null;
            UntitledNum := UntitledNum - 1
        end
end;
```

```
{ StandardFile prompts the user with a SFGetFile/SFPutFile dialog when saving or  }
{ opening a file. For SFGetFile, only files of type "fType" are displayed. If the dialog is  }
{ cancelled, the null string is returned. Otherwise, the file name and the VRefNum or  }
{ WDRefID are returned.  }
```

```
function StandardFile( ( opCode : StandardType; )
                       (    oldName : Str255; )
                       (  fType : OSType )
                       (    var vRef : Integer) : Str254 )
    var
        where: Point;
        reply: SFReply;
        testType: SFTypeList;
begin
    where.h := 88;
    where.v := 88;
    testType[0] :=  fType;
    if opCode = StandardGet then
        SFGetFile(where, Null, nil, 1, testType, nil, reply)
    else
        SFPutFile(where, Null, oldName, nil, reply);
    with reply do
        if not good then
            StandardFile := Null
        else
            begin
                StandardFile := fName;
                vRef := vRefNum
            end
end;
```

```
{ ReadFile attempts to read a file specified by the user into a handled buffer. The result of  }
{ each "FS" operation is tested, and if any errors occur, an Alert with an appropriate  }
{ message is put up and the procedure is exited. Note that the file is closed before  }
{ attempting  to allocate the handle in case we run out of memory and exit the program via  }
{ the GrowZone procedure. This insures that a file will not be left open.  }
```

```
function ReadFile( ( fName : Str255; )
                   (    vRef : Integer) : boolean )
    label
        10;
    var
        fSize: longint;
        opened: boolean;
        fRef, errCode: Integer;

    procedure ReadErr (msg: Str255);
    begin
        MyErrorAlert(Concat('Error while reading from ', fName, ': ', msg));
        ReadFile := FALSE;
        goto 10
    end;

    procedure TestErr (errCode: Integer);
    begin
        if errCode <> noErr then
            ReadErr(Concat('I/O error ', Num2Str(errCode)))
    end;

begin {ReadFile}
    opened := FALSE;
    TestErr(FSOpen(fName, vRef, fRef));
    opened := TRUE;
    TestErr(GetEOF(fRef, fSize));
    if fSize > MaxInt then                          {32767 is the limit of TB}
        ReadErr('File is too big to edit');
    errCode := FSClose(fRef);                       {Close file while allocating file buffer}
    errCode := FSOpen(fName, vRef, fRef);           {Buffer is safely allocated, re-open file}
    TestErr(FSRead(fRef, fSize, FilePtr));
    FileLength := fSize;
    ReadFile := TRUE;
10:
    if opened then
        errCode := FSClose(fRef)
end;
```

```
{ WriteFile attempts to write the text in the TEHandle "teh" to a file specified by the user.  }
{ As with ReadFile, each "FS" operation is tested for errors and an Alert with an  }
{ appropriate message is put up if any occur. Note that we must test for the case that a  }
{ duplicate file exists when creating the file. This is not an error.  }
```

```
function WriteFile( ( fName : Str255; )
                    ( vRef : integer; )
                    ( teh : TEHandle) : boolean )
  label
    10;
  var
    fSize: longint;
    fRef, ioResult: integer;

  procedure TestErr (errCode: integer);
  begin
    if errCode <> noErr then
      begin
        MyErrorAlert(Concat('Error while writing to ', fName, ': I/O error #', Num2Str(errCode)));
        WriteFile := FALSE;
        goto 10
      end
  end;

begin {WriteFile}
  ioResult := Create(fName, vRef, '    ', 'TEXT');
  if ioResult <> dupFNErr then
    TestErr(ioResult);
  TestErr(FSOpen(fName, vRef, fRef));
  TestErr(SetEOF(fRef, 0));                    {make sure that the file is empty}
  fSize := teh^^.teLength;
  TestErr(FSWrite(fRef, fSize, teh^^.hText^));
  WriteFile := TRUE;
10:
  ioResult := FSClose(fRef);
  ioResult := FlushVol(nil, vRef)
end;
```

AllocFile allocates one of the available files when the user does a "New" or "Open". Basically, it finds the first non-open slot and returns the index. The number of open files is incremented and the corresponding menu item in the "Windows" menu is enabled, with the file name as the item name.

```
function AllocFile( ( file : string ) : integer )
  var
    i: integer;
    found: boolean;
begin
  AllocFile := 0;

  i := 0;
  found := FALSE;
  while not found and (i < MaxFiles) do
    begin
      i := i + 1;
      with Files[i] do
        if not isOpen then
          begin
            AllocFile := i;
            isOpen := TRUE;
            FileCount := FileCount + 1;
            SetItem(WindowMenu, i, file);
            found := TRUE
          end
    end

end;
```

ChangeFile changes the window title and window-menu item name for a given window when the user does a "SaveAs...".

```
procedure ChangeFile( ( window : WindowPtr; )
                      ( fName : Str255) )
  var
    i: integer;
    found: boolean;
begin
  i := 0;
  found := FALSE;
  SetWTitle(window, fName);

  while not found and (i < MaxFiles) do
    begin
      i := i + 1;
      with Files[i] do
        if window = @isStorage then
          begin
            SetItem(WindowMenu, i, fName);
            found := TRUE
          end
    end

end;
```

DeallocFile releases a file after the user has closed it, so that it may be reused. The number of open files is decremented and the corresponding menu item in the "Windows" menu is disabled with "Available" as the item name. The PortRect of the window that the file was in is remembered so that the next file opened in this slot will occupy the same position. If the window is zoomed, the "preZoomRect" is remembered instead.

```
procedure DeallocFile( ( window : WindowPtr) )
  var
    i: integer;
    found: boolean;
    wrec: WindowRecord;
begin
  i := 0;
```

```
    found := FALSE;
    while not found and (i < MaxFiles) do
        begin
            i := i + 1;
            with Files[i] do
                if window = @utStorage then
                    begin
                        wrote := WriteHandle(GetWRefCon(window));
                        with wrote^^ do
                            if zoomed then
                                utRect := preZoomRect
                            else
                                begin
                                    utRect := window^.portRect;
                                    LocalToGlobal(utRect.topLeft);
                                    LocalToGlobal(utRect.botRight)
                                end;
                        FileCount := FileCount - 1;
                        SetItem(WindowMenu, i, 'Available');
                        utOpen := FALSE;
                        found := TRUE
                    end
        end
end;

{ TestFront is called before action is taken on menu commands to correctly enable or disable
  "Edit" menu and various "File" menu items in case a desk accessory is the front window, or
  no window is currently up. }

procedure TestFront;
    var
        hr: WindowPeek;
        deskAcc: boolean;
begin
    hr := WindowPeek(FrontWindow);
    if (hr = nil) or (TheWindow = nil) then
        begin
            if hr = nil then
                deskAcc := FALSE
            else
                deskAcc := hr^.windowKind < 0;
        end
end;

{ AllFileBytes is a utility used by "DoAbout" to calculate the number of bytes of memory
  being used in all open files. Note, the Clipboard window is NOT included. The number of
  bytes is returned as a string. }

function AllFileBytes: Str255;
    var
        wrote: WriteHandle;
        window: WindowPeek;
        bytes, kBytes: longint;
begin
    bytes := 0;
    window := WindowPeek(FrontWindow);
    while window <> nil do
        begin
            if window^.windowKind = userKind then
                begin
                    wrote := WriteHandle(GetWRefCon(WindowPtr(window)));
                    if wrote^^.wType = wEdit then
                        bytes := bytes + wrote^^.theText^.len^^.teLength
                end;
            window := window^.nextWindow
        end;
    if bytes = 0 then
        kBytes := 0
    else
        kBytes := (bytes div 1000) + 1;
    AllFileBytes := Num2Str(kBytes)
end;

{ TopWindowBytes is a utility used by "DoAbout" which returns a string representing the
  number of bytes in, and the name of the current top window. It works for both Edit
  windows and the Clipboard window. In all other cases, a message stating that no wndow
  is open will be returned. }

function TopWindowBytes: Str255;
    var
        title: Str255;
        wrote: WriteHandle;
begin
    if TheWindow = nil then
        TopWindowBytes := 'No window is open'
    else
        begin
            GetWTitle(TheWindow, title);
            wrote := WriteHandle(GetWRefCon(TheWindow));
            TopWindowBytes := ConcatNum2Str(wrote^^.theText^.len^^.teLength, ' bytes in "' + title + '"')
        end
end;

{ DrawOLine draws a horizontal line through the item rectangle of a given item in a given
  dialog. The item should in general be a user item which was put into the dialog for the
  specific purpose of allocating space for a line to be drawn in. }

procedure DrawOLine (dLog: DialogPtr; iNum: integer);
    var
        iBox: Rect;
```

```
        iType: integer;
        iHandle: Handle;
  begin
    GetDItem(dLog, iNum, iType, iHandle, iBox);
    with iBox do
      DrawLine(left, top + (bottom - top) div 2, right, top + (bottom - top) div 2)
  end;

{-------------------------------------------------------------------------------}
{ CenterDItem centers some text in the item rectangle of a given item in a given dialog. The }
{ item should be a static text or user item.                                    }
{-------------------------------------------------------------------------------}

  procedure CenterDItem (dLog: DialogPtr; iNum: integer; iText: Str255);
    var
      iBox: Rect;
      iType: integer;
      iHandle: Handle;
  begin
    GetDItem(dLog, iNum, iType, iHandle, iBox);
    TextBox(Ptr(Ord(@iText) + 1), Length(iText), iBox, teJustCenter)
  end;

{-------------------------------------------------------------------------------}
{ FrameDItem draws a round cornered rectangle around the item rectangle of a given item }
{ in a given dialog. This is usually done to indicate the default choice button of a dialog. }
{-------------------------------------------------------------------------------}

  procedure FrameDItem;  { (dLog : DialogPtr; }
        {    iNum : integer) }
    var
      iBox: Rect;
      iType: integer;
      iHandle: Handle;
      oldPenState: PenState;
  begin
    GetPenState(oldPenState);
    GetDItem(dLog, iNum, iType, iHandle, iBox);
    InsetRect(iBox, -4, -4);
    PenSize(3, 3);
    FrameRoundRect(iBox, 16, 16);
    SetPenState(oldPenState)
  end;

{-------------------------------------------------------------------------------}
{ DoAbout puts up the "About" dialog. The about box currently displays the number of bytes }
{ in all open files, the number of bytes available for use, and the number of bytes in the }
{ current top window. Clicking and releasing the mouse button cancels the about box.   }
{-------------------------------------------------------------------------------}

  procedure DoAbout;
    const
      topLine = 3;
      middleLine = 6;
      bottomLine = 8;
      allFilesSize = 4;
      freeSize = 5;
      topWindowSize = 7;
    var
      e: EventRecord;
      oldPort: GrafPtr;
      aDialog: DialogPtr;
      aboutStorage: DialogRecord;
  begin
    GetPort(oldPort);
    aDialog := GetNewDialog(AboutDLOG, @aboutStorage, WindowPtr(-1));
    SetPort(aDialog);
    DrawDialog(aDialog);
    PenSize(2, 2);
    DrawDLine(aDialog, topLine);
    DrawDLine(aDialog, middleLine);
    DrawDLine(aDialog, bottomLine);
    CenterDItem(aDialog, allFilesSize, Concat(AllFilesBytes, 'K bytes allocated for files'));
    CenterDItem(aDialog, freeSize, Concat(Num2Str(FreeMem div 1000), 'K bytes are free'));
    CenterDItem(aDialog, topWindowSize, TopWindowBytes);
    while not Button do

    while GetNextEvent(mDownMask + mUpMask, e) do

    CloseDialog(aDialog);
    SetPort(oldPort)
  end;


end.
```

```
{-------------------------------------------------------------------------}
{ Margaret Storo                                                          }
{ Sc.M. Project, Brown University                                         }
{                                                                         }
{ This unit contains the type definitions for the help record type (the user profile), and other help data.}
{ It also contains the code to initialize these structures.               }
{-------------------------------------------------------------------------}

unit EditorHelpers;

{-------------------------------------------------------------------------}
{ Interface Section                                                       }
{-------------------------------------------------------------------------}

interface

  uses
    EditorGlobals;

{Dialog box and string constants (from the resource file): }
  const
    HelpButtonDLOG = 201;
    FormatDLOG = 202;
    EdDLOG = 203;
    StyleDLOG = 204;
    SizeDLOG = 205;
    TypeFaceDLOG = 206;
    SpellingDLOG = 207;
    RewordingDLOG = 208;
    DeletingDLOG = 209;
    SwitchingDLOG = 210;
    MovingDLOG = 211;
    ReplacingDLOG = 212;
    DuplicatingDLOG = 213;
    Duplicating2DLOG = 219;
    CuttingDLOG = 214;
    CopyingDLOG = 215;
    PastingDLOG = 216;
    SelectDLOG = 217;
    ViewDLOG = 218;

    HelpFormatSTR = 100;
    HelpCCSTR = 101;
    HelpPasteSTR = 102;
    HelpMoveSTR = 103;
    HelpSelectSTR = 104;
    HelpViewSTR = 105;
    HelpReplaceSTR = 106;

{User profile and TOC (text on clipboard) type definitions: }
  type
    ProfileType = (H_neverUsed, H_SelectMenu, H_SelectMouse, H_SelectKey, H_ClickUpArrow, H_ClickUpArrowRepeat, H_PressUpArrow,
H_ScreenPMouse, H_ScreenPKey, H_ScreenPMenu, H_LinePKey, H_LinePMenu, H_ClickDnArrow, H_ClickDnArrowRepeat, H_PressDnArrow,
H_ScreenNMouse, H_ScreenNMenu, H_ScreenNKey, H_LineNMenu, H_LineNKey, H_DragThumb, H_Replace1, H_Replace2, H_Replace3Menu, H_Replace3Palette,
H_Replace4Menu, H_Replace4Palette, H_Replace5, H_Replace6, H_Replace7Menu, H_Replace7Palette, H_Replace8Menu, H_Replace8Palette, H_Delete1,
H_Delete2, H_Delete3Menu, H_Delete3Palette, H_Delete4Menu, H_Delete4Palette, H_Move1Menu, H_Move1Key, H_Move1Palette, H_Move2Menu,
H_Move2Key, H_Move2Palette, H_Move3Menu, H_Move3Key, H_Move3Palette, H_Move4Menu, H_Move4Key, H_Move4Palette, H_Move5Menu, H_Move5Key,
H_Move5Palette, H_Move6Menu, H_Move6Key, H_Move6Palette, H_Move7Menu, H_Move7Palette, H_Move8Menu, H_Move8Palette, H_Duplicate1,
H_Duplicate2, H_Duplicate3Menu, H_Duplicate3Key, H_Duplicate3Palette, H_Duplicate4Menu, H_Duplicate4Key, H_Duplicate4Palette, H_Cut1Menu,
H_Cut1Key, H_Cut1Palette, H_Cut2Menu, H_Cut2Key, H_Cut2Palette, H_Copy1Menu, H_Copy1Key, H_Copy1Palette, H_Copy2Menu, H_Copy2Key,
H_Copy2Palette, H_Paste1Menu, H_Paste1Key, H_Paste1Palette, H_Paste2Menu, H_Paste2Key, H_Paste2Palette, H_Text1Menu, H_Text1Palette,
H_Text2Menu, H_Text2Palette, H_Text3Menu, H_Text3Palette, H_Text4Menu, H_Text4Palette, H_Text5Menu, H_Text5Palette, H_Text6Menu,
H_Text6Palette, H_Text7Menu, H_Text7Palette, H_Text8Menu, H_Text8Palette, H_Text9Menu, H_Text9Palette);

      TOCMethod = (noTOC, CutTOC, CopyTOC, DuplicateTOC, MoveTOC, CopyDelete, CopyDeleteType);
      TOCNumber = 0..8;
      TOCHandle = ^TOCPtr;
      TOCPtr = ^TOCRec;
      TOCRec = record
          method: TOCMethod;
          number: TOCNumber;
          IPMouse, eventStruct: boolean
        end;

      HelpNumbers = 0..256;
      HelpHandle = ^HelpPtr;
      HelpPtr = ^HelpRec;
      HelpRec = record
          HRNumLabels: HelpNumbers;
          HRSelectFromMenu, HRSelectFromMouse, HRSelectFromKey: HelpNumbers;
          HRNumViewP: HelpNumbers;
          HRClickUpArrow, HRClickUpArrowRepeat, HRPressUpArrow, HRScreenPMouse, HRScreenPKey: HelpNumbers;
          HRScreenPMenu, HRLinePKey, HRLinePMenu: HelpNumbers;
          HRNumViewN: HelpNumbers;
          HRClickDnArrow, HRClickDnArrowRepeat, HRPressDnArrow, HRScreenNMouse: HelpNumbers;
          HRScreenNMenu, HRScreenNKey, HRLineNMenu, HRLineNKey: HelpNumbers;
          HRDragThumb: HelpNumbers;
          HRNumReplace: HelpNumbers;
          HRReplace1, HRReplace2, HRReplace3FromMenu, HRReplace3FromPalette, HRReplace4FromMenu, HRReplace4FromPalette, HRReplace5,
HelpNumbers;
          HRReplace6, HRReplace7FromMenu, HRReplace7FromPalette, HRReplace8FromMenu, HRReplace8FromPalette: HelpNumbers;
          HRNumDelete: HelpNumbers;
          HRDelete1, HRDelete2, HRDelete3FromMenu, HRDelete3FromPalette, HRDelete4FromMenu, HRDelete4FromPalette: HelpNumbers;
          HRNumMove: HelpNumbers;
          HRMove1Menu, HRMove1Key, HRMove1Palette, HRMove2Menu, HRMove2Key, HRMove2Palette, HRMove3Menu: HelpNumbers;
          HRMove3Key, HRMove3Palette, HRMove4Menu, HRMove4Key, HRMove4Palette, HRMove5Menu, HRMove5Key: HelpNumbers;
          HRMove5Palette, HRMove6Menu, HRMove6Palette, HRMove6Key, HRMove7Menu, HRMove7Palette: HelpNumbers;
          HRMove8Menu, HRMove8Palette: HelpNumbers;
          HRNumDuplicate: HelpNumbers;
          HRDuplicate1, HRDuplicate2, HRDuplicate3Menu: HelpNumbers;
          HRDuplicate3Key, HRDuplicate3Palette, HRDuplicate4Menu, HRDuplicate4Key, HRDuplicate4Palette: HelpNumbers;
          HRNumCut: HelpNumbers;
          HRCut1Menu, HRCut1Key, HRCut1Palette, HRCut2Menu, HRCut2Key, HRCut2Palette: HelpNumbers;
          HRNumCopy: HelpNumbers;
          HRCopy1Menu, HRCopy1Key, HRCopy1Palette, HRCopy2Menu, HRCopy2Palette, HRCopy2Key: HelpNumbers;
          HRNumPaste: HelpNumbers;
          HRPaste1Menu, HRPaste1Key, HRPaste1Palette, HRPaste2Menu, HRPaste2Key, HRPaste2Palette: HelpNumbers;
          HRNumText: HelpNumbers;
```

```
        HRText1Menu, HRText1Palette, HRText4Menu, HRText4Palette, HRText7Menu, HRText7Palette: HelpNumbers;
        HRNumFont: HelpNumbers;
        HRText2Menu, HRText2Palette, HRText5Menu, HRText5Palette, HRText8Menu, HRText8Palette: HelpNumbers;
        HRNumStyle: HelpNumbers;
        HRText3Menu, HRText3Palette, HRText6Menu, HRText6Palette, HRText9Menu, HRText9Palette: HelpNumbers;
      end;

  var
    TOC: TOCHandle;
    HR: HelpHandle;
    TutorPasse, TutorFontSet, MustTutorPasse, MustTutorFont: boolean;
    UserHelpPrefs: TEHandle;

  procedure InitHelp;

{-------------------------------------------------------------------------------}
{ Implementation Section                                                        }
{-------------------------------------------------------------------------------}

implementation

{-------------------------------------------------------------------------------}
{ Initialization Procedures                                                     }
{-------------------------------------------------------------------------------}

  procedure SetTOC (theMethod: TOCMethod; theNumber: TOCNumber);
  begin
    TOC^^.method := theMethod;
    TOC^^.number := theNumber;
    TOC^^.IfMoved := FALSE;
    TOC^^.eventSince := FALSE
  end;

  procedure InitTOC;
  begin
    SetTOC(noTOC, 0)
  end;

  procedure UpdateTOC (theWPEvent: WPEventType);
  begin
    if (TOC^^.method <> noTOC) then
      case theWPEvent of
        UserType, DeleteType, MoveIP, Select:
          begin
            TOC^^.eventSince := TRUE;
            TOC^^.IfMoved := TRUE;
          end;
        Paste, Delete, Replace:
          TOC^^.eventSince := TRUE;
        otherwise
          ;
      end
  end;

  procedure InitHR;
  begin
    HR^^.HRNumSelects := 0;
    HR^^.HRSelectFromMenu := 0;
    HR^^.HRSelectFromMouse := 0;
    HR^^.HRSelectFromKey := 0;
    HR^^.HRNumViewP := 0;
    HR^^.HRClickUpArrow := 0;
    HR^^.HRClickUpArrowRepeat := 0;
    HR^^.HRPressUpArrow := 0;
    HR^^.HRScreenPMenu := 0;
    HR^^.HRScreenPMouse := 0;
    HR^^.HRScreenPKey := 0;
    HR^^.HRLinePMenu := 0;
    HR^^.HRLinePKey := 0;
    HR^^.HRNumViewN := 0;
    HR^^.HRClickDnArrow := 0;
    HR^^.HRClickDnArrowRepeat := 0;
    HR^^.HRPressDnArrow := 0;
    HR^^.HRScreenNMenu := 0;
    HR^^.HRScreenNMouse := 0;
    HR^^.HRScreenNKey := 0;
    HR^^.HRLineNMenu := 0;
    HR^^.HRLineNKey := 0;
    HR^^.HRDragThumb := 0;
    HR^^.HRNumReplace := 0;
    HR^^.HRReplace1 := 0;
    HR^^.HRReplace2 := 0;
    HR^^.HRReplace3FromMenu := 0;
    HR^^.HRReplace3FromPalette := 0;
    HR^^.HRReplace4FromMenu := 0;
    HR^^.HRReplace4FromPalette := 0;
    HR^^.HRReplace5 := 0;
    HR^^.HRReplace6 := 0;
    HR^^.HRReplace7FromMenu := 0;
    HR^^.HRReplace7FromPalette := 0;
    HR^^.HRReplace8FromMenu := 0;
    HR^^.HRReplace8FromPalette := 0;
    HR^^.HRNumDelete := 0;
    HR^^.HRDelete1 := 0;
    HR^^.HRDelete2 := 0;
    HR^^.HRDelete3FromMenu := 0;
    HR^^.HRDelete3FromPalette := 0;
    HR^^.HRDelete4FromMenu := 0;
    HR^^.HRDelete4FromPalette := 0;
    HR^^.HRNumMove := 0;
    HR^^.HRMove1Menu := 0;
    HR^^.HRMove1Key := 0;
    HR^^.HRMove1Palette := 0;
    HR^^.HRMove2Menu := 0;
    HR^^.HRMove2Key := 0;
    HR^^.HRMove2Palette := 0;
    HR^^.HRMove3Menu := 0;
    HR^^.HRMove3Key := 0;
```

```
    HR^^.HRMove3Palette := 0;
    HR^^.HRMove4Menu := 0;
    HR^^.HRMove4Key := 0;
    HR^^.HRMove4Palette := 0;
    HR^^.HRMove5Menu := 0;
    HR^^.HRMove5Key := 0;
    HR^^.HRMove5Palette := 0;
    HR^^.HRMove6Menu := 0;
    HR^^.HRMove6Palette := 0;
    HR^^.HRMove6Key := 0;
    HR^^.HRMove7Menu := 0;
    HR^^.HRMove7Palette := 0;
    HR^^.HRMove8Menu := 0;
    HR^^.HRMove8Palette := 0;
    HR^^.HRNumDuplicate := 0;
    HR^^.HRDuplicate1 := 0;
    HR^^.HRDuplicate2 := 0;
    HR^^.HRDuplicate3Menu := 0;
    HR^^.HRDuplicate3Key := 0;
    HR^^.HRDuplicate3Palette := 0;
    HR^^.HRDuplicate4Menu := 0;
    HR^^.HRDuplicate4Key := 0;
    HR^^.HRDuplicate4Palette := 0;
    HR^^.HRNumCut := 0;
    HR^^.HRCut1Menu := 0;
    HR^^.HRCut1Key := 0;
    HR^^.HRCut1Palette := 0;
    HR^^.HRCut2Menu := 0;
    HR^^.HRCut2Key := 0;
    HR^^.HRCut2Palette := 0;
    HR^^.HRNumCopy := 0;
    HR^^.HRCopy1Menu := 0;
    HR^^.HRCopy1Key := 0;
    HR^^.HRCopy1Palette := 0;
    HR^^.HRCopy2Menu := 0;
    HR^^.HRCopy2Palette := 0;
    HR^^.HRCopy2Key := 0;
    HR^^.HRNumPaste := 0;
    HR^^.HRPaste1Menu := 0;
    HR^^.HRPaste1Key := 0;
    HR^^.HRPaste1Palette := 0;
    HR^^.HRPaste2Menu := 0;
    HR^^.HRPaste2Key := 0;
    HR^^.HRPaste2Palette := 0;
    HR^^.HRNumStyle := 0;
    HR^^.HRText1Menu := 0;
    HR^^.HRText1Palette := 0;
    HR^^.HRText4Menu := 0;
    HR^^.HRText4Palette := 0;
    HR^^.HRText7Menu := 0;
    HR^^.HRText7Palette := 0;
    HR^^.HRNumFont := 0;
    HR^^.HRText2Menu := 0;
    HR^^.HRText2Palette := 0;
    HR^^.HRText5Menu := 0;
    HR^^.HRText5Palette := 0;
    HR^^.HRText8Menu := 0;
    HR^^.HRText8Palette := 0;
    HR^^.HRNumSize := 0;
    HR^^.HRText3Menu := 0;
    HR^^.HRText3Palette := 0;
    HR^^.HRText6Menu := 0;
    HR^^.HRText6Palette := 0;
    HR^^.HRText9Menu := 0;
    HR^^.HRText9Palette := 0;
  end;

  procedure InitHelp;
    var
      HelpDest, HelpView: Rect;
  begin
    HR := HelpHandle(NewHandle(SizeOf(HelpRec)));
    TOC := TOCHandle(NewHandle(SizeOf(TOCRec)));
    InitTOC;
    InitHR;
    TutorPaste := TRUE;
    TutorFontSet := TRUE;
    MustTutorPaste := FALSE;
    MustTutorFont := FALSE;
    SetRect(HelpDest, 650, 40, 800, 475);
    SetRect(HelpView, 650, 40, 800, 475);
    UserHelpProRec := TENew(HelpDest, HelpView);
  end;
end.
```

```
{
  Margaret Stone
  Sc.M. Project, Brown University

  This unit contains the code to write an event to the user profile, and to write the profile to disk.
}

unit HelpFiling;

{
  Interface Section
}

interface

uses
   EditorGlobals, EditorUtilities, EditorHelpInit;

procedure DumpProfile;
procedure EventToJournal;

{
  Implementation Section
}

implementation

{
  DumpProfile writes the user help     profile to disk.
}

   procedure DumpProfile;
      var
         temp:  Str256;
         dummy: boolean;


 Procedure to print parts of the file which do not get indented: }
      procedure DumpParent (parent: integer; temp: str255; tempL: integer);
         var
            str:  Str255;
      begin
         str := '          ';
         NumToString(parent, str);
         TEInsert(@temp[1], tempL, UserHelpProfile);
         TEInsert(@str[1], 5, UserHelpProfile);
         TEKey(CR, UserHelpProfile);
      end;

{Procedure to print parts of the file which do get indented: }
      procedure DumpChild (child: integer; temp: str255; tempL: integer);
         var
            str:  Str255;
      begin
         str := '          ';
         NumToString(child, str);
         TEKey(Tab, UserHelpProfile);
         TEInsert(@temp[1], tempL, UserHelpProfile);
         TEInsert(@str[1], 6, UserHelpProfile);
         TEKey(CR, UserHelpProfile);
      end;


   begin
      HLock(Handle(UserHelpProfile));

{Header}
      temp := 'User Help Profile ';
      TEInsert(@temp[1], 18, UserHelpProfile);
      TEKey(CR, UserHelpProfile);
      TEKey(CR, UserHelpProfile);

{Selects}
      DumpParent(HR^^.HRnumSelects, 'Total Selects:  ', 16);
      if HR^^.HRSelectFromMenu <> 0 then
         DumpChild(HR^^.HRSelectFromMenu, 'Menu Selects:  ', 15);
      if HR^^.HRSelectFromMouse <> 0 then
         DumpChild(HR^^.HRSelectFromMouse, 'Mouse Selects:  ', 16);
      if HR^^.HRSelectFromKey <> 0 then
         DumpChild(HR^^.HRSelectFromKey, 'Key Selects:  ', 14);
      TEKey(CR, UserHelpProfile);

{View Previous}
      DumpParent(HR^^.HRnumViewP, 'Total Previous Views:  ', 23);
      if HR^^.HRClickUpArrow <> 0 then
         DumpChild(HR^^.HRClickUpArrow, 'Click Up Arrows:  ', 18);
      if HR^^.HRClickUpArrowRepeat <> 0 then
         DumpChild(HR^^.HRClickUpArrowRepeat, 'Click Up Arrow Repeats:  ', 25);
      if HR^^.HRPressUpArrow <> 0 then
         DumpChild(HR^^.HRPressUpArrow, 'Press Up Arrows:  ', 18);
      if HR^^.HRScreenPMenu <> 0 then
         DumpChild(HR^^.HRScreenPMenu, 'Screen Previous From Menu:  ', 28);
      if HR^^.HRScreenPMouse <> 0 then
         DumpChild(HR^^.HRScreenPMouse, 'Screen Previous From Mouse:  ', 29);
      if HR^^.HRScreenPKey <> 0 then
         DumpChild(HR^^.HRScreenPKey, 'Screen Previous By Key:  ', 26);
      if HR^^.HRLinePMenu <> 0 then
         DumpChild(HR^^.HRLinePMenu, 'Line Previous From Menu:  ', 28);
      if HR^^.HRLinePKey <> 0 then
         DumpChild(HR^^.HRLinePKey, 'Line Previous By Key:  ', 23);
      TEKey(CR, UserHelpProfile);

{View Next}
      DumpParent(HR^^.HRnumViewN, 'Total Next Views:  ', 19);
      if HR^^.HRClickDnArrow <> 0 then
         DumpChild(HR^^.HRClickDnArrow, 'Click Down Arrows:  ', 20);
      if HR^^.HRClickDnArrowRepeat <> 0 then
         DumpChild(HR^^.HRClickDnArrowRepeat, 'Click Down Arrow Repeats:  ', 27);
```

```
    if HR^^.HRPressOnArrow <> 0 then
      DumpChild(HR^^.HRPressOnArrow, 'Press Down Arrows:  ', 20);
    if HR^^.HRScreen4Menu <> 0 then
      DumpChild(HR^^.HRScreen4Menu, 'Screen Next From Menu:  ', 24);
    if HR^^.HRScreen4Mouse <> 0 then
      DumpChild(HR^^.HRScreen4Mouse, 'Screen Next From Mouse:  ', 25);
    if HR^^.HRScreen4Key <> 0 then
      DumpChild(HR^^.HRScreen4Key, 'Screen Next By Key:  ', 21);
    if HR^^.HRLine4Menu <> 0 then
      DumpChild(HR^^.HRLine4Menu, 'Line Next From Menu:  ', 22);
    if HR^^.HRLine4Key <> 0 then
      DumpChild(HR^^.HRLine4Key, 'Line Next By Key:  ', 19);
    TEKey(CR, UserHelpProBe);

  {General View:}
    DumpParent(HR^^.HRDragThumb, 'Thumb Drags:  ', 14);
    TEKey(CR, UserHelpProBe);

  {Replace:}
    DumpParent(HR^^.HRNumReplace, 'Total Replaces:  ', 17);
    if HR^^.HRReplace1 <> 0 then
      DumpChild(HR^^.HRReplace1, 'Delete and Retype:  ', 20);
    if HR^^.HRReplace2 <> 0 then
      DumpChild(HR^^.HRReplace2, 'Select, Press Delete, and Retype:  ', 34);
    if HR^^.HRReplace3FromMenu <> 0 then
      DumpChild(HR^^.HRReplace3FromMenu, 'Select, Choose Delete From the Menu, and Retype:  ', 49);
    if HR^^.HRReplace3FromPalette <> 0 then
      DumpChild(HR^^.HRReplace3FromPalette, 'Select, Choose Delete From the Palette, and Retype:  ', 52);
    if HR^^.HRReplace4FromMenu <> 0 then
      DumpChild(HR^^.HRReplace4FromMenu, 'Choose Delete From the Menu, Select, and Retype:  ', 49);
    if HR^^.HRReplace4FromPalette <> 0 then
      DumpChild(HR^^.HRReplace4FromPalette, 'Choose Delete From the Palette, Select, and Retype:  ', 52);
    if HR^^.HRReplace6 <> 0 then
      DumpChild(HR^^.HRReplace6, 'Cut and Retype:  ', 17);
    if HR^^.HRReplace6 <> 0 then
      DumpChild(HR^^.HRReplace6, 'Select and Retype:  ', 20);
    if HR^^.HRReplace7FromMenu <> 0 then
      DumpChild(HR^^.HRReplace7FromMenu, 'Select, Choose Replace From the Menu:  ', 39);
    if HR^^.HRReplace7FromPalette <> 0 then
      DumpChild(HR^^.HRReplace7FromPalette, 'Select, Choose Replace From the Palette:  ', 42);
    if HR^^.HRReplace8FromMenu <> 0 then
      DumpChild(HR^^.HRReplace8FromMenu, 'Choose Replace From the Menu, Etc:  ', 36);
    if HR^^.HRReplace8FromPalette <> 0 then
      DumpChild(HR^^.HRReplace8FromPalette, 'Choose Replace From the Palette, Etc:  ', 39);
    TEKey(CR, UserHelpProBe);

  {Deletes:}
    DumpParent(HR^^.HRNumDelete, 'Total Deletes:  ', 16);
    if HR^^.HRDelete1 <> 0 then
      DumpChild(HR^^.HRDelete1, 'Move IP and Press Delete:  ', 27);
    if HR^^.HRDelete2 <> 0 then
      DumpChild(HR^^.HRDelete2, 'Select and Press Delete:  ', 26);
    if HR^^.HRDelete3FromMenu <> 0 then
      DumpChild(HR^^.HRDelete3FromMenu, 'Select and Choose Delete From Menu:  ', 37);
    if HR^^.HRDelete3FromPalette <> 0 then
      DumpChild(HR^^.HRDelete3FromPalette, 'Select and Choose Delete From Palette:  ', 40);
    if HR^^.HRDelete4FromMenu <> 0 then
      DumpChild(HR^^.HRDelete4FromMenu, 'Choose Delete From Menu, then Select, Etc:  ', 44);
    if HR^^.HRDelete4FromPalette <> 0 then
      DumpChild(HR^^.HRDelete4FromPalette, 'Choose Delete From Palette, then Select, Etc:  ', 47);
    TEKey(CR, UserHelpProBe);

  {Moves:}
    DumpParent(HR^^.HRNumMove, 'Total Moves:  ', 14);
    if HR^^.HRMove1Menu <> 0 then
      DumpChild(HR^^.HRMove1Menu, 'Cut, Move IP, Paste From Menu:  ', 32);
    if HR^^.HRMove1Key <> 0 then
      DumpChild(HR^^.HRMove1Key, 'Cut, Move IP, Paste By Key:  ', 29);
    if HR^^.HRMove1Palette <> 0 then
      DumpChild(HR^^.HRMove1Palette, 'Cut, Move IP, Paste From Palette:  ', 35);
    if HR^^.HRMove2Menu <> 0 then
      DumpChild(HR^^.HRMove2Menu, 'Cut, Move IP,Native Paste From Menu:  ', 37);
    if HR^^.HRMove2Key <> 0 then
      DumpChild(HR^^.HRMove2Key, 'Cut, Move IP,Native Paste By Key:  ', 34);
    if HR^^.HRMove2Palette <> 0 then
      DumpChild(HR^^.HRMove2Palette, 'Cut, Move IP,Native Paste From Palette:  ', 38);
    if HR^^.HRMove3Menu <> 0 then
      DumpChild(HR^^.HRMove3Menu, 'Copy, Press Delete, Move IP, Paste From Menu:  ', 47);
    if HR^^.HRMove3Key <> 0 then
      DumpChild(HR^^.HRMove3Key, 'Copy, Press Delete, Move IP, Paste By Key:  ', 44);
    if HR^^.HRMove3Palette <> 0 then
      DumpChild(HR^^.HRMove3Palette, 'Copy, Press Delete, Move IP, Paste From Palette:  ', 50);
    if HR^^.HRMove4Menu <> 0 then
      DumpChild(HR^^.HRMove4Menu, 'Copy, Press Delete, Move IP,Native Paste From Menu:  ', 52);
    if HR^^.HRMove4Key <> 0 then
      DumpChild(HR^^.HRMove4Key, 'Copy, Press Delete, Move IP,Native Paste By Key:  ', 49);
    if HR^^.HRMove4Palette <> 0 then
      DumpChild(HR^^.HRMove4Palette, 'Copy, Press Delete, Move IP,Native Paste From Palette:  ', 55);
    if HR^^.HRMove5Menu <> 0 then
      DumpChild(HR^^.HRMove5Menu, 'Copy, Choose Delete, Move IP, Paste From Menu:  ', 48);
    if HR^^.HRMove5Key <> 0 then
      DumpChild(HR^^.HRMove5Key, 'Copy, Choose Delete, Move IP, Paste By Key:  ', 45);
    if HR^^.HRMove5Palette <> 0 then
      DumpChild(HR^^.HRMove5Palette, 'Copy, Choose Delete, Move IP, Paste From Palette:  ', 51);
    if HR^^.HRMove6Menu <> 0 then
      DumpChild(HR^^.HRMove6Menu, 'Copy, Choose Delete, Move IP,Native Paste From Menu:  ', 53);
    if HR^^.HRMove6Palette <> 0 then
      DumpChild(HR^^.HRMove6Palette, 'Copy, Choose Delete, Move IP,Native Paste From Palette:  ', 56);
    if HR^^.HRMove6Key <> 0 then
      DumpChild(HR^^.HRMove6Key, 'Copy, Choose Delete, Move IP,Native Paste By Key:  ', 50);
    if HR^^.HRMove7Menu <> 0 then
      DumpChild(HR^^.HRMove7Menu, 'Select, Choose Move From Menu:  ', 31);
    if HR^^.HRMove7Palette <> 0 then
      DumpChild(HR^^.HRMove7Palette, 'Select, Choose Move From Palette:  ', 33);
    if HR^^.HRMove8Menu <> 0 then
      DumpChild(HR^^.HRMove8Menu, 'Choose Move From Menu, Select, Etc:  ', 36);
    if HR^^.HRMove8Palette <> 0 then
      DumpChild(HR^^.HRMove8Palette, 'Choose Move From Palette, Select, Etc:  ', 39);
```

```
    TEKey(CR, UserHelpProfile);

:Duplicates:)
    DumpParent(HR^^.HRnumDuplicate, 'Total Duplicates: ', 19);
    if HR^^.HRDuplicate1 <> 0 then
      DumpChild(HR^^.HRDuplicate1, 'Select, Choose Duplicate, Etc: ', 32);
    if HR^^.HRDuplicate2 <> 0 then
      DumpChild(HR^^.HRDuplicate2, 'Choose Duplicate, Select, Etc: ', 32);
    if HR^^.HRDuplicate3Menu <> 0 then
      DumpChild(HR^^.HRDuplicate3Menu, 'TOC, Move IP, Paste From Menu: ', 32);
    if HR^^.HRDuplicate3Key <> 0 then
      DumpChild(HR^^.HRDuplicate3Key, 'TOC, Move IP, Paste From Key: ', 31);
    if HR^^.HRDuplicate3Palette <> 0 then
      DumpChild(HR^^.HRDuplicate3Palette, 'TOC, Move IP, Paste From Palette: ', 36);
    if HR^^.HRDuplicate4Menu <> 0 then
      DumpChild(HR^^.HRDuplicate4Menu, 'TOC, Move IP,Naive Paste From Menu: ', 37);
    if HR^^.HRDuplicate4Key <> 0 then
      DumpChild(HR^^.HRDuplicate4Key, 'TOC, Move IP,Naive Paste From Key: ', 36);
    if HR^^.HRDuplicate4Palette <> 0 then
      DumpChild(HR^^.HRDuplicate4Palette, 'TOC, Move IP,Naive Paste From Palette: ', 40);
    TEKey(CR, UserHelpProfile);

Cuts:)
    DumpParent(HR^^.HRnumCut, 'Total Cuts: ', 11);
    if HR^^.HRCut1Menu <> 0 then
      DumpChild(HR^^.HRCut1Menu, 'Cuts from Menu: ', 17);
    if HR^^.HRCut1Key <> 0 then
      DumpChild(HR^^.HRCut1Key, 'Cuts from Key: ', 16);
    if HR^^.HRCut1Palette <> 0 then
      DumpChild(HR^^.HRCut1Palette, 'Cuts from Palette: ', 20);
    if HR^^.HRCut2Menu <> 0 then
      DumpChild(HR^^.HRCut2Menu, 'Naive Cuts from Menu: ', 23);
    if HR^^.HRCut2Key <> 0 then
      DumpChild(HR^^.HRCut2Key, 'Naive Cuts from Key: ', 22);
    if HR^^.HRCut2Palette <> 0 then
      DumpChild(HR^^.HRCut2Palette, 'Naive Cuts from Palette: ', 26);
    TEKey(CR, UserHelpProfile);

Copies:)
    DumpParent(HR^^.HRnumCopy, 'Total Copies: ', 15);
    if HR^^.HRCopy1Menu <> 0 then
      DumpChild(HR^^.HRCopy1Menu, 'Copies from Menu: ', 19);
    if HR^^.HRCopy1Key <> 0 then
      DumpChild(HR^^.HRCopy1Key, 'Copies from Key: ', 18);
    if HR^^.HRCopy1Palette <> 0 then
      DumpChild(HR^^.HRCopy1Palette, 'Copies from Palette: ', 22);
    if HR^^.HRCopy2Menu <> 0 then
      DumpChild(HR^^.HRCopy2Menu, 'Naive Copies from Menu: ', 25);
    if HR^^.HRCopy2Key <> 0 then
      DumpChild(HR^^.HRCopy2Key, 'Naive Copies from Key: ', 24);
    if HR^^.HRCopy2Palette <> 0 then
      DumpChild(HR^^.HRCopy2Palette, 'Naive Copies from Palette: ', 28);
    TEKey(CR, UserHelpProfile);

:Pastes:)
    DumpParent(HR^^.HRnumPaste, 'Total Pastes: ', 15);
    if HR^^.HRPaste1Menu <> 0 then
      DumpChild(HR^^.HRPaste1Menu, 'Pastes from Menu: ', 19);
    if HR^^.HRPaste1Key <> 0 then
      DumpChild(HR^^.HRPaste1Key, 'Pastes from Key: ', 18);
    if HR^^.HRPaste1Palette <> 0 then
      DumpChild(HR^^.HRPaste1Palette, 'Pastes from Palette: ', 22);
    if HR^^.HRPaste2Menu <> 0 then
      DumpChild(HR^^.HRPaste2Menu, 'Naive Pastes from Menu: ', 25);
    if HR^^.HRPaste2Key <> 0 then
      DumpChild(HR^^.HRPaste2Key, 'Naive Pastes from Key: ', 24);
    if HR^^.HRPaste2Palette <> 0 then
      DumpChild(HR^^.HRPaste2Palette, 'Naive Pastes from Palette: ', 28);
    TEKey(CR, UserHelpProfile);

Size Changes:)
    DumpParent(HR^^.HRnumSize, 'Total Size Changes: ', 21);
    if HR^^.HRText1Menu <> 0 then
      DumpChild(HR^^.HRText1Menu, 'Select Text and Choose a Size from Menu: ', 42);
    if HR^^.HRText1Palette <> 0 then
      DumpChild(HR^^.HRText1Palette, 'Select Text and Choose a Size from Palette: ', 45);
    if HR^^.HRText4Menu <> 0 then
      DumpChild(HR^^.HRText4Menu, 'Choose a Size from Menu and Type: ', 36);
    if HR^^.HRText4Palette <> 0 then
      DumpChild(HR^^.HRText4Palette, 'Choose a Size from Palette and Type: ', 38);
    if HR^^.HRText7Menu <> 0 then
      DumpChild(HR^^.HRText7Menu, 'Naive Size Change from Menu: ', 30);
    if HR^^.HRText7Palette <> 0 then
      DumpChild(HR^^.HRText7Palette, 'Naive Size Change from Palette: ', 33);
    TEKey(CR, UserHelpProfile);

Font Changes:)
    DumpParent(HR^^.HRnumFont, 'Total Font Changes: ', 21);
    if HR^^.HRText2Menu <> 0 then
      DumpChild(HR^^.HRText2Menu, 'Select Text and Choose a Font from Menu: ', 42);
    if HR^^.HRText2Palette <> 0 then
      DumpChild(HR^^.HRText2Palette, 'Select Text and Choose a Font from Palette: ', 45);
    if HR^^.HRText5Menu <> 0 then
      DumpChild(HR^^.HRText5Menu, 'Choose a Font from Menu and Type: ', 36);
    if HR^^.HRText5Palette <> 0 then
      DumpChild(HR^^.HRText5Palette, 'Choose a Font from Palette and Type: ', 38);
    if HR^^.HRText8Menu <> 0 then
      DumpChild(HR^^.HRText8Menu, 'Naive Font Change from Menu: ', 30);
    if HR^^.HRText8Palette <> 0 then
      DumpChild(HR^^.HRText8Palette, 'Naive Font Change from Palette: ', 33);
    TEKey(CR, UserHelpProfile);

Style Changes:)
    DumpParent(HR^^.HRnumStyle, 'Total Style Changes: ', 22);
    if HR^^.HRText3Menu <> 0 then
      DumpChild(HR^^.HRText3Menu, 'Select Text and Choose a Style from Menu: ', 43);
    if HR^^.HRText3Palette <> 0 then
      DumpChild(HR^^.HRText3Palette, 'Select Text and Choose a Style from Palette: ', 46);
```

```
  if HR^^.HRTextStMenu <> 0 then
    DumpChild(HR^^.HRTextStMenu, 'Choose a Style from Menu and Type: ', 36);
  if HR^^.HRTextStPalette <> 0 then
    DumpChild(HR^^.HRTextStPalette, 'Choose a Style from Palette and Type: ', 39);
  if HR^^.HRTextStMenu <> 0 then
    DumpChild(HR^^.HRTextStMenu, 'Native Style Change from Menu: ', 31);
  if HR^^.HRTextStPalette <> 0 then
    DumpChild(HR^^.HRTextStPalette, 'Native Style Change from Palette: ', 34);

  dummy := WriteFile('Help Profile', 0, UserHelpProfile);
  HUnLock(Handle(UserHelpProfile));
end;


{-------------------------------------------------------------------------------}
{ EventToJournal writes the latest event to the user journal of word processor events, then prints }
{ the journal to disk (for debugging/testing purposes).                         }
{-------------------------------------------------------------------------------}

procedure EventToJournal;
  var
    theText, theFrom: str255;
    theLength, theFromLength: integer;
    dummy: boolean;
begin

  theText := '';
  theFrom := '';
  theLength := 0;
  theFromLength := 0;

  case WPER^^.theEvent of
    Select:
      begin
        theText := 'Select ';
        theLength := 7
      end;
    ClickDnArrow:
      begin
        theText := 'ClickDnArrow ';
        theLength := 13
      end;
    ClickUpArrow:
      begin
        theText := 'ClickUpArrow ';
        theLength := 13
      end;
    ClickDnArrowRepeat:
      begin
        theText := 'ClickDnArrowRepeat ';
        theLength := 18
      end;
    ClickUpArrowRepeat:
      begin
        theText := 'ClickUpArrowRepeat ';
        theLength := 18
      end;
    ScreenP:
      begin
        theText := 'ScreenP ';
        theLength := 7
      end;
    ScreenN:
      begin
        theText := 'ScreenN ';
        theLength := 7
      end;
    LineP:
      begin
        theText := 'LineP ';
        theLength := 6
      end;
    LineN:
      begin
        theText := 'LineN ';
        theLength := 6
      end;
    PressDnArrow:
      begin
        theText := 'PressDnArrow ';
        theLength := 13
      end;
    PressUpArrow:
      begin
        theText := 'PressUpArrow ';
        theLength := 13
      end;
    PressOK:
      begin
        theText := 'PressOK ';
        theLength := 8
      end;
    PressCancel:
      begin
        theText := 'PressCancel ';
        theLength := 12
      end;
    DragThumb:
      begin
        theText := 'DragThumb ';
        theLength := 10
      end;
    SizeSet:
      begin
        theText := 'SizeSet ';
        theLength := 8
      end;
    SizeChange:
```

```
      begin
        theText := 'SizeChange ';
        theLength := 11
      end;
    FontSet:
      begin
        theText := 'FontSet ';
        theLength := 8
      end;
    FontChange:
      begin
        theText := 'FontChange ';
        theLength := 11
      end;
    StyleSet:
      begin
        theText := 'StyleSet ';
        theLength := 9
      end;
    StyleChange:
      begin
        theText := 'StyleChange ';
        theLength := 12
      end;
    Replace:
      begin
        theText := 'Replace ';
        theLength := 8
      end;
    Delete:
      begin
        theText := 'Delete ';
        theLength := 7
      end;
    Move:
      begin
        theText := 'Move ';
        theLength := 5
      end;
    Copy:
      begin
        theText := 'Copy ';
        theLength := 5
      end;
    Cut:
      begin
        theText := 'Cut ';
        theLength := 4
      end;
    Paste:
      begin
        theText := 'Paste ';
        theLength := 6
      end;
    Duplicate:
      begin
        theText := 'Duplicate ';
        theLength := 10
      end;
    UserType:
      begin
        theText := 'UserType ';
        theLength := 9
      end;
    DeleteType:
      begin
        theText := 'DeleteType ';
        theLength := 11
      end;
    MoveIP:
      begin
        theText := 'MoveIP ';
        theLength := 7
      end;
    otherwise
  end;

  case WPER^^ from of
    Palette:
      begin
        theFrom := 'From:  Palette ';
        theFromLength := 14
      end;
    Mouse:
      begin
        theFrom := 'From:  Mouse ';
        theFromLength := 12
      end;
    Menu:
      begin
        theFrom := 'From:  Menu ';
        theFromLength := 11
      end;
    Key:
      begin
        theFrom := 'From:  Key ';
        theFromLength := 10
      end;
    otherwise
      begin
        theFrom := 'From:  N/A ';
        theFromLength := 10
      end;
  end;

  TEInsert(@theText[1], theLength, WPEventJournal);
  TEKey(CR, WPEventJournal);
```

```
        TEInsert(@theFrom[1], theFrom.Length, WPEventJournal);
        TEKey(CR, WPEventJournal);

        if (WPER^^.theEvent <> userType) and (WPER^^.theEvent <> deleteType) then
            TEKey(CR, WPEventJournal);

        dummy := WriteFile('Journal', 0, WPEventJournal);
    end;
end.
```

```
{------------------------------------------------------------------------------}
{ Margaret Stone                                                               }
{ Sc.M. Project, Brown University                                              }
{                                                                              }
{ This unit contains the code to determine from the user help profile the user's most often-used method }
{ of performing a given task.                                                  }
{------------------------------------------------------------------------------}

unit HelpBests;

{------------------------------------------------------------------------------}
{ Interface Section                                                            }
{------------------------------------------------------------------------------}

interface

  uses
    EditorGlobals, EditorHelpInt;

  function bestSelect: ProfileType;
  function bestMove: ProfileType;
  function bestReplace: ProfileType;
  function bestReplace81: ProfileType;
  function bestDelete: ProfileType;
  function bestDelete84: ProfileType;
  function bestMove: ProfileType;
  function bestMove78: ProfileType;
  function bestDuplicate: ProfileType;
  function bestDuplicate12: ProfileType;
  function bestCut: ProfileType;
  function bestCopy: ProfileType;
  function bestPaste: ProfileType;
  function bestSize: ProfileType;
  function bestFont: ProfileType;
  function bestStyle: ProfileType;
  function menuPref: boolean;

{------------------------------------------------------------------------------}
{ Implementation Section                                                       }
{------------------------------------------------------------------------------}

implementation

{------------------------------------------------------------------------------}
{ MenuPref is a last-resort sort of procedure, which determines if the user has a preference for the }
{ menu over the palette.                                                       }
{------------------------------------------------------------------------------}

  function menuPref;              {'boolean}
    var
      numMenu, numPalette, total: integer;
      temp, max: real;
  begin
    numMenu := 0;
    numPalette := 0;
    total := 0;
    max := 0;
    temp := 0;
    if HR^^.HRnumSelects <> 0 then
      begin
        if HR^^.HRSelectFromMenu <> 0 then
          begin
            numMenu := numMenu + HR^^.HRSelectFromMenu;
            total := total + HR^^.HRSelectFromMenu
          end
      end;
    if HR^^.HRnumReplace <> 0 then
      begin
        if HR^^.HRReplace3FromMenu <> 0 then
          begin
            numMenu := numMenu + HR^^.HRReplace3FromMenu;
            total := total + HR^^.HRReplace3FromMenu
          end;
        if HR^^.HRReplace3FromPalette <> 0 then
          begin
            numPalette := numPalette + HR^^.HRReplace3FromPalette;
            total := total + HR^^.HRReplace3FromPalette
          end;
        if HR^^.HRReplace4FromMenu <> 0 then
          begin
            numMenu := numMenu + HR^^.HRReplace4FromMenu;
            total := total + HR^^.HRReplace4FromMenu
          end;
        if HR^^.HRReplace4FromPalette <> 0 then
          begin
            numPalette := numPalette + HR^^.HRReplace4FromPalette;
            total := total + HR^^.HRReplace4FromPalette
          end;
        if HR^^.HRReplace7FromMenu <> 0 then
          begin
            numMenu := numMenu + HR^^.HRReplace7FromMenu;
            total := total + HR^^.HRReplace7FromMenu
          end;
        if HR^^.HRReplace7FromPalette <> 0 then
          begin
            numPalette := numPalette + HR^^.HRReplace7FromPalette;
            total := total + HR^^.HRReplace7FromPalette
          end;
        if HR^^.HRReplace8FromMenu <> 0 then
          begin
            numMenu := numMenu + HR^^.HRReplace8FromMenu;
            total := total + HR^^.HRReplace8FromMenu
          end;
        if HR^^.HRReplace8FromPalette <> 0 then
          begin
            numPalette := numPalette + HR^^.HRReplace8FromPalette;
            total := total + HR^^.HRReplace8FromPalette
          end;
```

```
    end;
if HR^^.HRnumCut <> 0 then
    begin
        if HR^^.HRCut1Palette <> 0 then
            begin
                numPalette := numPalette + HR^^.HRCut1Palette;
                total := total + HR^^.HRCut1Palette
            end;
        if HR^^.HRCut2Palette <> 0 then
            begin
                numPalette := numPalette + HR^^.HRCut2Palette;
                total := total + HR^^.HRCut2Palette
            end;
        if HR^^.HRCut1Menu <> 0 then
            begin
                numMenu := numMenu + HR^^.HRCut1Menu;
                total := total + HR^^.HRCut1Menu
            end;
        if HR^^.HRCut2Menu <> 0 then
            begin
                numMenu := numMenu + HR^^.HRCut2Key;
                total := total + HR^^.HRCut2Key
            end;
        if HR^^.HRCut1Key <> 0 then
            begin
                numMenu := numMenu + HR^^.HRCut1Menu;
                total := total + HR^^.HRCut1Menu
            end;
        if HR^^.HRCut2Key <> 0 then
            begin
                numMenu := numMenu + HR^^.HRCut2Key;
                total := total + HR^^.HRCut2Key
            end;
    end;
if HR^^.HRnumCopy <> 0 then
    begin
        if HR^^.HRCopy1Palette <> 0 then
            begin
                numPalette := numPalette + HR^^.HRCopy1Palette;
                total := total + HR^^.HRCopy1Palette
            end;
        if HR^^.HRCopy2Palette <> 0 then
            begin
                numPalette := numPalette + HR^^.HRCopy2Palette;
                total := total + HR^^.HRCopy2Palette
            end;
        if HR^^.HRCopy1Menu <> 0 then
            begin
                numMenu := numMenu + HR^^.HRCopy1Menu;
                total := total + HR^^.HRCopy1Menu
            end;
        if HR^^.HRCopy2Menu <> 0 then
            begin
                numMenu := numMenu + HR^^.HRCopy2Key;
                total := total + HR^^.HRCopy2Key
            end;
        if HR^^.HRCopy1Key <> 0 then
            begin
                numMenu := numMenu + HR^^.HRCopy1Menu;
                total := total + HR^^.HRCopy1Menu
            end;
        if HR^^.HRCopy2Key <> 0 then
            begin
                numMenu := numMenu + HR^^.HRCopy2Key;
                total := total + HR^^.HRCopy2Key
            end;
    end;                    <-
if HR^^.HRnumPaste <> 0 then
    begin
        if HR^^.HRPaste1Palette <> 0 then
            begin
                numPalette := numPalette + HR^^.HRPaste1Palette;
                total := total + HR^^.HRPaste1Palette
            end;
        if HR^^.HRPaste2Palette <> 0 then
            begin
                numPalette := numPalette + HR^^.HRPaste2Palette;
                total := total + HR^^.HRPaste2Palette
            end;
        if HR^^.HRPaste1Menu <> 0 then
            begin
                numMenu := numMenu + HR^^.HRPaste1Menu;
                total := total + HR^^.HRPaste1Menu
            end;
        if HR^^.HRPaste2Menu <> 0 then
            begin
                numMenu := numMenu + HR^^.HRPaste2Key;
                total := total + HR^^.HRPaste2Key
            end;
        if HR^^.HRPaste1Key <> 0 then
            begin
                numMenu := numMenu + HR^^.HRPaste1Menu;
                total := total + HR^^.HRPaste1Menu
            end;
        if HR^^.HRPaste2Key <> 0 then
            begin
                numMenu := numMenu + HR^^.HRPaste2Key;
                total := total + HR^^.HRPaste2Key
            end;
    end;
if HR^^.HRnumSize <> 0 then
    begin
        if HR^^.HRText1Palette <> 0 then
            begin
                numPalette := numPalette + HR^^.HRText1Palette;
                total := total + HR^^.HRText1Palette
            end;
        if HR^^.HRText1Palette <> 0 then
```

```
            begin
               numPalette := numPalette + HR^^.HRText4Palette;
               total := total + HR^^.HRText4Palette
            end;
         if HR^^.HRText7Palette <> 0 then
            begin
               numPalette := numPalette + HR^^.HRText7Palette;
               total := total + HR^^.HRText7Palette
            end;
         if HR^^.HRText1Menu <> 0 then
            begin
               numMenu := numMenu + HR^^.HRText1Menu;
               total := total + HR^^.HRText1Menu
            end;
         if HR^^.HRText4Menu <> 0 then
            begin
               numMenu := numMenu + HR^^.HRText4Menu;
               total := total + HR^^.HRText4Menu
            end;
         if HR^^.HRText7Menu <> 0 then
            begin
               numMenu := numMenu + HR^^.HRText7Menu;
               total := total + HR^^.HRText7Menu
            end;
      end;
   if HR^^.HRnumFont <> 0 then
      begin
         if HR^^.HRText2Palette <> 0 then
            begin
               numPalette := numPalette + HR^^.HRText2Palette;
               total := total + HR^^.HRText2Palette
            end;
         if HR^^.HRText5Palette <> 0 then
            begin
               numPalette := numPalette + HR^^.HRText5Palette;
               total := total + HR^^.HRText5Palette
            end;
         if HR^^.HRText8Palette <> 0 then
            begin
               numPalette := numPalette + HR^^.HRText8Palette;
               total := total + HR^^.HRText8Palette
            end;
         if HR^^.HRText2Menu <> 0 then
            begin
               numMenu := numMenu + HR^^.HRText2Menu;
               total := total + HR^^.HRText2Menu
            end;
         if HR^^.HRText5Menu <> 0 then
            begin
               numMenu := numMenu + HR^^.HRText5Menu;
               total := total + HR^^.HRText5Menu
            end;
         if HR^^.HRText8Menu <> 0 then
            begin
               numMenu := numMenu + HR^^.HRText8Menu;
               total := total + HR^^.HRText8Menu
            end;
      end;
   if HR^^.HRnumStyle <> 0 then
      begin
         if HR^^.HRText3Palette <> 0 then
            begin
               numPalette := numPalette + HR^^.HRText3Palette;
               total := total + HR^^.HRText3Palette
            end;
         if HR^^.HRText6Palette <> 0 then
            begin
               numPalette := numPalette + HR^^.HRText6Palette;
               total := total + HR^^.HRText6Palette
            end;
         if HR^^.HRText9Palette <> 0 then
            begin
               numPalette := numPalette + HR^^.HRText9Palette;
               total := total + HR^^.HRText9Palette
            end;
         if HR^^.HRText3Menu <> 0 then
            begin
               numMenu := numMenu + HR^^.HRText3Menu;
               total := total + HR^^.HRText3Menu
            end;
         if HR^^.HRText6Menu <> 0 then
            begin
               numMenu := numMenu + HR^^.HRText6Menu;
               total := total + HR^^.HRText6Menu
            end;
         if HR^^.HRText9Menu <> 0 then
            begin
               numMenu := numMenu + HR^^.HRText9Menu;
               total := total + HR^^.HRText9Menu
            end;
      end;
   if HR^^.HRnumDelete <> 0 then
      begin
         if HR^^.HRDelete3FromMenu <> 0 then
            begin
               numMenu := numMenu + HR^^.HRDelete3FromMenu;
               total := total + HR^^.HRDelete3FromMenu
            end;
         if HR^^.HRDelete4FromMenu <> 0 then
            begin
               numMenu := numMenu + HR^^.HRDelete4FromMenu;
               total := total + HR^^.HRDelete4FromMenu
            end;
         if HR^^.HRDelete3FromPalette <> 0 then
            begin
               numPalette := numPalette + HR^^.HRDelete3FromPalette;
               total := total + HR^^.HRDelete3FromPalette
            end;
```

```
            if HR^^.HRDelete4FromPalette <> 0 then
               begin
                  numPalette := numPalette + HR^^.HRDelete4FromPalette;
                  total := total + HR^^.HRDelete4FromPalette
               end;
         end;
      if HR^^.HRnumMove <> 0 then
         begin
            if HR^^.HRMove7Palette <> 0 then
               begin
                  numPalette := numPalette + HR^^.HRMove7Palette;
                  total := total + HR^^.HRMove7Palette
               end;
            if HR^^.HRMove8Palette <> 0 then
               begin
                  numPalette := numPalette + HR^^.HRMove8Palette;
                  total := total + HR^^.HRMove8Palette
               end;
            if HR^^.HRMove7Menu <> 0 then
               begin
                  numMenu := numMenu + HR^^.HRMove7Menu;
                  total := total + HR^^.HRMove7Menu
               end;
            if HR^^.HRMove8Menu <> 0 then
               begin
                  numMenu := numMenu + HR^^.HRMove8Menu;
                  total := total + HR^^.HRMove8Menu
               end;
         end;
      if HR^^.HRnumDuplicate <> 0 then
         begin
            if HR^^.HRDuplicate1 <> 0 then
               begin
                  numMenu := numMenu + HR^^.HRDuplicate1;
                  total := total + HR^^.HRDuplicate1
               end;
            if HR^^.HRDuplicate2 <> 0 then
               begin
                  numMenu := numMenu + HR^^.HRDuplicate2;
                  total := total + HR^^.HRDuplicate2
               end;
         end;

      if total <> 0 then
         begin
            if numPalette <> 0 then
               max := numPalette / total;
            if numMenu <> 0 then
               temp := numMenu / total;
            if temp > max then
               menuPref := TRUE
            else
               menuPref := FALSE
         end
      else
         menuPref := FALSE
   end;

{-------------------------------------------------------------------------------}
{ bestSelect determines the user's most commonly used procedure for selecting.  }
{-------------------------------------------------------------------------------}

   function bestSelect          : ProfileType;
      var
         bestMethod: ProfileType;
         best, temp: real;
   begin
      bestMethod := H_neverTried;
      temp := 0;
      best := 0;

      if HR^^.HRnumSelects <> 0 then
         begin
            if HR^^.HRSelectFromMenu <> 0 then
               begin
                  temp := HR^^.HRSelectFromMenu / HR^^.HRnumSelects;
                  if temp > best then
                     begin
                        best := temp;
                        bestMethod := H_SelectMenu;
                     end
               end;
            if HR^^.HRSelectFromMouse <> 0 then
               begin
                  temp := HR^^.HRSelectFromMouse / HR^^.HRnumSelects;
                  if temp > best then
                     begin
                        best := temp;
                        bestMethod := H_SelectMouse
                     end
               end;
            if HR^^.HRSelectFromKey <> 0 then
               begin
                  temp := HR^^.HRSelectFromKey / HR^^.HRnumSelects;
                  if temp > best then
                     begin
                        best := temp;
                        bestMethod := H_SelectKey
                     end
               end;
         end;
      bestSelect := bestMethod;
   end;

{-------------------------------------------------------------------------------}
{ bestSelect determines the user's most commonly used procedure for viewing.    }
{-------------------------------------------------------------------------------}
```

```
function bestView:    ; ProfileType)
   var
      bestMethod: ProfileType;
      best, temp: real;
      total: integer;
begin
   bestMethod := H_neverTried;
   temp := 0;
   best := 0;
   total := HR^^.HRnumViewsP + HR^^.HRnumViewsN + HR^^.HRDragThumb;

   if HR^^.HRnumViewsP <> 0 then
      begin
         if HR^^.HRClickUpArrow <> 0 then
            begin
               temp := HR^^.HRClickUpArrow / total;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_ClickUpArrow;
                  end
            end;
         if HR^^.HRClickUpArrowRepeat <> 0 then
            begin
               temp := HR^^.HRClickUpArrowRepeat / total;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_ClickUpArrowRepeat;
                  end
            end;
         if HR^^.HRPressUpArrow <> 0 then
            begin
               temp := HR^^.HRPressUpArrow / total;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_PressUpArrow;
                  end
            end;
         if HR^^.HRScreenPMouse <> 0 then
            begin
               temp := HR^^.HRScreenPMouse / total;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_ScreenPMouse;
                  end
            end;
         if HR^^.HRScreenPKey <> 0 then
            begin
               temp := HR^^.HRScreenPKey / total;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_ScreenPKey;
                  end
            end;
         if HR^^.HRScreenPMenu <> 0 then
            begin
               temp := HR^^.HRScreenPMenu / total;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_ScreenPMenu;
                  end
            end;
         if HR^^.HRLinePKey <> 0 then
            begin
               temp := HR^^.HRLinePKey / total;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_LinePKey;
                  end
            end;
         if HR^^.HRLinePMenu <> 0 then
            begin
               temp := HR^^.HRLinePMenu / total;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_LinePMenu;
                  end
            end;

      end;

   if HR^^.HRnumViewsN <> 0 then
      begin
         if HR^^.HRClickDnArrow <> 0 then
            begin
               temp  := HR^^.HRClickDnArrow / total;
               if temp > best then
                  begin
                     best  := temp;
                     bestMethod := H_ClickDnArrow;
                  end
            end;
         if HR^^.HRClickDnArrowRepeat <> 0 then
            begin
               temp := HR^^.HRClickDnArrowRepeat / total;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_ClickDnArrowRepeat;
                  end
            end;
```

```
        if HR^^.HRPressDnArrow <> 0 then
            begin
                temp := HR^^.HRPressDnArrow / total;
                if temp > best then
                    begin
                        best := temp;
                        bestMethod := H_PressDnArrow;
                    end
            end;
        if HR^^.HRScreenNMouse <> 0 then
            begin
                temp := HR^^.HRScreenNMouse / total;
                if temp > best then
                    begin
                        best := temp;
                        bestMethod := H_ScreenNMouse;
                    end
            end;
        if HR^^.HRScreenNKey <> 0 then
            begin
                temp := HR^^.HRScreenNKey / total;
                if temp > best then
                    begin
                        best := temp;
                        bestMethod := H_ScreenNKey;
                    end
            end;
        if HR^^.HRScreenNMenu <> 0 then
            begin
                temp := HR^^.HRScreenNMenu / total;
                if temp > best then
                    begin
                        best := temp;
                        bestMethod := H_ScreenNMenu;
                    end
            end;
        if HR^^.HRLineNKey <> 0 then
            begin
                temp := HR^^.HRLineNKey / total;
                if temp > best then
                    begin
                        best := temp;
                        bestMethod := H_LineNKey;
                    end
            end;
        if HR^^.HRLineNMenu <> 0 then
            begin
                temp := HR^^.HRLineNMenu / total;
                if temp > best then
                    begin
                        best := temp;
                        bestMethod := H_LineNMenu;
                    end
            end;

    end;

    if HR^^.HRDragThumb <> 0 then
        begin
            temp := HR^^.HRDragThumb / total;
            if temp > best then
                bestMethod := H_DragThumb
        end;

    bestView := bestMethod;
end;
                        ^
{-------------------------------------------------------------------------------
bestReplace determines the user's most commonly used procedure for replacing text.
-------------------------------------------------------------------------------}

function bestReplace:       ( ProfileType)
    var
        bestMethod:  ProfileType;
        best, temp: real;
begin
    bestMethod := H_neverTried;
    temp := 0;
    best := 0;

    if HR^^.HRnumReplace <> 0 then
        begin
            if HR^^.HRReplace1 <> 0 then
                begin
                    temp := HR^^.HRReplace1 / HR^^.HRnumReplace;
                    if temp > best then
                        begin
                            best := temp;
                            bestMethod := H_Replace1;
                        end
                end;
            if HR^^.HRReplace2 <> 0 then
                begin
                    temp := HR^^.HRReplace2 / HR^^.HRnumReplace;
                    if temp > best then
                        begin
                            best := temp;
                            bestMethod := H_Replace2
                        end
                end;
            if HR^^.HRReplace3FromMenu <> 0 then
                begin
                    temp := HR^^.HRReplace3FromMenu / HR^^.HRnumReplace;
                    if temp > best then
                        begin
                            best := temp;
                            bestMethod := H_Replace3Menu
                        end
```

```
        end;
      if HR^^.HRReplace3FromPalette <> 0 then
        begin
          temp := HR^^.HRReplace3FromPalette / HR^^.HRnumReplace;
          if temp > best then
            begin
              best := temp;
              bestMethod := H_Replace3Palette
            end
        end;
      if HR^^.HRReplace4FromMenu <> 0 then
        begin
          temp := HR^^.HRReplace4FromMenu / HR^^.HRnumReplace;
          if temp > best then
            begin
              best := temp;
              bestMethod := H_Replace4Menu
            end
        end;
      if HR^^.HRReplace4FromPalette <> 0 then
        begin
          temp := HR^^.HRReplace4FromPalette / HR^^.HRnumReplace;
          if temp > best then
            begin
              best := temp;
              bestMethod := H_Replace4Palette
            end
        end;
      if HR^^.HRReplace5 <> 0 then
        begin
          temp := HR^^.HRReplace5 / HR^^.HRnumReplace;
          if temp > best then
            begin
              best := temp;
              bestMethod := H_Replace5
            end
        end;
      if HR^^.HRReplace6 <> 0 then
        begin
          temp := HR^^.HRReplace6 / HR^^.HRnumReplace;
          if temp > best then
            begin
              best := temp;
              bestMethod := H_Replace6
            end
        end;
      if HR^^.HRReplace7FromMenu <> 0 then
        begin
          temp := HR^^.HRReplace7FromMenu / HR^^.HRnumReplace;
          if temp > best then
            begin
              best := temp;
              bestMethod := H_Replace7Menu
            end
        end;
      if HR^^.HRReplace7FromPalette <> 0 then
        begin
          temp := HR^^.HRReplace7FromPalette / HR^^.HRnumReplace;
          if temp > best then
            begin
              best := temp;
              bestMethod := H_Replace7Palette
            end
        end;
      if HR^^.HRReplace8FromMenu <> 0 then
        begin
          temp := HR^^.HRReplace8FromMenu / HR^^.HRnumReplace;
          if temp > best then
            begin
              best := temp;
              bestMethod := H_Replace8Menu
            end
        end;
      if HR^^.HRReplace8FromPalette <> 0 then
        begin
          temp := HR^^.HRReplace8FromPalette / HR^^.HRnumReplace;
          if temp > best then
            begin
              best := temp;
              bestMethod := H_Replace8Palette
            end
        end;
    end;

  bestReplace := bestMethod;
end;

{-------------------------------------------------------------------------------
bestReplace61 determines the user's most commonly used procedure for replacing text, not
considering replace method 1.
-------------------------------------------------------------------------------}

function bestReplace61:      ProfileType;
  var
    bestMethod: ProfileType;
    best, temp: real;
begin
  bestMethod := H_neverTried;
  temp := 0;
  best := 0;

  if HR^^.HRnumReplace <> 0 then
    begin
      if HR^^.HRReplace2 <> 0 then
        begin
          temp := HR^^.HRReplace2 / HR^^.HRnumReplace;
          if temp > best then
            begin
```

```
                    best := temp;
                    bestMethod := H_Replace2
                  end
              end;
            if HR^^.HRReplace3FromMenu <> 0 then
              begin
                temp := HR^^.HRReplace3FromMenu / HR^^.HRnumReplace;
                if temp > best then
                  begin
                    best := temp;
                    bestMethod := H_Replace3Menu
                  end
              end;
            if HR^^.HRReplace3FromPalette <> 0 then
              begin
                temp := HR^^.HRReplace3FromPalette / HR^^.HRnumReplace;
                if temp > best then
                  begin
                    best := temp;
                    bestMethod := H_Replace3Palette
                  end
              end;
            if HR^^.HRReplace4FromMenu <> 0 then
              begin
                temp := HR^^.HRReplace4FromMenu / HR^^.HRnumReplace;
                if temp > best then
                  begin
                    best := temp;
                    bestMethod := H_Replace4Menu
                  end
              end;
            if HR^^.HRReplace4FromPalette <> 0 then
              begin
                temp := HR^^.HRReplace4FromPalette / HR^^.HRnumReplace;
                if temp > best then
                  begin
                    best := temp;
                    bestMethod := H_Replace4Palette
                  end
              end;
            if HR^^.HRReplace6 <> 0 then
              begin
                temp := HR^^.HRReplace6 / HR^^.HRnumReplace;
                if temp > best then
                  begin
                    best := temp;
                    bestMethod := H_Replace6
                  end
              end;
            if HR^^.HRReplace8 <> 0 then
              begin
                temp := HR^^.HRReplace8 / HR^^.HRnumReplace;
                if temp > best then
                  begin
                    best := temp;
                    bestMethod := H_Replace8
                  end
              end;
            if HR^^.HRReplace7FromMenu <> 0 then
              begin
                temp := HR^^.HRReplace7FromMenu / HR^^.HRnumReplace;
                if temp > best then
                  begin
                    best := temp;
                    bestMethod := H_Replace7Menu
                  end
              end;
            if HR^^.HRReplace7FromPalette <> 0 then
              begin
                temp := HR^^.HRReplace7FromPalette / HR^^.HRnumReplace;
                if temp > best then
                  begin
                    best := temp;
                    bestMethod := H_Replace7Palette
                  end
              end;
            if HR^^.HRReplace8FromMenu <> 0 then
              begin
                temp := HR^^.HRReplace8FromMenu / HR^^.HRnumReplace;
                if temp > best then
                  begin
                    best := temp;
                    bestMethod := H_Replace8Menu
                  end
              end;
            if HR^^.HRReplace8FromPalette <> 0 then
              begin
                temp := HR^^.HRReplace8FromPalette / HR^^.HRnumReplace;
                if temp > best then
                  begin
                    best := temp;
                    bestMethod := H_Replace8Palette
                  end
              end;
          end;

        bestReplace8 := bestMethod;
      end;

{bestDelete determines the user's most commonly used procedure for deleting text.}

      function bestDelete:      ProfileType;
        var
          bestMethod: ProfileType;
          best, temp: real;
      begin
```

```
    bestMethod := H_neverTried;
    temp := 0;
    best := 0;

    if HR^^.HRnumDelete <> 0 then
        begin
            if HR^^.HRDelete1 <> 0 then
                begin
                    temp := HR^^.HRDelete1 / HR^^.HRnumDelete;
                    if temp > best then
                        begin
                            best := temp;
                            bestMethod := H_Delete1;
                        end
                end;
            if HR^^.HRDelete2 <> 0 then
                begin
                    temp := HR^^.HRDelete2 / HR^^.HRnumDelete;
                    if temp > best then
                        begin
                            best := temp;
                            bestMethod := H_Delete2;
                        end
                end;
            if HR^^.HRDelete3FromMenu <> 0 then
                begin
                    temp := HR^^.HRDelete3FromMenu / HR^^.HRnumDelete;
                    if temp > best then
                        begin
                            best := temp;
                            bestMethod := H_Delete3Menu;
                        end
                end;
            if HR^^.HRDelete3FromPalette <> 0 then
                begin
                    temp := HR^^.HRDelete3FromPalette / HR^^.HRnumDelete;
                    if temp > best then
                        begin
                            best := temp;
                            bestMethod := H_Delete3Palette;
                        end
                end;
            if HR^^.HRDelete4FromMenu <> 0 then
                begin
                    temp := HR^^.HRDelete4FromMenu / HR^^.HRnumDelete;
                    if temp > best then
                        begin
                            best := temp;
                            bestMethod := H_Delete4Menu;
                        end
                end;
            if HR^^.HRDelete4FromPalette <> 0 then
                begin
                    temp := HR^^.HRDelete4FromPalette / HR^^.HRnumDelete;
                    if temp > best then
                        begin
                            best := temp;
                            bestMethod := H_Delete4Palette;
                        end
                end;

        end;

    bestDelete := bestMethod;
end;
```

bestDelete34 determines the user's most commonly used procedure for deleting text, considering only delete methods 3 and 4.

```
function bestDelete34:          ProfileType)
    var
        bestMethod: ProfileType;
        best, temp: real;
begin
    bestMethod := H_neverTried;
    temp := 0;
    best := 0;

    if HR^^.HRnumDelete <> 0 then
        begin
            if HR^^.HRDelete3FromMenu <> 0 then
                begin
                    temp := HR^^.HRDelete3FromMenu / HR^^.HRnumDelete;
                    if temp > best then
                        begin
                            best := temp;
                            bestMethod := H_Delete3Menu;
                        end
                end;
            if HR^^.HRDelete3FromPalette <> 0 then
                begin
                    temp := HR^^.HRDelete3FromPalette / HR^^.HRnumDelete;
                    if temp > best then
                        begin
                            best := temp;
                            bestMethod := H_Delete3Palette;
                        end
                end;
            if HR^^.HRDelete4FromMenu <> 0 then
                begin
                    temp := HR^^.HRDelete4FromMenu / HR^^.HRnumDelete;
                    if temp > best then
                        begin
                            best := temp;
                            bestMethod := H_Delete4Menu;
                        end
```

```
            end;
         if HR^^.HRDeleted4FromPalette <> 0 then
            begin
               temp := HR^^.HRDeleted4FromPalette / HR^^.HRnumDelete;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Deleted4Palette;
                  end
            end;

      end;

   bestDelete34 := bestMethod;
end;

{-------------------------------------------------------------------------------
bestMove determines the user's most commonly used procedure for moving text.
-------------------------------------------------------------------------------}

function bestMove:       ProfileType;
   var
      bestMethod: ProfileType;
      best, temp: real;
begin
   bestMethod := H_neverTried;
   temp := 0;
   best := 0;

   if HR^^.HRnumMove <> 0 then
      begin
         if HR^^.HRMove1Menu <> 0 then
            begin
               temp := HR^^.HRMove1Menu / HR^^.HRnumMove;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Move1Menu;
                  end
            end;
         if HR^^.HRMove1Key <> 0 then
            begin
               temp := HR^^.HRMove1Key / HR^^.HRnumMove;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Move1Key;
                  end
            end;
         if HR^^.HRMove1Palette <> 0 then
            begin
               temp := HR^^.HRMove1Palette / HR^^.HRnumMove;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Move1Palette;
                  end
            end;
         if HR^^.HRMove2Menu <> 0 then
            begin
               temp := HR^^.HRMove2Menu / HR^^.HRnumMove;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Move2Menu;
                  end
            end;
         if HR^^.HRMove2Key <> 0 then
            begin
               temp := HR^^.HRMove2Key / HR^^.HRnumMove;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Move2Key;
                  end
            end;
         if HR^^.HRMove2Palette <> 0 then
            begin
               temp := HR^^.HRMove2Palette / HR^^.HRnumMove;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Move2Palette;
                  end
            end;
         if HR^^.HRMove3Menu <> 0 then
            begin
               temp := HR^^.HRMove3Menu / HR^^.HRnumMove;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Move3Menu;
                  end
            end;
         if HR^^.HRMove3Key <> 0 then
            begin
               temp := HR^^.HRMove3Key / HR^^.HRnumMove;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Move3Key;
                  end
            end;
         if HR^^.HRMove3Palette <> 0 then
            begin
               temp := HR^^.HRMove3Palette / HR^^.HRnumMove;
               if temp > best then
                  begin
```

```
            best := temp;
            bestMethod := H_Move3Palette;
          end
    end;
if HR^^.HRMove4Menu <> 0 then
    begin
      temp := HR^^.HRMove4Menu / HR^^.HRnumMove;
      if temp > best then
        begin
          best := temp;
          bestMethod := H_Move4Menu;
        end
    end;
if HR^^.HRMove4Key <> 0 then
    begin
      temp := HR^^.HRMove4Key / HR^^.HRnumMove;
      if temp > best then
        begin
          best := temp;
          bestMethod := H_Move4Key;
        end
    end;
if HR^^.HRMove4Palette <> 0 then
    begin
      temp := HR^^.HRMove4Palette / HR^^.HRnumMove;
      if temp > best then
        begin
          best := temp;
          bestMethod := H_Move4Palette;
        end
    end;
if HR^^.HRMove5Menu <> 0 then
    begin
      temp := HR^^.HRMove5Menu / HR^^.HRnumMove;
      if temp > best then
        begin
          best := temp;
          bestMethod := H_Move5Menu;
        end
    end;
if HR^^.HRMove5Key <> 0 then
    begin
      temp := HR^^.HRMove5Key / HR^^.HRnumMove;
      if temp > best then
        begin
          best := temp;
          bestMethod := H_Move5Key;
        end
    end;
if HR^^.HRMove5Palette <> 0 then
    begin
      temp := HR^^.HRMove5Palette / HR^^.HRnumMove;
      if temp > best then
        begin
          best := temp;
          bestMethod := H_Move5Palette;
        end
    end;
if HR^^.HRMove6Menu <> 0 then
    begin
      temp := HR^^.HRMove6Menu / HR^^.HRnumMove
      if temp > best then
        begin
          best := temp;
          bestMethod := H_Move6Menu;
        end
    end;
if HR^^.HRMove6Key <> 0 then
    begin
      temp := HR^^.HRMove6Key / HR^^.HRnumMove;
      if temp > best then
        begin
          best := temp;
          bestMethod := H_Move6Key;
        end
    end;
if HR^^.HRMove6Palette <> 0 then
    begin
      temp := HR^^.HRMove6Palette / HR^^.HRnumMove;
      if temp > best then
        begin
          best := temp;
          bestMethod := H_Move6Palette;
        end
    end;
if HR^^.HRMove7Menu <> 0 then
    begin
      temp := HR^^.HRMove7Menu / HR^^.HRnumMove;
      if temp > best then
        begin
          best := temp;
          bestMethod := H_Move7Menu;
        end
    end;
if HR^^.HRMove7Palette <> 0 then
    begin
      temp := HR^^.HRMove7Palette / HR^^.HRnumMove;
      if temp > best then
        begin
          best := temp;
          bestMethod := H_Move7Palette;
        end
    end;
if HR^^.HRMove8Menu <> 0 then
    begin
      temp := HR^^.HRMove8Menu / HR^^.HRnumMove;
      if temp > best then
        begin
```

```
                    best := temp;
                    bestMethod := H_Move5Menu;
                  end
              end;
            if HR^^ HRMove6Palette <> 0 then
              begin
                temp := HR^^ HRMove6Palette / HR^^ HRnumMove;
                if temp > best then
                  begin
                    best := temp;
                    bestMethod := H_Move6Palette;
                  end
              end;

        end;
      bestMove := bestMethod;
    end;

{ ..... }
{bestMove78 determines the user's most commonly used procedure for moving text, considering }
{only move methods 7 and 8.                                                                 }
{ ..... }

  function bestMove78:    ( ProfileType)
    var
      bestMethod: ProfileType;
      best, temp: real;
  begin
    bestMethod := H_neverTried;
    temp := 0;
    best := 0;

    if HR^^ HRnumMove <> 0 then
      begin
        if HR^^ HRMove7Menu <> 0 then
          begin
            temp := HR^^ HRMove7Menu / HR^^ HRnumMove;
            if temp > best then
              begin
                best := temp;
                bestMethod := H_Move7Menu;
              end
          end;
        if HR^^ HRMove7Palette <> 0 then
          begin
            temp := HR^^ HRMove7Palette / HR^^ HRnumMove
            if temp > best then
              begin
                best := temp;
                bestMethod := H_Move7Palette
              end
          end;
        if HR^^ HRMove8Menu <> 0 then
          begin
            temp := HR^^ HRMove8Menu / HR^^ HRnumMove;
            if temp > best then
              begin
                best := temp;
                bestMethod := H_Move8Menu;
              end
          end;
        if HR^^ HRMove8Palette <> 0 then
          begin
            temp := HR^^ HRMove8Palette / HR^^ HRnumMove
            if temp > best then
              begin
                best := temp;
                bestMethod := H_Move8Palette;
              end
          end;
      end;
    bestMove78 := bestMethod;
  end;

{ ..... }
{bestDuplicate determines the user's most commonly used procedure for duplicating text.  }
{ ..... }

  function bestDuplicate:          ProfileType)
    var
      bestMethod: ProfileType;
      best, temp: real;
  begin
    bestMethod := H_neverTried;
    temp := 0;
    best := 0;

    if HR^^ HRnumDuplicate <> 0 then
      begin
        if HR^^ HRDuplicate1 <> 0 then
          begin
            temp := HR^^ HRDuplicate1 / HR^^ HRnumDuplicate;
            if temp > best then
              begin
                best := temp;
                bestMethod := H_Duplicate1;
              end
          end;
        if HR^^ HRDuplicate2 <> 0 then
          begin
            temp := HR^^ HRDuplicate2 / HR^^ HRnumDuplicate;
            if temp > best then
              begin
                best := temp;
                bestMethod := H_Duplicate2;
              end
          end;
        if HR^^ HRDuplicate3Menu <> 0 then
```

```
            begin
               temp := HR** HRDuplicate3Menu / HR** HRnumDuplicate;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Duplicate3Menu;
                  end
            end;
         if HR** HRDuplicate3Key <> 0 then
            begin
               temp := HR** HRDuplicate3Key / HR** HRnumDuplicate;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Duplicate3Key;
                  end
            end;
         if HR** HRDuplicate3Palette <> 0 then
            begin
               temp := HR** HRDuplicate3Palette / HR** HRnumDuplicate;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Duplicate3Palette;
                  end
            end;
         if HR** HRDuplicate4Menu <> 0 then
            begin
               temp := HR** HRDuplicate4Menu / HR** HRnumDuplicate;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Duplicate4Menu;
                  end
            end;
         if HR** HRDuplicate4Key <> 0 then
            begin
               temp := HR** HRDuplicate4Key / HR** HRnumDuplicate;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Duplicate4Key
                  end
            end
         if HR** HRDuplicate4Palette <> 0 then
            begin
               temp := HR** HRDuplicate4Palette / HR** HRnumDuplicate;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Duplicate4Palette
                  end
            end;

      end;
   bestDuplicate := bestMethod;
end;

{.....}
{bestDuplicate12 determines the user's most commonly used procedure for duplicating text  }
{considering only duplicate methods 1 and 2                                               }
{.....}

function bestDuplicate12:    ProfileType;
   var
      bestMethod: ProfileType;
      best, temp: real;
begin
   bestMethod := H_neverTried;
   temp := 0;
   best := 0;

   if HR** HRnumDuplicate <> 0 then
      begin
         if HR** HRDuplicate1 <> 0 then
            begin
               temp := HR** HRDuplicate1 / HR** HRnumDuplicate;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Duplicate1
                  end
            end;
         if HR** HRDuplicate2 <> 0 then
            begin
               temp := HR** HRDuplicate2 / HR** HRnumDuplicate;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Duplicate2;
                  end
            end;

      end;
   bestDuplicate12 := bestMethod;
end;

{.....}
{bestCut determines the user's most commonly used procedure for cutting text.  }
{.....}

function bestCut:       ProfileType;
   var
      bestMethod: ProfileType;
      best, temp: real;
begin
   bestMethod := H_neverTried;
   temp := 0;
```

```
best := 0;

if HR^^.HRnumCut <> 0 then
   begin
      if HR^^.HRCut1Menu <> 0 then
         begin
            temp := HR^^.HRCut1Menu / HR^^.HRnumCut;
            if temp > best then
               begin
                  best := temp;
                  bestMethod := H_Cut1Menu;
               end
         end;
      if HR^^.HRCut1Key <> 0 then
         begin
            temp := HR^^.HRCut1Key / HR^^.HRnumCut;
            if temp > best then
               begin
                  best := temp;
                  bestMethod := H_Cut1Key;
               end
         end;
      if HR^^.HRCut1Palette <> 0 then
         begin
            temp := HR^^.HRCut1Palette / HR^^.HRnumCut;
            if temp > best then
               begin
                  best := temp;
                  bestMethod := H_Cut1Palette;
               end
         end;
      if HR^^.HRCut2Menu <> 0 then
         begin
            temp := HR^^.HRCut2Menu / HR^^.HRnumCut;
            if temp > best then
               begin
                  best := temp;
                  bestMethod := H_Cut2Menu;
               end
         end;
      if HR^^.HRCut2Key <> 0 then
         begin
            temp := HR^^.HRCut2Key / HR^^.HRnumCut;
            if temp > best then
               begin
                  best := temp;
                  bestMethod := H_Cut2Key;
               end
         end;
      if HR^^.HRCut2Palette <> 0 then
         begin
            temp := HR^^.HRCut2Palette / HR^^.HRnumCut;
            if temp > best then
               begin
                  best := temp;
                  bestMethod := H_Cut2Palette;
               end
         end;

      end;
   bestCut := bestMethod;
end

{.....}
{bestCopy determines the user's most commonly used procedure for copying text.}
{.....}

function bestCopy: ProfileType;
   var
      bestMethod: ProfileType;
      best, temp: real;
begin
   bestMethod := H_neverTried;
   temp := 0;
   best := 0;

   if HR^^.HRnumCopy <> 0 then
      begin
         if HR^^.HRCopy1Menu <> 0 then
            begin
               temp := HR^^.HRCopy1Menu / HR^^.HRnumCopy;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Copy1Menu;
                  end
            end;
         if HR^^.HRCopy1Key <> 0 then
            begin
               temp := HR^^.HRCopy1Key / HR^^.HRnumCopy;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Copy1Key;
                  end
            end;
         if HR^^.HRCopy1Palette <> 0 then
            begin
               temp := HR^^.HRCopy1Palette / HR^^.HRnumCopy;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Copy1Palette;
                  end
            end;
         if HR^^.HRCopy2Menu <> 0 then
            begin
               temp := HR^^.HRCopy2Menu / HR^^.HRnumCopy;
```

```
            if temp > best then
               begin
                  best := temp;
                  bestMethod := H_Copy2Menu;
               end
         end;
      if HR^^ HRCopy2Key <> 0 then
         begin
            temp := HR^^ HRCopy2Key / HR^^ HRnumCopy;
            if temp > best then
               begin
                  best := temp;
                  bestMethod := H_Copy2Key;
               end
         end;
      if HR^^ HRCopy2Palette <> 0 then
         begin
            temp := HR^^ HRCopy2Palette / HR^^ HRnumCopy;
            if temp > best then
               begin
                  best := temp;
                  bestMethod := H_Copy2Palette;
               end
         end;

   end;
  bestCopy := bestMethod;
end;

{ ..... }
{ bestPaste determines the user's most commonly used procedure for pasting text }
{ ..... }

function bestPaste:      ProfileType)
   var
      bestMethod: ProfileType;
      best, temp: real;
begin
   bestMethod := H_neverTried;
   temp := 0;
   best := 0;

   if HR^^ HRnumPaste <> 0 then
      begin
         if HR^^ HRPaste1Menu <> 0 then
            begin
               temp := HR^^ HRPaste1Menu / HR^^ HRnumPaste
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Paste1Menu;
                  end
            end;
         if HR^^ HRPaste1Key <> 0 then
            begin
               temp := HR^^ HRPaste1Key / HR^^ HRnumPaste
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Paste1Key
                  end
            end;
         if HR^^ HRPaste1Palette <> 0 then
            begin
               temp := HR^^ HRPaste1Palette / HR^^ HRnumPaste
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Paste1Palette;
                  end
            end;
         if HR^^ HRPaste2Menu <> 0 then
            begin
               temp := HR^^ HRPaste2Menu / HR^^ HRnumPaste;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Paste2Menu;
                  end
            end;
         if HR^^ HRPaste2Key <> 0 then
            begin
               temp := HR^^ HRPaste2Key / HR^^ HRnumPaste;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Paste2Key;
                  end
            end;
         if HR^^ HRPaste2Palette <> 0 then
            begin
               temp := HR^^ HRPaste2Palette / HR^^ HRnumPaste;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Paste2Palette;
                  end
            end;

      end;
   bestPaste := bestMethod;
end;

{ ..... }
{ bestSize determines the user's most commonly used procedure for changing text size. }
{ ..... }

function bestSize:      ProfileType)
```

```
var
   bestMethod: ProfileType;
   best, temp: real;
begin
   bestMethod := H_neverTried;
   temp := 0;
   best := 0;

   if HR^^ HRnumSize <> 0 then
      begin
         if HR^^ HRTest1Menu <> 0 then
            begin
               temp := HR^^ HRTest1Menu / HR^^ HRnumSize;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Test1Menu;
                  end
            end;
         if HR^^ HRTest1Palette <> 0 then
            begin
               temp := HR^^ HRTest1Palette / HR^^ HRnumSize;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Test1Palette;
                  end
            end;
         if HR^^ HRTest4Menu <> 0 then
            begin
               temp := HR^^ HRTest4Menu / HR^^ HRnumSize;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Test4Menu;
                  end
            end;
         if HR^^ HRTest4Palette <> 0 then
            begin
               temp := HR^^ HRTest4Palette / HR^^ HRnumSize;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Test4Palette;
                  end
            end;
         if HR^^ HRTest7Menu <> 0 then
            begin
               temp := HR^^ HRTest7Menu / HR^^ HRnumSize;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Test7Menu;
                  end
            end;
         if HR^^ HRTest7Palette <> 0 then
            begin
               temp := HR^^ HRTest7Palette / HR^^ HRnumSize;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Test7Palette;
                  end
            end;

      end;
   bestSize := bestMethod;
end;

{ ---------------------------------------------------------------------------- }
{ bestFont determines the user's most commonly used procedure for changing text font. }
{ ---------------------------------------------------------------------------- }

function bestFont:      ProfileType;
   var
      bestMethod: ProfileType;
      best, temp: real;
begin
   bestMethod := H_neverTried;
   temp := 0;
   best := 0;

   if HR^^ HRnumFont <> 0 then
      begin
         if HR^^ HRTest2Menu <> 0 then
            begin
               temp := HR^^ HRTest2Menu / HR^^ HRnumFont;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Test2Menu;
                  end
            end;
         if HR^^ HRTest2Palette <> 0 then
            begin
               temp := HR^^ HRTest2Palette / HR^^ HRnumFont;
               if temp > best then
                  begin
                     best := temp;
                     bestMethod := H_Test2Palette;
                  end
            end;
         if HR^^ HRTest5Menu <> 0 then
            begin
               temp := HR^^ HRTest5Menu / HR^^ HRnumFont;
               if temp > best then
                  begin
                     best := temp;
```

```
                    bestMethod := H_TextMMenu;
                  end;
              end;
            if HR^^.HRTextBPalette <> 0 then
              begin
                temp := HR^^.HRTextBPalette / HR^^.HRnumFont;
                if temp > best then
                  begin
                    best := temp;
                    bestMethod := H_TextBPalette;
                  end
              end;
            if HR^^.HRTextMMenu <> 0 then
              begin
                temp := HR^^.HRTextMMenu / HR^^.HRnumFont;
                if temp > best then
                  begin
                    best := temp;
                    bestMethod := H_TextMMenu;
                  end
              end;
            if HR^^.HRTextBPalette <> 0 then
              begin
                temp := HR^^.HRTextBPalette / HR^^.HRnumFont;
                if temp > best then
                  begin
                    best := temp;
                    bestMethod := H_TextBPalette;
                  end
              end;

          end;
        bestFont := bestMethod;
      end;

{testStyle determines the user's most commonly used procedure for changing text style}

    function bestStyle:          ProfileType;
      var
        bestMethod: ProfileType;
        best, temp: real;
    begin
      bestMethod := H_neverTried;
      temp := 0;
      best := 0;

      if HR^^.HRnumStyle <> 0 then
        begin
          if HR^^.HRTextQMenu <> 0 then
            begin
              temp := HR^^.HRTextQMenu / HR^^.HRnumStyle;
              if temp > best then
                begin
                  best := temp;
                  bestMethod := H_TextQMenu;
                end
            end;
          if HR^^.HRTextQPalette <> 0 then
            begin
              temp := HR^^.HRTextQPalette / HR^^.HRnumStyle
              if temp > best then
                begin
                  best := temp;
                  bestMethod := H_TextQPalette;
                end
            end;
          if HR^^.HRTextMMenu <> 0 then
            begin
              temp := HR^^.HRTextMMenu / HR^^.HRnumStyle;
              if temp > best then
                begin
                  best := temp;
                  bestMethod := H_TextMMenu;
                end
            end;
          if HR^^.HRTextBPalette <> 0 then
            begin
              temp := HR^^.HRTextBPalette / HR^^.HRnumStyle;
              if temp > best then
                begin
                  best := temp;
                  bestMethod := H_TextBPalette;
                end
            end;
          if HR^^.HRTextMMenu <> 0 then
            begin
              temp := HR^^.HRTextMMenu / HR^^.HRnumStyle;
              if temp > best then
                begin
                  best := temp;
                  bestMethod := H_TextMMenu;
                end
            end;
          if HR^^.HRTextBPalette <> 0 then
            begin
              temp := HR^^.HRTextBPalette / HR^^.HRnumStyle;
              if temp > best then
                begin
                  best := temp;
                  bestMethod := H_TextBPalette;
                end
            end;
        end;
      bestStyle := bestMethod;
    end;
end.
```

```
Margaret Stone
SoM  Project, Brown University

This unit contains the code to determine from the user help profile the user's most often-used methods
of performing a given task.

unit helpBasis2;

interface Section

interface

  uses
    EditorGlobals, EditorHelper;

  function bestReplace78: ProfileType;

implementation Section

implementation

bestReplace78 determines the user's most commonly used procedure for replacing text, considering
only replace methods 7 and 8

  function bestReplace78:              ProfileType)
    var
      bestMethod: ProfileType;
      best, temp: real;
  begin
    bestMethod  = H_NeverTried;
    temp  = 0;
    best  = 0;

    if HR^.HRNumReplace <> 0 then
      begin
        if HR^.HRReplace7FromMenu <> 0 then
          begin
            temp := HR^.HRReplace7FromMenu / HR^.HRNumReplace
            if temp > best then
              begin
                best  = temp;
                bestMethod = H_Replace7Menu
              end
          end;
        if HR^.HRReplace7FromPalette <> 0 then
          begin
            temp  = HR^.HRReplace7FromPalette / HR^.HRNumReplace;
            if temp > best then
              begin
                best  = temp;
                bestMethod  = H_Replace7Palette
              end
          end;
        if HR^.HRReplace8FromMenu <> 0 then
          begin
            temp  = HR^.HRReplace8FromMenu / HR^.HRNumReplace
            if temp > best then
              begin
                best  = temp;
                bestMethod  = H_Replace8Menu
              end
          end;
        if HR^.HRReplace8FromPalette <> 0 then
          begin
            temp  = HR^.HRReplace8FromPalette / HR^.HRNumReplace;
            if temp > best then
              begin
                best  = temp;
                bestMethod  = H_Replace8Palette
              end
          end;
      end;

    bestReplace78  = bestMethod;
  end;
end.
```

```
{
  Margaret Stone
  Sc.M. Project, Brown University

  This unit contains the code to provide help to the user when the user asks for help deleting text.
}

unit EditHelp;

{
  Interface Section
}

interface

  uses
    EditorHelpIntf, EditorUtilities, HelpBasis;

  procedure helpDeleting (fromMenu: boolean);

{
  Implementation Section
}

implementation

  const
{string indices in string list HelpFormatSTR:}
    select1 = 4;
    select2 = 5;
    select3 = 6;
    select4 = 7;
    select5 = 8;
    result3 = 13;

{string indices in string list HelpCCSTR:}
    reminder1 = 1;
    reminder2 = 2;
    first1 = 3;
    firstnext = 4;
    result1 = 5;
    result2 = 6;

{string indices in string list HelpReplaceSTR:}
    first1replace = 1;
    next1replace = 2;
    result1replace = 3;
    first2replace = 4;
    next2replace = 5;
    next3replace = 6;
    first3replace = 7;
    first4replace = 8;
    result2replace = 9;
    deleteResult = 10;

{item numbers in dialog box:}
    reminderItem = 3;
    firstItem = 4;
    nextItem = 5;
    resultItem = 6;

{
  helpDeleting gives the user help when they've chosen Help->Editing->Deleting text.
}

  procedure helpDeleting; <-- (fromMenu:boolean);
    const
      DoneBusan = 1;
    var
      dPtr: DialogPtr;
      hit, itemType: integer;
      DLOGDone: boolean;
      beeMethod: probeType;
      textHandle: Handle;
      box: Rect;
      reminderStr, firstStr, nextStr, resultStr, caret0, caret1, caret2, caret3, selectStr: Str255;

    procedure DeleteMess1;
    begin
      caret2 := 'the passage you want to delete';
      GetIndString(reminderStr, HelpCCSTR, reminder1);
      GetIndString(firstStr, HelpReplaceSTR, first1replace);
      GetIndString(nextStr, HelpReplaceSTR, next1replace);
      GetIndString(resultStr, HelpReplaceSTR, deleteResult);
      caret3 := '';
    end;

    procedure DeleteMess2;
    begin
      caret2 := 'the passage you want to delete';
      GetIndString(reminderStr, HelpCCSTR, reminder1);
      GetIndString(firstStr, HelpReplaceSTR, first2replace);
      firstStr := concat(firstStr, selectStr);
      GetIndString(nextStr, HelpReplaceSTR, next2replace);
      GetIndString(resultStr, HelpReplaceSTR, deleteResult);
      caret3 := '';
    end;

    procedure DeleteMess3 (fromMenu: boolean);
    begin
      caret2 := 'the passage you want to delete';
      GetIndString(reminderStr, HelpCCSTR, reminder1);
      GetIndString(firstStr, HelpReplaceSTR, first2replace);
      firstStr := concat(firstStr, selectStr);
      GetIndString(nextStr, HelpReplaceSTR, next3replace);
      if fromMenu then
        caret3 := 'choose Delete from the Edit menu'
      else
```

```
            caret3 := 'click in the delete rectangle in the palette (the delete rectangle consists of text that has been struck-through)';
          GetIndString(resultStr, HelpReplaceSTR, deleteResult);
        end;

        procedure DeleteMess4 (fromMenu: boolean);
        begin
          caret2 := '';
          if fromMenu then
            caret3 := 'choose Delete from the Edit menu'
          else
            caret3 := 'click in the delete rectangle in the palette (the delete rectangle consists of text that has been struck-through)';
          GetIndString(nextStr, HelpReplaceSTR, firstreplace);
          GetIndString(resultStr, HelpFormatSTR, result3);
          GetIndString(firstStr, HelpCCSTR, remainder1);
          remainderStr := '';
        end;

      begin
    create help deleting message: }
        bestMethod := bestSelect;
        case bestMethod of
          H_SelectMenu:
            GetIndString(selectStr, HelpFormatSTR, select3);
          H_SelectMouse:
            GetIndString(selectStr, HelpFormatSTR, select4);
          H_SelectKey:
            GetIndString(selectStr, HelpFormatSTR, select5);
          otherwise
            if fromMenu then
              GetIndString(selectStr, HelpFormatSTR, select1)
            else
              GetIndString(selectStr, HelpFormatSTR, select2)
        end;
        bestMethod := H_neverTried;

    See if they've deleted previously: }
        if HR^^ HRnumDelete <> 0 then
          begin
            bestMethod := bestDelete;
            caret1 := 'delete text previously';
            case bestMethod of
              H_delete1:
                deleteMess1;
              H_delete2:
                deleteMess2;
              H_delete3Menu:
                deleteMess3(TRUE);
              H_delete3Palette:
                deleteMess3(FALSE);
              H_delete4Menu:
                deleteMess4(TRUE);
              H_delete4palette:
                deleteMess4(FALSE);
              otherwise

            end;
          end

    If not, see if they've cut: }
        else if HR^^ HRnumCut <> 0 then
          begin
            bestMethod := bestCut;
            caret1 := 'cut text, except the text will not be placed on the Clipboard, and ';
            case bestMethod of
              H_Cut1Menu, H_Cut1Key:
                begin
                  deleteMess3(TRUE);
                  caret1 := concat(caret1, 'you will choose Delete from the Edit menu rather than Cut');
                end;
              H_Cut1Palette:
                begin
                  deleteMess3(FALSE);
                  caret1 := concat(caret1, 'you will click in the delete rectangle of the palette rather than the scissors');
                end;
              H_Cut2Menu, H_Cut2Key:
                begin
                  deleteMess4(TRUE);
                  caret1 := concat(caret1, 'you will choose Delete from the Edit menu rather than Cut');
                end;
              H_Cut2Palette:
                begin
                  deleteMess4(FALSE);
                  caret1 := concat(caret1, 'you will click in the delete rectangle of the palette rather than the scissors');
                end;
              otherwise

            end
          end

    If not, see if they've copied: }
        else if HR^^ HRnumCopy <> 0 then
          begin
            bestMethod := bestCopy;
            caret1 := 'copy text, except the text will not be placed on the Clipboard, and it will be removed from your document';
            case bestMethod of
              H_Copy1Menu, H_Copy1Key:
                begin
                  deleteMess3(TRUE);
                  caret1 := concat(caret1, 'you will choose Delete from the Edit menu rather than Copy');
                end;
              H_Copy1Palette:
                begin
                  deleteMess3(FALSE);
                  caret1 := concat(caret1, 'you will click in the delete rectangle of the palette rather than Text->Clip');
                end;
              H_Copy2Menu, H_Copy2Key:
                begin
                  deleteMess4(TRUE);
```

```
                caret1 := concat(caret1, 'you will choose Delete from the Edit menu rather than Copy');
              end;
            H_Copy2Palette:
              begin
                deleteMess4(FALSE);
                caret1 := concat(caret1, 'you will click in the delete rectangle of the palette rather than Text->Clip');
              end;
            otherwise

          end
        end

{if not, see if they've duplicated: }
      else if HR^.HRnumDuplicate <> 0 then
        begin
          bestMethod := bestDuplicate;
          caret1 := 'duplicate, except ';
          case bestMethod of
            H_duplicate1:
              begin
                caret1 := concat(caret1, 'you will choose Delete from the Edit menu rather than Duplicate');
                deleteMess3(TRUE);
              end;
            H_duplicate2:
              begin
                caret1 := concat(caret1, 'you will choose Delete from the Edit menu rather than Duplicate');
                deleteMess4(TRUE);
              end;
            H_duplicate3menu, H_duplicate3Key:
              begin
                caret1 := concat(caret1, 'you will choose Delete from the Edit menu');
                deleteMess3(TRUE);
              end;
            H_duplicate3Palette:
              begin
                caret1 := concat(caret1, 'you will click in the delete rectangle in the palette');
                deleteMess3(FALSE);
              end;
            H_duplicate4menu, H_duplicate4key:
              begin
                caret1 := concat(caret1, 'you will choose Delete from the Edit menu');
                deleteMess4(TRUE);
              end;
            H_duplicate4palette:
              begin
                caret1 := concat(caret1, 'you will click in the delete rectangle in the palette');
                deleteMess4(FALSE);
              end;
            otherwise

          end
        end

{if not, see if they've moved: }
      else if HR^.HRnumMove <> 0 then
        begin
          bestMethod := bestMove;
          caret1 := 'move text, except ';
          case bestMethod of
            H_Move7Menu:
              begin
                caret1 := concat(caret1, 'you will choose Delete from the Edit menu rather than Move');
                deleteMess3(TRUE);
              end;
            H_Move7Palette:
              begin
                caret1 := concat(caret1, 'you will click in the delete rectangle in the palette rather than the move rectangle');
                deleteMess3(FALSE);
              end;
            H_Move8Menu:
              begin
                caret1 := concat(caret1, 'you will choose Delete from the Edit menu rather than Move');
                deleteMess4(TRUE);
              end;
            H_Move8Palette:
              begin
                caret1 := concat(caret1, 'you will click in the delete rectangle in the palette rather than the move rectangle');
                deleteMess4(FALSE);
              end;
            otherwise

          end;
        end;

{otherwise, if they haven't done any of the above: }
      if bestMethod = H_neverTried then
        begin
          DeleteMess4(FALSE);
          caret1 := '';
          if not FromMenu then
            caret2 := 'click on the delete rectangle in the palette (the delete rectangle consists of letters that have been struck-through)'
          else
            caret2 := 'choose Delete from the Edit menu';
          GetIndString(BriefStr, HelpCCSTR, reminder2);
        end;

{Get the dialog box: }
      DLOGDone := FALSE;
      dPtr := GetNewDialog(DeletingDLOG, nil, pointer(-1));
      SetPort(dPtr);
      TextFont(geneve);
      TextSize(10);
      caret0 := 'delete';
      ParamText(caret0, caret1, caret2, caret3);

{Do the dialog box: }
      GetDItem(dPtr, reminderItem, itemType, textHandle, box);
      SetIText(textHandle, reminderStr);
```

```
    GetDItem(dPtr, firstItem, itemType, textHandle, box);
    SetIText(textHandle, firstStr);

    GetDItem(dPtr, nextItem, itemType, textHandle, box);
    SetIText(textHandle, nextStr);

    GetDItem(dPtr, resultItem, itemType, textHandle, box);
    SetIText(textHandle, resultStr);

    ShowWindow(dPtr);
    PrimeDItem(dPtr, DoneButton);

wait until Done is pressed: }
    while not DLOGDone do
      begin
        ModalDialog(nil, hit);
        if (hit = DoneButton) then
          DLOGDone := TRUE;
      end;
    DisposDialog(dPtr);
  end;
end
```

```
{
  Margaret Stone
  Sc.M. Project, Brown University

  This unit contains the code to provide help to the user when the user asks for help moving text,
  or replacing text.
}

unit EditHelp2;

{
  Interface Section
}

interface

  uses
    EditorHelpInt, EditorUtilities, HelpSets, HelpSets2;

  procedure helpMoving (fromMenu: boolean; isMoving: boolean);
  procedure helpReplacing (fromMenu: boolean; isCorrecting: boolean; isRewording: boolean);

{
  Implementation Section
}

implementation

  const
string indices in string list HelpFormatSTR:}
      select1 = 4;
      select2 = 5;
      select3 = 6;
      select4 = 7;
      select5 = 8;
      result3 = 13;

string indices in string list HelpCCSTR:}
      reminder1 = 1;
      reminder2 = 2;
      first1 = 3;
      firstnext = 4;
      result1 = 5;
      result2 = 6;

string indices in string list HelpPasteSTR:}
      first = 1;
      result = 2;
      dupResult1 = 3;
      dupResult2 = 4;
      dupResult3 = 5;
      dupResult4 = 6;
      dupResult5 = 7;

string indices in string list HelpMoveSTR:}
      next1 = 1;
      next2 = 2;
      firstMove1 = 3;
      firstMove2 = 4;
      next3 = 5;
      next4 = 6;
      firstnextMove = 7

string indices in string list HelpReplaceSTR:}
      first1replace = 1;
      next1replace = 2;
      result1replace = 3;
      first2replace = 4
      next2replace = 5;
      next3replace = 6;
      first3replace = 7
      first4replace = 8;
      result2replace = 9;

item numbers in dialog box:}
      reminderItem = 3;
      firstItem = 4;
      nextItem = 5;
      resultItem = 6;

{
  helpMoving gives the user help when they've chosen Help->Editing->moving or switching text.
}

  procedure helpMoving;          {fromMenu:boolean; isMoving:boolean}
                                 {isMoving=TRUE means moving, otherwise, switching}
    const
      DoneButton = 1;
    var
      dPtr: DialogPtr;
      ht, itemType: integer;
      DLOGDone: boolean;
      oldMethod: ProfileType;
      textHandle: Handle;
      box: Rect;
      reminderStr, firstStr, nextStr, resultStr, caret0, caret1, caret2, caret3, selectStr: Str255;

    procedure MoveMess1Menu;
    begin
      GetIndString(resultStr, HelpFormatSTR, result3);
      caret2 := 'First';
      if isMoving then
        caret3 := 'select the passage you want to move '
      else
        caret3 := 'select one of the passages that you want to switch ';
      GetIndString(firstStr, HelpMoveSTR, firstNextMove);
      GetIndString(nextStr, HelpMoveSTR, next2);
```

```
    firstStr := concat(firstStr, selectStr);
    GetIndString(reminderStr, HelpCCSTR, reminder1);
  end;

  procedure MoveMess1Palette;
  begin
    GetIndString(resultStr, HelpFormatSTR, result3);
    caret2 := 'First';
    if isMoving then
      caret3 := 'select the passage you want to move '
    else
      caret3 := 'select one of the passages that you want to switch '
    GetIndString(firstStr, HelpMoveSTR, firstNextMove);
    GetIndString(nextStr, HelpMoveSTR, next1);
    firstStr := concat(firstStr, selectStr);
    GetIndString(reminderStr, HelpCCSTR, reminder1);
  end;

  procedure MoveMess2Menu;
  begin
    reminderStr := '';
    GetIndString(firstStr, HelpCCSTR, reminder1);
    GetIndString(resultStr, HelpFormatSTR, result3);
    GetIndString(nextStr, HelpMoveSTR, firstMove2);
  end;

  procedure MoveMess2Palette;
  begin
    reminderStr := '';
    GetIndString(firstStr, HelpCCSTR, reminder1);
    GetIndString(resultStr, HelpFormatSTR, result3);
    GetIndString(nextStr, HelpMoveSTR, firstMove1);
  end;

  procedure MoveMess3;
  begin
    GetIndString(reminderStr, HelpCCSTR, reminder1);
    GetIndString(firstStr, HelpCCSTR, firstItem);
    caret2 := 'First';
    if isMoving then
      begin
        caret3 := 'cut the passage you want to move'
        GetIndString(nextStr, HelpMoveSTR, next3)
      end
    else
      begin
        caret3 := 'cut one of the passages you want to switch';
        GetIndString(nextStr, HelpMoveSTR, next4)
      end
  end;

  procedure MoveMess4;
  begin
    GetIndString(reminderStr, HelpCCSTR, reminder1);
    GetIndString(firstStr, HelpCCSTR, firstItem);
    caret2 := 'First';
    if isMoving then
      caret3 := 'cut the passage you want to move'
    else
      caret3 := 'cut one of the passages you want to switch'
    GetIndString(resultStr, HelpFormatSTR, result3);
  end;

  procedure MoveMess5;
  begin
    GetIndString(reminderStr, HelpCCSTR, reminder1);
    GetIndString(firstStr, HelpCCSTR, firstItem);
    caret2 := 'First';
    if isMoving then
      begin
        caret3 := 'copy the passage you want to move, '
        GetIndString(nextStr, HelpMoveSTR, next3);
      end
    else
      begin
        caret3 := 'copy one of the passages you want to switch, ';
        GetIndString(nextStr, HelpMoveSTR, next4);
      end
  end;

  procedure MoveMess6;
  begin
    reminderSTR := ''
    GetIndString(firstStr, HelpCCSTR, reminder1)
    GetIndString(nextStr, HelpCCSTR, firstItem);
    caret2 := 'First';
    if isMoving then
      caret3 := 'copy the passage you want to move, '
    else
      caret3 := 'copy one of the passages you want to switch, ';
    GetIndString(resultStr, HelpFormatSTR, result3);
  end;


begin
create help moving message( )
    bestMethod := bestSelect;
    case bestMethod of
      H_SelectMenu:
        GetIndString(selectStr, HelpFormatSTR, select3);
      H_SelectMouse:
        GetIndString(selectStr, HelpFormatSTR, select4);
      H_SelectKey:
        GetIndString(selectStr, HelpFormatSTR, select5);
      otherwise
        if itemMenu then
          GetIndString(selectStr, HelpFormatSTR, select1)
```

```
        else
          GetIndString(selectStr, HelpFormatSTR, select2)
    end;
    bestMethod := H_neverTried;

{Check to see if they've moved previously: }
    if HR^^.HPrevMove <> 0 then
      begin
        bestMethod := bestMove;
        careIt := 'move text previously';
        case bestMethod of
          H_Move1Menu, H_Move1Key, H_Move1Palette:
            begin
              MoveMess3;
              if bestMethod = H_Move1Menu then
                resultSTR := 'Choose Paste from the Edit menu to paste the cut passage at the insertion point (where you clicked).'
              else if bestMethod = H_Move1Key then
                resultSTR := 'Simultaneously press the command (clover) and v keys to paste the cut passage at the insertion point (where you
clicked).'
              else
                resultSTR := 'Click in the Clip->Text rectangle in the palette to paste the cut passage at the insertion point (where you clicked).'
            end;
          H_Move2Menu, H_Move2Key, H_Move2Palette:
            begin
              MoveMess4;
              if bestMethod = H_Move1Menu then
                if isMoving then
                  nextSTR := 'Next, choose Paste from the Edit menu to paste the cut passage where you want.'
                else
                  nextSTR := 'Next, choose Paste from the Edit menu to paste the cut passage where you want in relation to the other passage.'
              else if bestMethod = H_Move1Key then
                if isMoving then
                  nextSTR := 'Next, simultaneously press the command (clover) and v keys to paste the cut passage where you want.'
                else
                  nextSTR := 'Next, simultaneously press the command (clover) and v keys to paste the cut passage where you want in relation to
the other passage.'
              else if isMoving then
                nextSTR := 'Next, click in the Clip->Text rectangle in the palette to paste the cut passage where you want.'
              else
                nextSTR := 'Next, click in the Clip->Text rectangle in the palette to paste the cut passage where you want in relation to the other
passage.'
            end;
          H_Move3Menu, H_Move3Key, H_Move3Palette:
            begin
              MoveMess5;
              careIt := concat(careIt, 'then press the delete key ');
              if bestMethod = H_Move3Menu then
                if isMoving then
                  resultSTR := 'Next, choose Paste from the Edit menu to paste the cut passage where you want.'
                else
                  resultSTR := 'Next, choose Paste from the Edit menu to paste the cut passage where you want in relation to the other passage.'
              else if bestMethod = H_Move3Key then
                if isMoving then
                  resultSTR := 'Next, simultaneously press the command (clover) and v keys to paste the cut passage where you want.'
                else
                  resultSTR := 'Next, simultaneously press the command (clover) and v keys to paste the cut passage where you want in relation
to the other passage.'
              else if isMoving then
                resultSTR := 'Next, click in the Clip->Text rectangle in the palette to paste the cut passage where you want.'
              else
                resultSTR := 'Next, click in the Clip->Text rectangle in the palette to paste the cut passage where you want in relation to the other
passage.'
            end;
          H_Move4Menu, H_Move4Key, H_Move4Palette:
            begin
              MoveMess6;
              careIt := concat(careIt, 'then press the delete key ');
            end;
          H_Move5Menu, H_Move5Key, H_Move5Palette:
            begin
              MoveMess6;
              if bestMethod = H_Move5Menu then
                if isMoving then
                  resultSTR := 'Next, choose Paste from the Edit menu to paste the cut passage where you want.'
                else
                  resultSTR := 'Next, choose Paste from the Edit menu to paste the cut passage where you want in relation to the other passage.'
              else if bestMethod = H_Move5Key then
                if isMoving then
                  resultSTR := 'Next, simultaneously press the command (clover) and v keys to paste the cut passage where you want.'
                else
                  resultSTR := 'Next, simultaneously press the command (clover) and v keys to paste the cut passage where you want in relation
to the other passage.'
              else if isMoving then
                resultSTR := 'Next, click in the Clip->Text rectangle in the palette to paste the cut passage where you want.'
              else
                resultSTR := 'Next, click in the Clip->Text rectangle in the palette to paste the cut passage where you want in relation to the other
passage.';
              careIt := concat(careIt, 'then choose Delete or click in the delete rectangle in the palette (the rectangle with the letters struck-
through) ');
            end;
          H_Move6Menu, H_Move6Key, H_Move6Palette:
            begin
              MoveMess6;
              careIt := concat(careIt, 'then choose Delete or click in the delete rectangle in the palette (the rectangle with the letters struck-
through) ');
            end;
          H_Move7Menu:
            MoveMess1Menu;
          H_Move7Palette:
            MoveMess1Palette;
          H_Move8Menu:
            MoveMess2Menu;
          H_Move8Palette:
            MoveMess2Palette;
          otherwise

        end;
      end
```

```
{ if not, see if they've duplicated: }
    else if HR^.^ HRnumDuplicate <> 0 then
      begin
        bestMethod := bestDuplicate;
        caret1 := 'duplicate text';
        case bestMethod of
          H_Duplicate1:
            begin
              MoveMess1Menu;
              caret1 := concat(caret1, ', except the passage will occupy only one position in your document');
            end;
          H_Duplicate2:
            begin
              MoveMess2Menu;
              caret1 := concat(caret1, ', except the passage will occupy only one position in your document');
            end;
          H_Duplicate3Menu, H_Duplicate3Key, H_Duplicate3Palette:
            begin
              MoveMess6;
              if bestMethod = H_Duplicate3Menu then
                if isMoving then
                  resultSTR := 'Next, choose Paste from the Edit menu to paste the cut passage where you want.'
                else
                  resultSTR := 'Next, choose Paste from the Edit menu to paste the cut passage where you want in relation to the other passage.'
              else if bestMethod = H_Duplicate3Key then
                if isMoving then
                  resultSTR := 'Next, simultaneously press the command (clover) and v keys to paste the cut passage where you want.'
                else
                  resultSTR := 'Next, simultaneously press the command (clover) and v keys to paste the cut passage where you want in relation
to the other passage.'
              else if isMoving then
                resultSTR := 'Next, click in the Clip->Text rectangle in the palette to paste the cut passage where you want.'
              else
                resultSTR := 'Next, click in the Clip->Text rectangle in the palette to paste the cut passage where you want in relation to the other
passage.';
              caret3 := concat(caret3, 'then press the delete key.');
              if isMoving then
                caret1 := concat(caret1, '  To move the passage, you will cut and then paste it')
              else
                caret1 := concat(caret1, '  To switch two passages, you will cut one and then paste it');
            end;
          H_Duplicate4Menu, H_Duplicate4Key, H_Duplicate4Palette:
            begin
              MoveMess6;
              caret3 := concat(caret3, 'then press the delete key.');
              if isMoving then
                caret1 := concat(caret1, '  To move the passage, you will cut and then paste it')
              else
                caret1 := concat(caret1, '  To switch two passages, you will cut one and then paste it');
            end;
          otherwise

        end;
      end

{ if not, see if they've cut: }
    else if HR^.^ HRnumCut <> 0 then
      begin
        bestMethod := bestCut;
        caret1 := 'cut text, except you will move the insertion point and paste after you cut';
        case bestMethod of
          H_Cut1Menu, H_Cut1Key, H_Cut1Palette:
            begin
              MoveMess3;
              if bestMethod = H_Cut1Menu then
                resultSTR := 'Choose Paste from the Edit menu to paste the cut passage at the insertion point (where you clicked).'
              else if bestMethod = H_Cut1Key then
                resultSTR := 'Simultaneously press the command (clover) and v keys to paste the cut passage at the insertion point (where you
clicked)'
              else
                resultSTR := 'Click in the Clip->Text rectangle in the palette to paste the cut passage at the insertion point (where you clicked)'
            end;
          H_Cut2Menu:
            begin
              GetIndString(reminderStr, HelpCCSTR, reminder1);
              if isMoving then
                begin
                  firstStr := 'First, choose Cut from the Edit menu and follow the instructions in the Instructions window to cut the passage you
want to move.  Click in the Okay button in the palette when you are done.';
                  nextStr := 'After you have clicked in the Okay button, click where you want the text from the Clipboard to be.';
                  resultSTR := 'Finally, choose Paste from the Edit menu to paste the cut passage where you clicked.'
                end
              else
                begin
                  firstStr := 'First, choose Cut from the Edit menu and follow the instructions in the Instructions window to cut one of the passages
you want to switch.  Click in the Okay button in the palette when you are done.';
                  nextStr := 'After you have clicked in the Okay button, click where you want the text from the Clipboard to be in relation to the
other passage.';
                  resultSTR := 'Finally, choose Paste from the Edit menu to paste the cut passage where you clicked.'
                end
            end;
          H_Cut2Key:
            begin
              GetIndString(reminderStr, HelpCCSTR, reminder1);
              if isMoving then
                begin
                  firstStr := 'First, simultaneously press the command (clover) and x keys and follow the instructions in the Instructions window
to cut the passage you want to move.  Click in the Okay button in the palette when you are done.';
                  nextSTR := 'After you have clicked in the Okay button, click where you want the text from the Clipboard to be.';
                  resultSTR := 'Finally, simultaneously press the command (clover) and v keys to paste the cut passage where you clicked.';
                end
              else
                begin
                  firstStr := 'First, simultaneously press the command (clover) and x keys and follow the instructions in the Instructions window
to cut the passage you want to move.  Click in the Okay button in the palette when you are done.';
                  nextSTR := 'After you have clicked in the Okay button, click where you want the text from the Clipboard to be in relation to the
other passage.';
```

```
                        resultSTR := 'Finally, simultaneously press the command (clover) and v keys to paste the cut passage where you clicked.';
                      end
                  end;
                H_Cut2Palette:
                  begin
                    GetIndString(reminderStr, HelpCCSTR, reminder1);
                    if isMoving then
                      begin
                        firstStr := 'First, click in the scissors rectangle in the palette and follow the instructions in the instructions window to cut the
passage you want to move.  Click in the Okay button in the palette when you are done.';
                        nextSTR := 'After you have clicked in the Okay button, click where you want the text from the Clipboard to be.';
                        resultSTR := 'Finally, click in the Clip->Text rectangle in the palette to paste the cut passage where you clicked.';
                      end
                    else
                      begin
                        firstStr := 'First, click in the scissors rectangle in the palette and follow the instructions in the instructions window to cut the
passage you want to move.  Click in the Okay button in the palette when you are done.';
                        nextSTR := 'After you have clicked in the Okay button, click where you want the text from the Clipboard to be in relation to the
other passage.';
                        resultSTR := 'Finally, click in the Clip->Text rectangle in the palette to paste the cut passage where you clicked.';
                      end
                  end;
                otherwise

              end;
            end

  { not, see if they've copied: }
        else if HR^^ HRnumCopy <> 0 then
          begin
            bestMethod := bestCopy;
            caret1 := 'copy text, except you will choose Cut rather than Copy, then move the insertion point and paste';
            case bestMethod of
              H_Copy1Menu, H_Copy1Key, H_Copy1Palette:
                begin
                  MoveMeta3:
                  if bestMethod = H_Copy1Menu then
                    resultSTR := 'Choose Paste from the Edit menu to paste the cut passage at the insertion point (where you clicked).'
                  else if bestMethod = H_Copy1Key then
                    resultSTR := 'Simultaneously press the command (clover) and v keys to paste the cut passage at the insertion point (where you
clicked)'
                  else
                    resultSTR := 'Click in the Clip->Text rectangle in the palette to paste the cut passage at the insertion point (where you clicked)'
                end;
              H_Copy2Menu:
                begin
                  GetIndString(reminderStr, HelpCCSTR, reminder1);
                  if isMoving then
                    begin
                      firstStr := 'First, choose Cut from the Edit menu and follow the instructions in the instructions window to cut the passage you
want to move.  Click in the Okay button in the palette when you are done.';
                      nextStr := 'After you have clicked in the Okay button, click where you want the text from the Clipboard to be.';
                      resultSTR := 'Finally, choose Paste from the Edit menu to paste the cut passage where you clicked.'
                    end
                  else
                    begin
                      firstStr := 'First, choose Cut from the Edit menu and follow the instructions in the instructions window to cut one of the passages
you want to switch.  Click in the Okay button in the palette when you are done.';
                      nextStr := 'After you have clicked in the Okay button, click where you want the text from the Clipboard to be in relation to the
other passage.';
                      resultSTR := 'Finally, choose Paste from the Edit menu to paste the cut passage where you clicked.'
                    end
                end;
              H_Copy2Key:
                begin
                  GetIndString(reminderStr, HelpCCSTR, reminder1);
                  if isMoving then
                    begin
                      firstStr := 'First, simultaneously press the command (clover) and x keys and follow the instructions in the instructions window
to cut the passage you want to move.  Click in the Okay button in the palette when you are done.';
                      nextSTR := 'After you have clicked in the Okay button, click where you want the text from the Clipboard to be.';
                      resultSTR := 'Finally, simultaneously press the command (clover) and v keys to paste the cut passage where you clicked.';
                    end
                  else
                    begin
                      firstStr := 'First, simultaneously press the command (clover) and x keys and follow the instructions in the instructions window
to cut the passage you want to move.  Click in the Okay button in the palette when you are done.';
                      nextSTR := 'After you have clicked in the Okay button, click where you want the text from the Clipboard to be in relation to the
other passage.';
                      resultSTR := 'Finally, simultaneously press the command (clover) and v keys to paste the cut passage where you clicked.';
                    end
                end;
              H_Copy2Palette:
                begin
                  GetIndString(reminderStr, HelpCCSTR, reminder1);
                  if isMoving then
                    begin
                      firstStr := 'First, click in the scissors rectangle in the palette and follow the instructions in the instructions window to cut the
passage you want to move.  Click in the Okay button in the palette when you are done.';
                      nextSTR := 'After you have clicked in the Okay button, click where you want the text from the Clipboard to be.';
                      resultSTR := 'Finally, click in the Clip->Text rectangle in the palette to paste the cut passage where you clicked.';
                    end
                  else
                    begin
                      firstStr := 'First, click in the scissors rectangle in the palette and follow the instructions in the instructions window to cut the
passage you want to move.  Click in the Okay button in the palette when you are done.';
                      nextSTR := 'After you have clicked in the Okay button, click where you want the text from the Clipboard to be in relation to the
other passage.';
                      resultSTR := 'Finally, click in the Clip->Text rectangle in the palette to paste the cut passage where you clicked.';
                    end
                end;
              otherwise

            end;
          end

  { not, see if they've replaced: }
        else if HR^^ HRnumReplace <> 0 then
```

```
        begin
          bestMethod := bestReplace78;
          if bestMethod <> H_neverTried then
            begin
              caret1 := 'replace text, except you will choose a place for the selected text to move rather than typing new text';
              case bestMethod of
                H_Replace7Menu:
                  MoveMess1Menu;
                H_Replace7Palette:
                  MoveMess1Palette;
                H_Replace8Menu:
                  MoveMess2Menu;
                H_Replace8Palette:
                  MoveMess2Palette;
                otherwise

              end;
            end;
        end;

{Otherwise, if they've never tried any of the above things, }
      if bestMethod = H_neverTried then
        begin
          MoveMess2Palette;
          caret1 := '';
          GetIndString(firstStr, HelpCCSTR, reminder2);
        end;

{Get the dialog box: }
      DLOGDone := FALSE;
      if isMoving then
        begin
          dPtr := GetNewDialog(MovingDLOG, nil, pointer(-1));
          caret0 := 'move'
        end
      else
        begin
          dPtr := GetNewDialog(SwitchingDLOG, nil, pointer(-1));
          caret0 := 'switch'
        end;
      SetPort(dPtr);
      TextFont(geneva);
      TextSize(10);
      ParamText(caret0, caret1, caret2, caret3);

{Do the dialog text: }
      GetDItem(dPtr, reminderItem, itemType, textHandle, box);
      SetIText(textHandle, reminderStr);

      GetDItem(dPtr, firstItem, itemType, textHandle, box);
      SetIText(textHandle, firstStr);

      GetDItem(dPtr, nextItem, itemType, textHandle, box);
      SetIText(textHandle, nextStr);

      GetDItem(dPtr, resultItem, itemType, textHandle, box);
      SetIText(textHandle, resultStr);

      ShowWindow(dPtr);
      FrameDItem(dPtr, DoneButton);

{Wait until Done is pressed: }
      while not DLOGDone do
        begin
          ModalDialog(nil, hit);
          if (hit = DoneButton) then
            DLOGDone := TRUE;
        end;
      DisposDialog(dPtr);
    end;

{-----------------------------------------------------------------------------
HelpReplacing gives the user help when they've chosen Help->Editing->replacing or rewording or
correcting errors.
-----------------------------------------------------------------------------}

  procedure helpReplacing;    {(itemMenu:boolean;  isCorrecting:boolean;  isRewording:boolean)
                               {isCorrecting means chose correcting typos, isrewording means chose reword}
                               {neither means chose Replace}
    const
      DoneButton = 1;
    var
      dPtr: DialogPtr;
      hit, itemType: integer;
      DLOGDone: boolean;
      bestMethod: probeType;
      textHandle: Handle;
      box: Rect;
      reminderStr, firstStr, nextStr, resultStr, caret0, caret1, caret2, caret3, selectStr: Str255;

      procedure Replace1Mess;
      begin
        if isCorrecting then
          caret2 := 'your error'
        else if isRewording then
          caret2 := 'the passage you want to reword'
        else
          caret2 := 'the passage you want to replace';
        GetIndString(firstStr, HelpReplaceSTR, firstReplace);
        GetIndString(nextStr, HelpReplaceSTR, nextReplace);
        GetIndString(resultStr, HelpReplaceSTR, resultReplace);
        caret3 := '';
      end;

      procedure Replace2Mess;
      begin
        if isCorrecting then
          caret2 := 'your error'
```

```
    else if isRewording then
      caret2 := 'the passage you want to reword'
    else
      caret2 := 'the passage you want to replace';
    GetIndString(firstStr, HelpReplaceSTR, first2replace);
    firstStr := concat(firstStr, selectStr);
    GetIndString(nextStr, HelpReplaceSTR, next2replace);
    GetIndString(resultStr, HelpReplaceSTR, result1replace);
    caret3 := '';
  end;

  procedure Replace3Mess (fromMenu: boolean);
  begin
    if isCorrecting then
      caret2 := 'your error'
    else if isRewording then
      caret2 := 'the passage you want to reword'
    else
      caret2 := 'the passage you want to replace';
    GetIndString(firstStr, HelpReplaceSTR, first2replace);
    firstStr := concat(firstStr, selectStr);
    GetIndString(nextStr, HelpReplaceSTR, next3replace);
    if fromMenu then
      caret3 := 'choose Delete from the Edit menu'
    else
      caret3 := 'click in the delete rectangle in the palette (the delete rectangle consists of text that has been struck-through)';
    GetIndString(resultStr, HelpReplaceSTR, result1replace);
  end;

  procedure Replace4Mess (fromMenu: boolean);
  begin
    caret2 := '';
    if fromMenu then
      caret3 := 'choose Delete from the Edit menu'
    else
      caret3 := 'click in the delete rectangle in the palette (the delete rectangle consists of text that has been struck-through)';
    GetIndString(nextStr, HelpReplaceSTR, first4replace);
    GetIndString(resultStr, HelpFormatSTR, result3)
  end;

  procedure Replace5Mess;
  begin
    firstStr := '';
    caret3 := '';
    GetIndString(nextStr, HelpReplaceSTR, first3replace);
    if isCorrecting then
      caret2 := 'Cut the your error'
    else if isRewording then
      caret2 := 'Cut the passage you want to reword'
    else
      caret2 := 'Cut the passage you want to replace';
    GetIndString(resultStr, HelpReplaceSTR, result1replace);
  end;

  procedure Replace6Mess;
  begin
    firstStr := '';
    caret3 := '';
    if isCorrecting then
      caret2 := 'your error'
    else if isRewording then
      caret2 := 'the passage you want to reword'
    else
      caret2 := 'the passage you want to replace';
    GetIndString(nextStr, HelpReplaceSTR, first2replace);
    nextStr := concat(nextStr, selectStr);
    GetIndString(resultStr, HelpReplaceSTR, result1replace);
  end;

  procedure Replace7Mess (fromMenu: boolean);
  begin
    if isCorrecting then
      caret2 := 'your error'
    else if isRewording then
      caret2 := 'the passage you want to reword'
    else
      caret2 := 'the passage you want to replace';
    GetIndString(firstStr, HelpReplaceSTR, first2replace);
    firstStr := concat(firstStr, selectStr);
    GetIndString(nextStr, HelpReplaceSTR, next3replace);
    if fromMenu then
      caret3 := 'choose Replace from the Edit menu'
    else
      caret3 := 'click in the replace rectangle in the palette (the replace rectangle consists of text that has been struck-through, with circled text being
nserted)';
    GetIndString(resultStr, HelpFormatSTR, result3);
  end;

  procedure Replace8Mess (fromMenu: boolean);
  begin
    if isCorrecting then
      caret2 := 'your error'
    else if isRewording then
      caret2 := 'the passage you want to reword'
    else
      caret2 := 'the passage you want to replace';
    GetIndString(nextStr, HelpReplaceSTR, first4replace);
    if fromMenu then
      caret3 := 'choose Replace from the Edit menu'
    else
      caret3 := 'click in the replace rectangle in the palette (the replace rectangle consists of text that has been struck-through, with circled text being
nserted)';
    GetIndString(resultStr, HelpFormatSTR, result3);
  end;

begin
create help replacing message: }
  bestMethod := bestSelect;
```

```
    case bestMethod of
      H_SelectMenu:
        GetIndString(selectStr, HelpFormatSTR, select3);
      H_SelectMouse:
        GetIndString(selectStr, HelpFormatSTR, select4);
      H_SelectKey:
        GetIndString(selectStr, HelpFormatSTR, select5);
      otherwise
        if fromMenu then
          GetIndString(selectStr, HelpFormatSTR, select1)
        else
          GetIndString(selectStr, HelpFormatSTR, select2)
    end;
    bestMethod := H_neverTried;

{Check to see if they've replaced before: }
    if HR^^ HRnumReplace <> 0 then
      begin
        bestMethod := bestReplace;
        careIt := 'replace text previously';
        GetIndString(reminderStr, HelpCCSTR, reminder1);
        case bestMethod of
          H_Replace1:
            Replace1Mess;
          H_Replace2:
            Replace2Mess;
          H_Replace3Menu:
            Replace3Mess(TRUE);
          H_Replace3Palette:
            Replace3Mess(FALSE);
          H_Replace4Menu:
            begin
              Replace4Mess(TRUE);
              reminderStr := '';
              GetIndString(firstStr, HelpCCSTR, reminder1);
            end;
          H_Replace4Palette:
            begin
              Replace4Mess(FALSE);
              reminderStr := '';
              GetIndString(firstStr, HelpCCSTR, reminder1);
            end;
          H_Replace5:
            Replace5Mess;
          H_Replace6:
            Replace6Mess;
          H_Replace7Menu:
            Replace7Mess(TRUE);
          H_Replace7Palette:
            Replace7Mess(FALSE);
          H_Replace8Menu:
            begin
              Replace8Mess(TRUE);
              reminderStr := '';
              GetIndString(firstStr, HelpCCSTR, reminder1);
            end;
          H_Replace8Palette:
            begin
              Replace8Mess(FALSE);
              reminderStr := '';
              GetIndString(firstStr, HelpCCSTR, reminder1);
            end;
          otherwise

        end;
      end

{ not, see if they've deleted: }
    else if HR^^ HRnumDelete <> 0 then
      begin
        bestMethod := bestDelete;
        GetIndString(reminderStr, HelpCCSTR, reminder1);
        careIt := 'delete text, except you will retype the passage after you delete it';
        case bestMethod of
          H_Delete1:
            Replace1Mess;
          H_Delete2:
            Replace2Mess;
          H_Delete3Menu:
            Replace3Mess(TRUE);
          H_Delete3Palette:
            Replace3Mess(FALSE);
          H_Delete4Menu:
            begin
              Replace4Mess(TRUE);
              reminderStr := '';
              GetIndString(firstStr, HelpCCSTR, reminder1);
            end;
          H_Delete4Palette:
            begin
              Replace4Mess(FALSE);
              reminderStr := '';
              GetIndString(firstStr, HelpCCSTR, reminder1);
            end;
          otherwise

        end;
      end

{ not, see if they've cut: }
    else if HR^^ HRnumCut <> 0 then
      begin
        bestMethod := bestCut;
        GetIndString(reminderStr, HelpCCSTR, reminder1);
        careIt := 'cut text, except you will retype the passage after you cut it';
        case bestMethod of
          H_Cut1Menu, H_Cut2Menu:
            begin
```

```
                    ReplaceAMess;
                    if isCorrecting then
                      caret2 := 'use the Edit menu to Cut the your error'
                    else if isRewording then
                      caret2 := 'use the Edit menu to Cut the passage you want to reword'
                    else
                      caret2 := 'use the Edit menu to Cut the passage you want to replace';
                  end;
                H_Cut1Key, H_Cut2Key:
                  begin
                    ReplaceAMess;
                    if isCorrecting then
                      caret2 := 'press command (clover) and x to Cut the your error'
                    else if isRewording then
                      caret2 := 'press command (clover) and x to Cut the passage you want to reword'
                    else
                      caret2 := 'press command (clover) and x to Cut the passage you want to replace';
                  end;
                H_Cut1Palette, H_Cut2Palette:
                  begin
                    ReplaceAMess;
                    if isCorrecting then
                      caret2 := 'click on the scissors in the palette to Cut the your error'
                    else if isRewording then
                      caret2 := 'click on the scissors in the palette to Cut the passage you want to reword'
                    else
                      caret2 := 'click on the scissors in the palette to Cut the passage you want to replace';
                  end;
                otherwise

              end;
            end;

Otherwise, if they've done none of the above: }
      if badMethod = H_neverTried then
        begin
          ReplaceAMess(FALSE);
          caret1 := '';
          reminderStr := '';
          GetIndString(firstStr, HelpCCSTR, reminder2);
        end;

Set the dialog box: }
      DLOGDone := FALSE;
      if isCorrecting then
        begin
          dPtr := GetNewDialog(SpellingDLOG, nil, pointer(-1));
          caret0 := 'correct minor errors in'
        end
      else if isRewording then
        begin
          dPtr := GetNewDialog(RewordingDLOG, nil, pointer(-1));
          caret0 := 'reword'
        end
      else
        begin
          dPtr := GetNewDialog(ReplacingDLOG, nil, pointer(-1));
          caret0 := 'replace'
        end;
      SetPort(dPtr);
      TextFont(geneva);
      TextSize(10);
      ParamText(caret0, caret1, caret2, caret3);

Do the dialog text: }
      GetDItem(dPtr, reminderItem, itemType, textHandle, box);
      SetText(textHandle, reminderStr);

      GetDItem(dPtr, firstItem, itemType, textHandle, box);
      SetText(textHandle, firstStr);

      GetDItem(dPtr, nextItem, itemType, textHandle, box);
      SetText(textHandle, nextStr);

      GetDItem(dPtr, resultItem, itemType, textHandle, box);
      SetText(textHandle, resultStr);

      ShowWindow(dPtr);
      PenNormal;
      FrameDItem(dPtr, DoneButton);

Wait until Done is pressed: }
      while not DLOGDone do
        begin
          ModalDialog(nil, hit);
          if (hit = DoneButton) then
            DLOGDone := TRUE;
        end;
      DisposDialog(dPtr);
    end;
end
```

```
{
  Margaret Stone
  Sc.M. Project, Brown University

  This unit contains the code to provide help to the user when the user asks for help copying text,
  or pasting text.
}

unit EditHelp3;

{ Interface Section }

interface

  uses
    EditorHelpInt, EditorUtilities, HelpBasics;

  procedure helpCopying (fromMenu: boolean);
  procedure helpPasting (fromMenu: boolean);

{ Implementation Section }

implementation

  const
{string indices in string list HelpFormatSTR:}
      select1 = 4;
      select2 = 5;
      select3 = 6;
      select4 = 7;
      select5 = 8;
      result3 = 13;

{string indices in string list HelpCCSTR:}
      reminder1 = 1;
      reminder2 = 2;
      first1 = 3;
      firstNext = 4;
      result1 = 5;
      result2 = 6;

{string indices in string list HelpPasteSTR:}
      first = 1;
      result = 2;
      dupResult1 = 3;
      dupResult2 = 4;
      dupResult3 = 5;
      dupResult4 = 6;
      dupResult5 = 7;

{item numbers in dialog box:}
      reminderItem = 3;
      firstItem = 4;
      nextItem = 5;
      resultItem = 6;


{ helpCopying gives the user help when they've chosen Help->Editing->copying text. }

  procedure helpCopying;      {fromMenu:boolean}
    const
      DoneButton = 1;
    var
      dPtr: DialogPtr;
      hit, itemType: integer;
      OLCXDone: boolean;
      possibleHost: profileType;
      textHandle: Handle;
      box: Rect;
      reminderStr, firstStr, nextStr, resultStr, caret0, caret1, caret2, caret3, selectStr: Str255;

    procedure CopyMess1Menu;
    begin
      GetIndString(resultStr, HelpCCSTR, result2);
      caret2 := 'Next';
      caret3 := 'choose Copy from the Edit menu';
      GetIndString(firstStr, HelpCCSTR, first1);
      GetIndString(nextStr, HelpCCSTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpCCSTR, reminder1);
    end;

    procedure CopyMess1Palette;
    begin
      GetIndString(resultStr, HelpCCSTR, result2);
      caret2 := 'Next';
      caret3 := 'click on the Text->Clip rectangle in the palette';
      GetIndString(firstStr, HelpCCSTR, first1);
      GetIndString(nextStr, HelpCCSTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpCCSTR, reminder1);
    end;

    procedure CopyMess1Key;
    begin
      GetIndString(resultStr, HelpCCSTR, result2);
      caret2 := 'Next';
      caret3 := 'press the command (clover) and c keys simultaneously on the keyboard';
      GetIndString(firstStr, HelpCCSTR, first1);
      GetIndString(nextStr, HelpCCSTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpCCSTR, reminder1);
```

```
    end;

    procedure CopyMess2Menu;
    begin
      GetIndString(resultStr, HelpFormatSTR, result3);
      caret2  := 'First';
      reminderStr := ''
      caret3 := 'choose Copy from the Edit menu';
      GetIndString(nextStr, HelpCCSTR, firstNext);
      GetIndString(firstStr, HelpCCSTR, reminder1);
    end;

    procedure CopyMess2Palette;
    begin
      GetIndString(resultStr, HelpFormatSTR, result3);
      caret2 := 'First';
      caret3 := 'click on the Text->Clip rectangle in the palette';
      reminderStr := '';
      GetIndString(nextStr, HelpCCSTR, firstNext);
      GetIndString(firstStr, HelpCCSTR, reminder1);
    end;

    procedure CopyMess2Key;
    begin
      GetIndString(resultStr, HelpFormatSTR, result3);
      caret2 := 'First';
      reminderStr := '';
      caret3 := 'press the command (clover) and c keys simultaneously on the keyboard';
      GetIndString(nextStr, HelpCCSTR, firstNext);
      GetIndString(firstStr, HelpCCSTR, reminder1);
    end;

    procedure CopyMess3;
    begin
      GetIndString(resultStr, HelpCCSTR, result2);
      caret2 := 'Next';
      GetIndString(firstStr, HelpCCSTR, first1);
      GetIndString(nextStr, HelpCCSTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpCCSTR, reminder1);
      if menuPref then
        caret3 := 'choose Copy from the Edit menu'
      else
        caret3 := 'click on the Text->Clip rectangle in the palette'
    end;


  begin
{create help copying message: }
    bestMethod := bestSelect;
    case bestMethod of
      H_SelectMenu:
        GetIndString(selectStr, HelpFormatSTR, select3);
      H_SelectMouse:
        GetIndString(selectStr, HelpFormatSTR, select4);
      H_SelectKey:
        GetIndString(selectStr, HelpFormatSTR, select5);
      otherwise
        if fromMenu then
          GetIndString(selectStr, HelpFormatSTR, select1)
        else
          GetIndString(selectStr, HelpFormatSTR, select2)
    end;
    bestMethod := H_neverTried;

{Check to see if the user's copied before: }
    {HR** HPInumCopy <> 0 then
      begin
        bestMethod := bestCopy;
        caret1 := 'copy text previously';
        case bestMethod of
          H_copy1menu:
            CopyMess1Menu;
          H_copy1key:
            CopyMess1Key;
          H_copy1palette:
            CopyMess1Palette;
          H_copy2menu:
            CopyMess2Menu;
          H_copy2key:
            CopyMess2Key;
          H_copy2palette:
            CopyMess2Palette;
          otherwise

        end
      end

  { not, see if they've cut before: }
    else if HR** HPInumCut <> 0 then
      begin
        bestMethod := bestCut;
        caret1  := 'cut text, except you will ';
        case bestMethod of
          H_cut1menu:
            begin
              CopyMess1Menu;
              caret1 := concat(caret1, 'choose Copy rather than Cut from the Edit menu');
            end;
          H_cut1key:
            begin
              CopyMess1Key;
              caret1 := concat(caret1, 'press Command-c rather than Command-x');
            end;
          H_cut1palette:
            begin
              CopyMess1Palette;
              caret1 := concat(caret1, 'click on the Text->Clip rectangle rather than the scissors icon');
```

```
                end;
            H_cut2menu:
                begin
                    CopyMess2Menu;
                    caret1 := concat(caret1, 'choose Copy rather than Cut from the Edit menu\');
                end;
            H_cut2key:
                begin
                    CopyMess2Key;
                    caret1 := concat(caret1, 'press Command-c rather than Command-x');
                end;
            H_cut2palette:
                begin
                    CopyMess2Palette;
                    caret1 := concat(caret1, 'click on the Text->Clip rectangle rather than the scissors icon');
                end;
            otherwise

        end
    end

{ not, see if they've deleted before: }
    else if HR^^ HPrnumDelete <> 0 then
        begin
            bestMethod := bestDelete34;
            caret1 := 'delete text, except you will ';
            if bestMethod <> H_neverTried then
                begin
                    caret1 := 'delete text';
                    case bestMethod of
                        H_delete3menu:
                            begin
                                CopyMess1Menu;
                                caret1 := concat(caret1, 'choose Copy rather than Delete from the Edit menu\');
                            end;
                        H_delete3palette:
                            begin
                                CopyMess1Palette;
                                caret1 := concat(caret1, 'click on the Text->Clip rectangle rather than the delete icon');
                            end;
                        H_delete4menu:
                            begin
                                CopyMess2Menu;
                                caret1 := concat(caret1, 'choose Copy rather than Delete from the Edit menu\');
                            end;
                        H_delete4palette:
                            begin
                                CopyMess2Palette;
                                caret1 := concat(caret1, 'click on the Text->Clip rectangle rather than the delete icon');
                            end;
                        otherwise

                    end
                end

{ not, see if they've change the text style before: }
            else if HR^^ HPrnumStyle <> 0 then
                begin
                    bestMethod := bestStyle;
                    caret1 := 'change the lettering style, except you will ';
                    case bestMethod of
                        H_Text3Menu:
                            begin
                                CopyMess1Menu;
                                caret1 := concat(caret1, 'choose Copy from the Edit menu, rather than a style from the Set Text Style menu\');
                            end;
                        H_Text3Palette:
                            begin
                                CopyMess1Palette;
                                caret1 := concat(caret1, 'click on the Text->Clip rectangle in the palette, rather than a style');
                            end;
                        H_Text5Menu, H_Text6Menu:
                            begin
                                CopyMess2Menu;
                                caret1 := concat(caret1, 'choose Copy from the Edit menu, rather than a style from the Set Text Style menu\');
                            end;
                        H_Text5Palette, H_Text6Palette:
                            begin
                                CopyMess2Palette;
                                caret1 := concat(caret1, 'click on the Text->Clip rectangle in the palette, rather than a style');
                            end;
                    end
                end

{ not, see if they've changes the text font before: }
            else if HR^^ HPrnumFont <> 0 then
                begin
                    bestMethod := bestStyle;
                    caret1 := 'change the typeface, except you will ';
                    case bestMethod of
                        H_Text2Menu:
                            begin
                                CopyMess1Menu;
                                caret1 := concat(caret1, 'choose Copy from the Edit menu, rather than a typeface from the Set Text Style menu\');
                            end;
                        H_Text2Palette:
                            begin
                                CopyMess1Palette;
                                caret1 := concat(caret1, 'click on the Text->Clip rectangle in the palette, rather than a typeface');
                            end;
                        H_Text5Menu, H_Text6Menu:
                            begin
                                CopyMess2Menu;
                                caret1 := concat(caret1, 'choose Copy from the Edit menu, rather than a typeface from the Set Text Style menu\');
                            end;
                        H_Text5Palette, H_Text6Palette:
                            begin
                                CopyMess2Palette;
```

```
                    caret1 := concat(caret1, 'click on the Text->Clip rectangle in the palette, rather than a typeface');
                  end;
              end
            end

{ If not, see if they've changed the text size before: }
          else if HR^^.HRnumSize <> 0 then
            begin
              bestMethod := bestStyle;
              caret1 := 'change the lettering size, except you will ';
              case bestMethod of
                H_Text1Menu:
                  begin
                    CopyMess1Menu;
                    caret1 := concat(caret1, 'choose Copy from the Edit menu, rather than a size from the Set Text Style menu')
                  end;
                H_Text1Palette:
                  begin
                    CopyMess1Palette;
                    caret1 := concat(caret1, 'click on the Text->Clip rectangle in the palette, rather than a size');
                  end;
                H_Text7Menu, H_Text4Menu:
                  begin
                    CopyMess2Menu;
                    caret1 := concat(caret1, 'choose Copy from the Edit menu, rather than a size from the Set Text Style menu')
                  end;
                H_Text7Palette, H_Text4Palette:
                  begin
                    CopyMess2Palette;
                    caret1 := concat(caret1, 'click on the Text->Clip rectangle in the palette, rather than a size');
                  end;
              end
            end

{ If not, see if they've moved before: }
          else if HR^^.HRnumMove <> 0 then
            begin
              bestMethod := bestMove78;
              if bestMethod <> H_neverTried then
                begin
                  caret1 := 'move text, except you will ';
                  case bestMethod of
                    H_move7menu:
                      begin
                        CopyMess1Menu;
                        caret1 := concat(caret1, 'choose Copy rather than Move from the Edit menu');
                      end;
                    H_move7palette:
                      begin
                        CopyMess1Palette;
                        caret1 := concat(caret1, 'click on the Text->Clip rectangle in the palette, rather than the move icon');
                      end;
                    H_move8menu:
                      begin
                        CopyMess2Menu;
                        caret1 := concat(caret1, 'choose Copy rather than Move from the Edit menu');
                      end;
                    H_move8palette:
                      begin
                        CopyMess2Palette;
                        caret1 := concat(caret1, 'click on the Text->Clip rectangle in the palette, rather than the move icon');
                      end;
                    otherwise

                  end
                end

{ If not, see if they've replaced before: }
              else if HR^^.HRnumReplace <> 0 then
                begin
                  bestMethod := bestReplace61;
                  if bestMethod <> H_neverTried then
                    begin
                      caret1 := 'replace text, except you will ';
                      case bestMethod of
                        H_replace2, H_replace6:
                          begin
                            CopyMess3;
                            caret1 := concat(caret1, 'choose Copy rather than typing or pressing delete');
                          end;
                        H_replace3menu, H_replace7menu:
                          begin
                            CopyMess1Menu;
                            caret1 := concat(caret1, 'choose Copy rather than Replace from the Edit menu');
                          end;
                        H_replace3palette, H_replace7palette:
                          begin
                            CopyMess1Palette;
                            caret1 := concat(caret1, 'click on the Text->Clip rectangle in the palette, rather than the replace icon');
                          end;
                        H_replace8palette:
                          begin
                            CopyMess2Palette;
                            caret1 := concat(caret1, 'click on the Text->Clip rectangle in the palette, rather than the replace icon')
                          end;
                        H_replace8menu:
                          begin
                            CopyMess2Menu;
                            caret1 := concat(caret1, 'choose Copy rather than Replace from the Edit menu');
                          end;
                        otherwise

                      end
                    end

{ If not, see if they've duplicated before: }
                  else if HR^^.HRnumDuplicate <> 0 then
                    begin
```

```
                        bestMethod := bestDuplicate;
                        caret1 := 'duplicate text, except you will choose Copy rather than Duplicate from the Edit menu';
                        case bestMethod of
                          H_duplicate1:
                            CopyMess1Menu;
                          H_duplicate2:
                            CopyMess2Menu;
                          otherwise

                        end
                      end
                    end
                  end
              end;

{ none of the above: }
      if bestMethod = H_neverTried then
        begin
          reminderStr := '';
          GetIndString(firstStr, HelpCCSTR, remainder2);
          GetIndString(nextStr, HelpCCSTR, firstNext);
          GetIndString(resultStr, HelpFormatSTR, result3);
          caret1 := '';
          caret2 := 'First';
          if not fromMenu then
            caret3 := 'click on the Text->Clip rectangle in the palette'
          else
            caret3 := 'Choose Copy from the Edit menu';
        end;

{ Get the dialog box: }
      DLOGDone := FALSE;
      dPtr := GetNewDialog(CopyingDLOG, nil, pointer(-1));
      SetPort(dPtr);
      TextFont(geneva);
      TextSize(10);
      caret0 := 'copy';
      ParamText(caret0, caret1, caret2, caret3);

{ Do the dialog text: }
      GetDItem(dPtr, reminderItem, itemType, textHandle, box);
      SetIText(textHandle, reminderStr);

      GetDItem(dPtr, firstItem, itemType, textHandle, box);
      SetIText(textHandle, firstStr);

      GetDItem(dPtr, nextItem, itemType, textHandle, box);
      SetIText(textHandle, nextStr);

      GetDItem(dPtr, resultItem, itemType, textHandle, box);
      SetIText(textHandle, resultStr);

      ShowWindow(dPtr);
      PlotDItem(dPtr, DoneButton);

{ Wait until Done is pressed: }
      while not DLOGDone do
        begin
          ModalDialog(nil, hit);
          if (hit = DoneButton) then
            DLOGDone := TRUE;
        end;
      DisposDialog(dPtr);
    end;


{ ------------------------------------------------------------------------------ }
{ HelpPasting gives the user help when they've chosen Help->Editing->pasting text }
{ ------------------------------------------------------------------------------ }

procedure HelpPasting;          {fromMenu:boolean});
  const
    DoneButton = 1;
  var
    dPtr: DialogPtr;
    hit, itemType: integer;
    DLOGDone: boolean;
    bestMethod: profileType;
    textHandle: Handle;
    box: Rect;
    reminderStr, firstStr, nextStr, resultStr, caret0, caret1, caret2, caret3, selectStr: Str255;


  function bestText147: ProfileType;
    var
      numSelect, numChoose, numNative, numMenu, numPalette, total: integer;
      temp, temp2, max, tempMenu, tempPalette: real;
  begin
    numSelect := 0;
    numChoose := 0;
    numNative := 0;
    total := 0;
    max := 0;
    temp := 0;
    temp2 := 0;
    numMenu := 0;
    numPalette := 0;

    if HR^^.HRnumSize <> 0 then
      begin
        if HR^^.HRText1Palette <> 0 then
          begin
            numSelect := numSelect + HR^^.HRText1Palette;
            numPalette := numPalette + HR^^.HRText1Palette;
            total := total + HR^^.HRText1Palette
          end;
        if HR^^.HRText4Palette <> 0 then
          begin
```

```
            numChoose := numChoose + HR^^.HRText4Palette;
            numPalette := numPalette + HR^^.HRText4Palette;
            total := total + HR^^.HRText4Palette
          end;
        if HR^^.HRText7Palette <> 0 then
          begin
            numNative := numNative + HR^^.HRText7Palette;
            numPalette := numPalette + HR^^.HRText7Palette;
            total := total + HR^^.HRText7Palette
          end;
        if HR^^.HRText1Menu <> 0 then
          begin
            numSelect := numSelect + HR^^.HRText1Menu;
            numMenu := numMenu + HR^^.HRText1Menu;
            total := total + HR^^.HRText1Menu
          end;
        if HR^^.HRText4Menu <> 0 then
          begin
            numChoose := numChoose + HR^^.HRText4Menu;
            numMenu := numMenu + HR^^.HRText4Menu;
            total := total + HR^^.HRText4Menu
          end;
        if HR^^.HRText7Menu <> 0 then
          begin
            numNative := numNative + HR^^.HRText7Menu;
            numMenu := numMenu + HR^^.HRText7Menu;
            total := total + HR^^.HRText7Menu
          end;
      end;
    if HR^^.HRnumFont <> 0 then
      begin
        if HR^^.HRText2Palette <> 0 then
          begin
            numSelect := numSelect + HR^^.HRText2Palette;
            numPalette := numPalette + HR^^.HRText2Palette;
            total := total + HR^^.HRText2Palette
          end;
        if HR^^.HRText5Palette <> 0 then
          begin
            numChoose := numChoose + HR^^.HRText5Palette;
            numPalette := numPalette + HR^^.HRText5Palette;
            total := total + HR^^.HRText5Palette
          end;
        if HR^^.HRText8Palette <> 0 then
          begin
            numNative := numNative + HR^^.HRText8Palette;
            numPalette := numPalette + HR^^.HRText8Palette;
            total := total + HR^^.HRText8Palette
          end;
        if HR^^.HRText2Menu <> 0 then
          begin
            numSelect := numSelect + HR^^.HRText2Menu;
            numMenu := numMenu + HR^^.HRText2Menu;
            total := total + HR^^.HRText2Menu
          end;
        if HR^^.HRText5Menu <> 0 then
          begin
            numChoose := numChoose + HR^^.HRText5Menu;
            numMenu := numMenu + HR^^.HRText5Menu;
            total := total + HR^^.HRText5Menu
          end;
        if HR^^.HRText8Menu <> 0 then
          begin
            numNative := numNative + HR^^.HRText8Menu;
            numMenu := numMenu + HR^^.HRText8Menu;
            total := total + HR^^.HRText8Menu
          end;
      end;
    if HR^^.HRnumStyle <> 0 then
      begin
        if HR^^.HRText3Palette <> 0 then
          begin
            numSelect := numSelect + HR^^.HRText3Palette;
            numPalette := numPalette + HR^^.HRText3Palette;
            total := total + HR^^.HRText3Palette
          end;
        if HR^^.HRText6Palette <> 0 then
          begin
            numChoose := numChoose + HR^^.HRText6Palette;
            numPalette := numPalette + HR^^.HRText6Palette;
            total := total + HR^^.HRText6Palette
          end;
        if HR^^.HRText9Palette <> 0 then
          begin
            numNative := numNative + HR^^.HRText9Palette;
            numPalette := numPalette + HR^^.HRText9Palette;
            total := total + HR^^.HRText9Palette
          end;
        if HR^^.HRText3Menu <> 0 then
          begin
            numSelect := numSelect + HR^^.HRText3Menu;
            numMenu := numMenu + HR^^.HRText3Menu;
            total := total + HR^^.HRText3Menu
          end;
        if HR^^.HRText6Menu <> 0 then
          begin
            numChoose := numChoose + HR^^.HRText6Menu;
            numMenu := numMenu + HR^^.HRText6Menu;
            total := total + HR^^.HRText6Menu
          end;
        if HR^^.HRText9Menu <> 0 then
          begin
            numNative := numNative + HR^^.HRText9Menu;
            numMenu := numMenu + HR^^.HRText9Menu;
            total := total + HR^^.HRText9Menu
          end;
      end;
```

```
    if total <> 0 then
       begin
          if numSelect <> 0 then
             max := numSelect / total;
          if numChoose <> 0 then
             temp := numChoose / total;
          if numNative <> 0 then
             temp2 := numNative / total;
          if numMenu <> 0 then
             tempMenu := numMenu / total;
          if numPalette <> 0 then
             tempPalette := numPalette / total;

          bestText147 := H_neverTried;

          if temp > max then
             begin
                if temp2 > temp then
                   if tempMenu > tempPalette then
                      bestText147 := H_Text7Menu
                   else
                      bestText147 := H_Text7Palette
                else if tempMenu > tempPalette then
                   bestText147 := H_Text4Menu
                else
                   bestText147 := H_Text4Palette
             end
          else
             begin
                if temp2 > max then
                   if tempMenu > tempPalette then
                      bestText147 := H_Text7Menu
                   else
                      bestText147 := H_Text7Palette
                else if tempMenu > tempPalette then
                   bestText147 := H_Text1Menu
                else
                   bestText147 := H_Text1Palette
             end
       end;

  end;

  procedure PasteMess1Menu;
  begin
     GetIndString(resultStr, HelpPasteSTR, result);
     caret2 := 'Next';
     caret3 := 'choose Paste from the Edit menu';
     GetIndString(firstStr, HelpPasteSTR, first);
     GetIndString(nextStr, HelpCCSTR, firstNext);
     GetIndString(reminderStr, HelpCCSTR, reminder1);
  end;

  procedure PasteMess1Palette;
  begin
     GetIndString(resultStr, HelpPasteSTR, result);
     caret2 := 'Next';
     caret3 := 'click on the Clip->Text rectangle in the palette';
     GetIndString(firstStr, HelpPasteSTR, first);
     GetIndString(nextStr, HelpCCSTR, firstNext);
     GetIndString(reminderStr, HelpCCSTR, reminder1);
  end;

  procedure PasteMess1Key;
  begin
     GetIndString(resultStr, HelpPasteSTR, result);
     caret2 := 'Next';
     caret3 := 'press the command (clover) and v keys simultaneously on the keyboard';
     GetIndString(firstStr, HelpPasteSTR, first);
     GetIndString(nextStr, HelpCCSTR, firstNext);
     GetIndString(reminderStr, HelpCCSTR, reminder1);
  end;

  procedure PasteMess2Menu;
  begin
     GetIndString(resultStr, HelpFormatSTR, result3);
     caret2 := 'First';
     reminderStr := '';
     caret3 := 'choose Paste from the Edit menu';
     GetIndString(nextStr, HelpCCSTR, firstNext);
     GetIndString(firstStr, HelpCCSTR, reminder1);
  end;

  procedure PasteMess2Palette;
  begin
     GetIndString(resultStr, HelpFormatSTR, result3);
     caret2 := 'First';
     caret3 := 'click on the Clip->Text rectangle in the palette';
     reminderStr := '';
     GetIndString(nextStr, HelpCCSTR, firstNext);
     GetIndString(firstStr, HelpCCSTR, reminder1);
  end;

  procedure PasteMess2Key;
  begin
     GetIndString(resultStr, HelpFormatSTR, result3);
     caret2 := 'First';
     reminderStr := '';
     caret3 := 'press the command (clover) and v keys simultaneously on the keyboard';
     GetIndString(nextStr, HelpCCSTR, firstNext);
     GetIndString(firstStr, HelpCCSTR, reminder1);
  end;


begin
create help passing message: }
   bestMethod := H_neverTried;
```

```
{Check to see if they've pasted before: }
    if HR^^ HRnumPaste <> 0 then
      begin
        bestMethod := bestPaste;
        caret1 := 'paste text previously';
        case bestMethod of
          H_paste1menu:
            PasteMess1Menu;
          H_paste1key:
            PasteMess1Key;
          H_paste1palette:
            PasteMess1Palette;
          H_paste2menu:
            PasteMess2menu;
          H_paste2key:
            PasteMess2Key;
          H_paste2palette:
            PasteMess2Palette;
          otherwise

        end
      end

{ not, see if they've duplicated: }
    else if HR^^ HRnumDuplicate <> 0 then
      begin
        bestMethod := bestDuplicate12;
        if bestMethod <> H_neverTried then
          begin
            caret1 := 'duplicate text, except there is already text on the Clipboard.  Therefore, you will not select text to duplicate, you will just click
where you want the Clipboard text to go';
            case bestMethod of
              H_Duplicate1:
                PasteMess1Menu;
              H_Duplicate2:
                PasteMess2Menu;
              otherwise

            end
          end

{ not, see if they've made a style, size, or font change: }
        else if ((HR^^ HRnumStyle <> 0) or (HR^^ HRnumSize <> 0) or (HR^^ HRnumFont <> 0)) then
          begin
            bestMethod := bestText147;
            if bestMethod = H_Text4Palette then
              begin
                caret1 := 'change text style, except that you click where you want to paste, not where you want to type in the new style';
                PasteMess1Palette
              end
            else if bestMethod = H_Text4Menu then
              begin
                caret1 := 'change text style, except that you click where you want to paste, not where you want to type in the new style';
                PasteMess1Menu
              end
            else if bestMethod = H_Text7Menu then
              begin
                MustTutorPaste := TRUE;
                caret1 := 'change text style, except the instructions will ask you to click where you want to paste';
                PasteMess2Menu
              end
            else if bestMethod = H_Text7Palette then
              begin
                MustTutorPaste := TRUE;
                caret1 := 'change text style, except the instructions will ask you to click where you want to paste';
                PasteMess2Palette
              end
          end
      end;

{ none of the above: }
    if bestMethod = H_neverTried then
      begin
        reminderStr := '';
        GetIndString(firstStr, HelpCCSTR, reminder2);
        GetIndString(nextStr, HelpCCSTR, firstNext);
        GetIndString(resultStr, HelpFormatSTR, result3);
        caret1 := '';
        caret2 := 'First';
        if not FromMenu then
          caret3 := 'click on the Clip->Text rectangle in the palette'
        else
          caret3 := 'choose Paste from the Edit menu';
        MustTutorPaste := TRUE;
      end;

{Get the dialog box: }
    DLOGDone := FALSE;
    dPtr := GetNewDialog(PastingDLOG, nil, pointer(-1));
    SetPortDialog(dPtr);
    TextFont(geneva);
    TextSize(10);
    caret0 := 'paste';
    ParamText(caret0, caret1, caret2, caret3);

{Do the dialog text: }
    GetDItem(dPtr, reminderItem, itemType, textHandle, box);
    SetIText(textHandle, reminderStr);

    GetDItem(dPtr, firstItem, itemType, textHandle, box);
    SetIText(textHandle, firstStr);

    GetDItem(dPtr, nextItem, itemType, textHandle, box);
    SetIText(textHandle, nextStr);

    GetDItem(dPtr, resultItem, itemType, textHandle, box);
    SetIText(textHandle, resultStr);
```

```
    ShowWindow(dPtr);
    FrameDitem(dPtr, DoneButton);

{Wait for the Done button to be pressed! }
    while not DLOGDone do
      begin
        ModalDialog(nil, hit);
        if (hit = DoneButton) then
          DLOGDone := TRUE;
      end;
    DisposDialog(dPtr);
  end;
end.
```

```
{ Margaret Stone
  Sc.M. Project, Brown University

  This unit contains the code to provide help to the user when the user asks for help duplicating text. }

unit EditHelp4;

{ Interface Section }

interface

  uses
    EditorHelpInt, EditorUtilities, HelpBasis, HelpBasis2;

  procedure helpDuplicating (fromMenu: boolean);

{ Implementation Section }

implementation

  const
string indices in string list HelpFormatSTR:}
    select1 = 4;
    select2 = 5;
    select3 = 6;
    select4 = 7;
    select5 = 4;
    result3 = 13;

string indices in string list HelpCCSTR:}
    reminder1 = 1;
    reminder2 = 2;
    first1 = 3;
    firstNext = 4;
    result1 = 5;
    result2 = 6;

string indices in string list HelpPasteSTR:}
    first = 1;
    result = 2;
    dupResult1 = 3;
    dupResult2 = 4;
    dupResult3 = 5;
    dupResult4 = 6;
    dupResult5 = 7;

item numbers in dialog box}
    reminderItem = 3;
    firstItem = 4;
    nextItem = 5;
    resultItem = 6;

{ helpDuplicating gives the user help when they've chosen Help->Editing->duplicating text. }

  procedure helpDuplicating;          {fromMenu: boolean}
    const
      DoneButton = 1;
    var
      dPtr: DialogPtr;
      ret, itemType: integer;
      OLGADone: boolean;
      beaMethod: probeType;
      textHandle: Handle;
      box: Rect;
      reminderStr, firstStr, nextStr, resultStr, caret0, caret1, caret2, caret3, selectStr: Str255;
      box1: boolean;


    procedure DuplicateMess1;
    begin
      box1 := TRUE;
      GetIndString(resultStr, HelpPasteSTR, dupResult1);
      caret2 := 'Next';
      caret3 := 'choose Duplicate from the Edit menu';
      GetIndString(firstStr, HelpCCSTR, first1);
      GetIndString(nextStr, HelpCCSTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpCCSTR, reminder1);
    end;

    procedure DuplicateMess2;
    begin
      box1 := TRUE;
      reminderStr := '';
      GetIndString(firstStr, HelpCCSTR, reminder1);
      GetIndString(nextStr, HelpCCSTR, firstNext);
      resultStr := '';
      caret1 := '';
      caret2 := 'First';
      caret3 := 'choose Duplicate from the Edit menu';
    end;

    procedure DuplicateMess3;
      var
        thePref: boolean;
    begin
      thePref := menuPref;
      box1 := FALSE;
```

```
        if thePref then
           GetIndString(resultStr, HelpCCSTR, dupResult2)
        else
           GetIndString(resultStr, HelpPasteSTR, dupResult3);
        caret2 := 'Next';
        GetIndString(firstStr, HelpCCSTR, first1);
        GetIndString(nextStr, HelpCCSTR, firstNext);
        firstStr := concat(firstStr, selectStr);
        GetIndString(reminderStr, HelpCCSTR, reminder1);
        if thePref then
           caret3 := 'choose Copy from the Edit menu'
        else
           caret3 := 'click on the Text->Clip rectangle in the palette';
     end;

     procedure DuplicateMess4;
        var
           thePref: boolean;
     begin
        thePref := menuPref;
        boat := TRUE;
        if thePref then
           GetIndString(resultStr, HelpCCSTR, dupResult4)
        else
           GetIndString(resultStr, HelpPasteSTR, dupResult5);
        caret2 := 'Next';
        GetIndString(firstStr, HelpCCSTR, first1);
        GetIndString(nextStr, HelpCCSTR, firstNext);
        firstStr := concat(firstStr, selectStr);
        GetIndString(reminderStr, HelpCCSTR, reminder1);
        if thePref then
           caret3 := 'choose Copy from the Edit menu'
        else
           caret3 := 'click on the Text->Clip rectangle in the palette';
        MustTutorPaste := TRUE;
     end;


  begin
{create help duplicating message: }
     bestMethod := bestObject;
     case bestMethod of
        H_SelectMenu:
           GetIndString(selectStr, HelpFormatSTR, select3);
        H_SelectMouse:
           GetIndString(selectStr, HelpFormatSTR, select4);
        H_SelectKey:
           GetIndString(selectStr, HelpFormatSTR, select5);
        otherwise
           if itemMenu then
              GetIndString(selectStr, HelpFormatSTR, select1)
           else
              GetIndString(selectStr, HelpFormatSTR, select2)
     end;
     bestMethod := H_neverTried;

{Check to see if they've duplicated previously: }
     if HR^^ HRnumDuplicate <> 0 then
        begin
           bestMethod := bestDuplicate;
           caret1 := 'duplicate text previously';
           case bestMethod of
              H_Duplicate1:
                 DuplicateMess1;
              H_Duplicate2:
                 DuplicateMess2;
              H_Duplicate3Menu, H_Duplicate3Key, H_Duplicate3Palette:
                 DuplicateMess3;
              H_Duplicate4Menu, H_Duplicate4Key, H_Duplicate4Palette:
                 DuplicateMess4;
              otherwise

           end;
        end

{If not, see if they've moved: }
     else if HR^^ HRnumMove <> 0 then
        begin
           bestMethod := bestMove;
           caret1 := 'move text, except ';
           case bestMethod of
              H_Move1Menu, H_Move1Key, H_Move1Palette:
                 begin
                    DuplicateMess3;
                    caret1 := concat(caret1, 'you will choose Copy instead of Cut to place text on the Clipboard before you Paste');
                 end;
              H_Move2Menu, H_Move2Key, H_Move2Palette:
                 begin
                    DuplicateMess4;
                    caret1 := concat(caret1, 'you will choose Copy instead of Cut to place text on the Clipboard before you Paste');
                 end;
              H_Move3Menu, H_Move5Menu, H_Move3Key, H_Move5Key, H_Move3Palette, H_Move5Palette:
                 begin
                    DuplicateMess3;
                    caret1 := concat(caret1, 'you will not delete the selected text after you Copy it ( before you paste ) ');
                 end;
              H_Move4Menu, H_Move6Menu, H_Move4Key, H_Move6Key, H_Move4Palette, H_Move6Palette:
                 begin
                    DuplicateMess4;
                    caret1 := concat(caret1, 'you will not delete the selected text after you Copy it ( before you paste ) ');
                 end;
              H_Move7Menu, H_Move7Palette:
                 begin
                    DuplicateMess1;
                    caret1 := concat(caret1, 'the text will not be moved, instead there will be two copies of it in your document');
                 end;
              H_Move8Menu, H_Move8Palette:
                 begin
```

```
                DuplicateMess2:
                caret1 := concat(caret1, 'the text will not be moved. Instead there will be two copies of it in your document');
              end;
            otherwise

          end;
        end

{if not, see if they've copied: }
      else if HR^^.HRnumCopy <> 0 then
        begin
          bestMethod := bestCopy;
          caret1 := 'copy text, except after you copy the text, you will paste it into your document';
          case bestMethod of
            H_Copy1Menu, H_Copy1Key, H_Copy1Palette:
              DuplicateMess1;
            H_Copy2Menu, H_Copy2Key, H_Copy2Palette:
              DuplicateMess2;
            otherwise

          end;
        end

{if not, see if they've cut: }
      else if HR^^.HRnumCut <> 0 then
        begin
          bestMethod := bestCut;
          caret1 := 'cut text, except you will choose Copy instead of Cut, and you will then paste the text into your document';
          case bestMethod of
            H_Cut1Menu, H_Cut1Key, H_Cut1Palette:
              DuplicateMess1;
            H_Cut2Menu, H_Cut2Key, H_Cut2Palette:
              DuplicateMess2;
            otherwise

          end;
        end

{if not, see if they've replaced: }
      else if HR^^.HRnumReplace <> 0 then
        begin
          bestMethod := bestReplace78;
          caret1 := 'replace text, except ';
          if bestMethod <> H_neverTried then
            begin
              case bestMethod of
                H_Replace7Menu, H_Replace7Palette:
                  begin
                    DuplicateMess1;
                    caret1 := concat(caret1, 'after you have selected text, you will choose Duplicate rather than Replace from the Edit menu');
                  end;
                H_Replace8Menu, H_Replace8Palette:
                  begin
                    DuplicateMess2;
                    caret1 := concat(caret1, 'you will choose Duplicate rather than Replace, and the instructions will be different');
                  end;
                otherwise

              end;
            end

{if not, see if they've deleted: }
          else if HR^^.HRnumDelete <> 0 then
            begin
              bestMethod := bestDelete34;
              case bestMethod of
                H_Delete3Menu, H_Delete3Palette:
                  begin
                    DuplicateMess1;
                    caret1 := concat(caret1, 'the instructions window will appear after you choose Duplicate (rather than Delete), and you will
follow the instructions');
                  end;
                H_Delete4Menu, H_Delete4Palette:
                  begin
                    DuplicateMess2;
                    caret1 := concat(caret1, 'the instructions will ask you to select text to be duplicated, and then to click where you want the
duplicate to be');
                  end;
                otherwise

              end;
            end;
        end;


{Otherwise, if none of the above: }
      if bestMethod = H_neverTried then
        begin
          box1 := TRUE;
          reminderStr := '';
          firstStr := '';
          GetIndString(reminderStr, HelpCCSTR, reminder2);
          GetIndString(firstStr, HelpCCSTR, firstItem);
          caret1 := '';
          caret2 := 'First';
          caret3 := 'choose Duplicate from the Edit menu';
        end;

{Set the dialog box: }
      DLOGDone := FALSE;
      if box1 then
        dPtr := GetNewDialog(DuplicatingDLOG, nil, pointer(-1))
      else
        dPtr := GetNewDialog(Duplicating2DLOG, nil, pointer(-1));
      SetPort(dPtr);
      TextFont(geneva);
      TextSize(10);
      caret0 := 'duplicate';
```

```
        ParamText(caret0, caret1, caret2, caret3);

    {Do the dialog text }
        GetDItem(dPtr, reminderItem, itemType, textHandle, box);
        SetIText(textHandle, reminderStr);

        GetDItem(dPtr, firstItem, itemType, textHandle, box);
        SetIText(textHandle, firstStr);

        GetDItem(dPtr, nextItem, itemType, textHandle, box);
        SetIText(textHandle, nextStr);

        GetDItem(dPtr, resultItem, itemType, textHandle, box);
        SetIText(textHandle, resultStr);

        ShowWindow(dPtr);
        FrameDItem(dPtr, DoneButton);

    {Wait until Done is pressed: }
        while not DLOGDone do
          begin
            ModalDialog(nil, hit);
            if (hit = DoneButton) then
              DLOGDone := TRUE;
          end;
        DisposDialog(dPtr);
    end;
end.
```

```
{
{  Margaret Storey                                                                }
{  Sc.M.  Project, Brown University                                               }
{                                                                                 }
{  This unit contains the code to provide help to the user when the user asks for help cutting text.  }
{.................................................................................}

unit EditHelp6;

{.................................................................................}
{  Interface Section                                                              }
{.................................................................................}

interface

  uses
    EditorHelpIntf, EditorUtilities, HelpBasis;

  procedure helpCutting (fromMenu: boolean);

{.................................................................................}
{  Implementation Section                                                         }
{.................................................................................}

implementation

  const
{string indices in string list HelpFormatsSTR:}
    select1 = 4;
    select2 = 5;
    select3 = 6;
    select4 = 7;
    select5 = 8;
    result3 = 13;

{string indices in string list HelpCCSTR:}
    reminder1 = 1;
    reminder2 = 2;
    first1 = 3;
    firstNext = 4;
    result1 = 5;
    result2 = 6;

{item numbers in dialog box:}
    reminderItem = 3;
    firstItem = 4;
    nextItem = 5;
    resultItem = 6;

{.................................................................................}
{  helpCutting gives the user help when they've chosen Help->Editing->cutting text.  }
{.................................................................................}

  procedure helpCutting;        {(fromMenu:boolean)}
    const
      DoneButton = 1;
    var
      dPtr: DialogPtr;
      hit, itemType: integer;
      DLOGDone: boolean;
      bestMethod: profileType;
      textHandle: Handle;
      box: Rect;
      reminderStr, firstStr, nextStr, resultStr, caret0, caret1, caret2, caret3, selectStr: Str255;

    procedure CutMess1Menu;
    begin
      GetIndString(resultStr, HelpCCSTR, result1);
      caret2 := 'Next';
      caret3 := 'choose Cut from the Edit menu';
      GetIndString(firstStr, HelpCCSTR, first1);
      GetIndString(nextStr, HelpCCSTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpCCSTR, reminder1);
    end;

    procedure CutMess1Palette;
    begin
      GetIndString(resultStr, HelpCCSTR, result1);
      caret2 := 'Next';
      caret3 := 'click on the scissors in the palette';
      GetIndString(firstStr, HelpCCSTR, first1);
      GetIndString(nextStr, HelpCCSTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpCCSTR, reminder1);
    end;

    procedure CutMess1Key;
    begin
      GetIndString(resultStr, HelpCCSTR, result1);
      caret2 := 'Next';
      caret3 := 'press the command (clover) and x keys simultaneously on the keyboard';
      GetIndString(firstStr, HelpCCSTR, first1);
      GetIndString(nextStr, HelpCCSTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpCCSTR, reminder1);
    end;

    procedure CutMess2Menu;
    begin
      GetIndString(resultStr, HelpFormatsSTR, result3);
      caret2 := 'First';
      reminderStr := '';
      caret3 := 'choose Cut from the Edit menu';
      GetIndString(nextStr, HelpCCSTR, firstNext);
      GetIndString(firstStr, HelpCCSTR, reminder1);
    end;
```

```
    procedure CutMess2Palette;
    begin
      GetIndString(resultStr, HelpFormatSTR, result3);
      caret2 := 'First';
      caret3 := 'click on the scissors in the palette';
      reminderStr := '';
      GetIndString(nextStr, HelpCCSTR, firstNext);
      GetIndString(firstStr, HelpCCSTR, reminder1);
    end;

    procedure CutMess2Key;
    begin
      GetIndString(resultStr, HelpFormatSTR, result3);
      caret2 := 'First';
      reminderStr := '';
      caret3 := 'press the command (clover) and x keys simultaneously on the keyboard';
      GetIndString(nextStr, HelpCCSTR, firstNext);
      GetIndString(firstStr, HelpCCSTR, reminder1);
    end;

    procedure CutMess3;
    begin
      GetIndString(resultStr, HelpCCSTR, result1);
      caret2 := 'Next';
      GetIndString(firstStr, HelpCCSTR, first1);
      GetIndString(nextStr, HelpCCSTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpCCSTR, reminder1);
      if menuPref then
        caret3 := 'choose Cut from the Edit menu'
      else
        caret3 := 'click on the scissors in the palette';
    end;

  begin
{create help cutting message: }
    bestMethod := bestSelect;
    case bestMethod of
      H_SelectMenu:
        GetIndString(selectStr, HelpFormatSTR, select3);
      H_SelectMouse:
        GetIndString(selectStr, HelpFormatSTR, select4);
      H_SelectKey:
        GetIndString(selectStr, HelpFormatSTR, select5);
      otherwise
        if fromMenu then
          GetIndString(selectStr, HelpFormatSTR, select1)
        else
          GetIndString(selectStr, HelpFormatSTR, select2)
    end;
    bestMethod := H_neverThisX;

{Check to see if they've cut previously: }
    if HP^^.HPnumCut <> 0 then
      begin
        bestMethod := bestCut;
        caret1 := 'cut text previously';
        case bestMethod of
          H_cut1menu:
            CutMess1Menu;
          H_cut1key:
            CutMess1Key;
          H_cut1palette:
            CutMess1Palette;
          H_cut2menu:
            CutMess2menu;
          H_cut2key:
            CutMess2Key;
          H_cut2palette:
            CutMess2Palette;
          otherwise

        end
      end

{if not, see if they've copied: }
    else if HP^^.HPnumCopy <> 0 then
      begin
        bestMethod := bestCopy;
        caret1 := 'copy text, except you will ';
        case bestMethod of
          H_copy1menu:
            begin
              CutMess1Menu;
              caret1 := concat(caret1, 'choose Cut rather than Copy from the Edit menu');
            end;
          H_copy1key:
            begin
              CutMess1Key;
              caret1 := concat(caret1, 'press Command-x rather than Command-c');
            end;
          H_copy1palette:
            begin
              CutMess1Palette;
              caret1 := concat(caret1, 'click on the scissors icon rather than the Text->Clip rectangle');
            end;
          H_copy2menu:
            begin
              CutMess2Menu;
              caret1 := concat(caret1, 'choose Cut rather than Copy from the Edit menu');
            end;
          H_copy2key:
            begin
              CutMess2Key;
              caret1 := concat(caret1, 'press Command-x rather than Command-c');
            end;
          H_copy2palette:
            begin
```

```
                CutMess2Palette:
                caret1 := concat(caret1, 'click on the scissors icon rather than the Text->Clip rectangle');
              end;
            otherwise

          end
        end

{ not, see if they've deleted: }
      else if HR^^.HRnumDelete <> 0 then
        begin
          bestMethod := bestDelete34;
          if bestMethod <> H_neverTried then
            begin
              caret1 := 'delete text, except you will ';
              case bestMethod of
                H_delete3Menu:
                  begin
                    CutMess1Menu:
                    caret1 := concat(caret1, 'choose Cut rather than Delete from the Edit menu');
                  end;
                H_delete3Palette:
                  begin
                    CutMess1Palette:
                    caret1 := concat(caret1, 'click on the scissors icon rather than the delete icon');
                  end;
                H_delete4menu:
                  begin
                    CutMess2Menu:
                    caret1 := concat(caret1, 'choose Cut rather than Delete from the Edit menu');
                  end;
                H_delete4palette:
                  begin
                    CutMess2Palette:
                    caret1 := concat(caret1, 'click on the scissors icon rather than the delete icon');
                  end;
                otherwise

              end
            end

{ not, see if they've changed the style: }
          else if HR^^.HRnumStyle <> 0 then
            begin
              bestMethod := bestStyle;
              caret1 := 'change the lettering style, except you will ';
              case bestMethod of
                H_Text3Menu:
                  begin
                    CutMess1Menu:
                    caret1 := concat(caret1, 'choose Cut from the Edit menu, rather than a style from the Set Text Style menu');
                  end;
                H_Text3Palette:
                  begin
                    CutMess1Palette:
                    caret1 := concat(caret1, 'click on the scissors in the palette, rather than a style');
                  end;
                H_Text4Menu, H_Text5Menu:
                  begin
                    CutMess2Menu:
                    caret1 := concat(caret1, 'choose Cut from the Edit menu, rather than a style from the Set Text Style menu');
                  end;
                H_Text4Palette, H_Text5Palette:
                  begin
                    CutMess2Palette:
                    caret1 := concat(caret1, 'click on the scissors in the palette, rather than a style');
                  end;
              end
            end

{ not, see if they've changed the font: }
          else if HR^^.HRnumFont <> 0 then
            begin
              bestMethod := bestStyle;
              caret1 := 'change the typeface, except you will ';
              case bestMethod of
                H_Text2Menu:
                  begin
                    CutMess1Menu:
                    caret1 := concat(caret1, 'choose Cut from the Edit menu, rather than a typeface from the Set Text Style menu');
                  end;
                H_Text2Palette:
                  begin
                    CutMess1Palette:
                    caret1 := concat(caret1, 'click on the scissors in the palette, rather than a typeface');
                  end;
                H_Text5Menu, H_Text6Menu:
                  begin
                    CutMess2Menu:
                    caret1 := concat(caret1, 'choose Cut from the Edit menu, rather than a typeface from the Set Text Style menu');
                  end;
                H_Text5Palette, H_Text6Palette:
                  begin
                    CutMess2Palette:
                    caret1 := concat(caret1, 'click on the scissors in the palette, rather than a typeface');
                  end;
              end
            end

{ not, see if they've changed the size: }
          else if HR^^.HRnumSize <> 0 then
            begin
              bestMethod := bestStyle;
              caret1 := 'change the lettering size, except you will ';
              case bestMethod of
                H_Text1Menu:
                  begin
                    CutMess1Menu:
```

```
                        caret1 := concat(caret1, 'choose Cut from the Edit menu, rather than a size from the Set Text Style menu');
                      end;
                    H_Text1Palette:
                      begin
                        CutMess1Palette;
                        caret1 := concat(caret1, 'click on the scissors in the palette, rather than a size');
                      end;
                    H_Text7Menu, H_Text4Menu:
                      begin
                        CutMess2Menu;
                        caret1 := concat(caret1, 'choose Cut from the Edit menu, rather than a size from the Set Text Style menu');
                      end;
                    H_Text7Palette, H_Text4Palette:
                      begin
                        CutMess2Palette;
                        caret1 := concat(caret1, 'click on the scissors in the palette, rather than a size');
                      end;
                  end
              end

{if not, see if they've moved: }
            else if HR^.HRnumMove <> 0 then
              begin
                bestMethod := bestMove78;
                if bestMethod <> H_neverTried then
                  begin
                    caret1 := 'move text, except you will ';
                    case bestMethod of
                      H_move7menu:
                        begin
                          CutMess1Menu;
                          caret1 := concat(caret1, 'choose Cut rather than Move from the Edit menu');
                        end;
                      H_move7palette:
                        begin
                          CutMess1Palette;
                          caret1 := concat(caret1, 'click on the scissors in the palette, rather than the move icon');
                        end;
                      H_move4menu:
                        begin
                          CutMess2Menu;
                          caret1 := concat(caret1, 'choose Cut rather than Move from the Edit menu');
                        end;
                      H_move4palette:
                        begin
                          CutMess2Palette;
                          caret1 := concat(caret1, 'click on the scissors in the palette, rather than the move icon');
                        end;
                      otherwise

                    end
                  end

{if not, see if they've replaced: }
                else if HR^.HRnumReplace <> 0 then
                  begin
                    bestMethod := bestReplace81;
                    if bestMethod <> H_neverTried then
                      begin
                        caret1 := 'replace text, except you will ';
                        case bestMethod of
                          H_replace2, H_replace4:
                            begin
                              CutMess3;
                              caret1 := concat(caret1, 'choose Cut rather than typing or pressing delete');
                            end;
                          H_replace3menu, H_replace7menu:
                            begin
                              CutMess1Menu;
                              caret1 := concat(caret1, 'choose Cut rather than Replace from the Edit menu');
                            end;
                          H_replace3palette, H_replace7palette:
                            begin
                              CutMess1Palette;
                              caret1 := concat(caret1, 'click on the scissors in the palette, rather than the replace icon');
                            end;
                          H_replace8palette:
                            begin
                              CutMess2Palette;
                              caret1 := concat(caret1, 'click on the scissors in the palette, rather than the replace icon');
                            end;
                          H_replace8menu:
                            begin
                              CutMess2Menu;
                              caret1 := concat(caret1, 'choose Cut rather than Replace from the Edit menu');
                            end;
                          otherwise

                        end
                      end

{if not, see if they've duplicated: }
                else if HR^.HRnumDuplicate <> 0 then
                  begin
                    bestMethod := bestDuplicate;
                    caret1 := 'duplicate text, except you will choose Cut rather than Dupecate from the Edit menu';
                    case bestMethod of
                      H_duplicate1:
                        CutMess1Menu;
                      H_duplicate2:
                        CutMess2Menu;
                      otherwise

                    end
                  end
              end
          end;
```

```
{ Margaret Stone
  Sc.M. Project, Brown University

  This unit contains the code to provide help to the user when the user asks for help changing
  lettering style }

unit StyleHelp;

{ Interface Section }

interface

  uses
    EditorHelpers, EditorUtilities, HelpBasics;

  procedure helpStyle (fromMenu: boolean);

{ Implementation Section }

implementation

  const
{string indices in string list}
      reminder1 = 1;
      reminder2 = 2;
      first1 = 3;
      select1 = 4;
      select2 = 5;
      select3 = 6;
      select4 = 7;
      select5 = 8;
      firstNext = 9;
      first2 = 10;
      result1 = 11;
      result2 = 12;
      result3 = 13;

{item numbers in dialog box}
      reminderItem = 3;
      firstItem = 4;
      nextItem = 5;
      resultItem = 6;

  { Procedure helpStyle provides help when the user has chosen Changing any document's
    appearance -> Underlining/Boldfacing.
    It creates a help message based on actions they've taken previously in the word processor. }

  procedure helpStyle;        {(fromMenu:boolean)}
    const
      DoneButton = 1;
    var
      dPtr: DialogPtr;
      hit, itemType: integer;
      DLOGDone: boolean;
      bestMethod: profileType;
      itemHandle: Handle;
      box: Rect;
      reminderStr, firstStr, nextStr, resultStr, caret0, caret1, caret2, caret3, selectStr: Str255;

  begin
{create help style message}

      bestMethod := bestSelect;
      case bestMethod of
        H_SelectMenu:
          GetIndString(selectStr, HelpFormatSTR, select3);
        H_SelectMouse:
          GetIndString(selectStr, HelpFormatSTR, select4);
        H_SelectKey:
          GetIndString(selectStr, HelpFormatSTR, select5);
        otherwise
          if fromMenu then
            GetIndString(selectStr, HelpFormatSTR, select1)
          else
            GetIndString(selectStr, HelpFormatSTR, select2)
      end;
      bestMethod := H_neverTried;

{Check to see if the user's changed style before}
      if HFnumStyle <> 0 then
        begin
          bestMethod := bestStyle;
          caret1 := 'change the lettering style previously';
          case bestMethod of
            H_textSmenu:
              begin
                GetIndString(resultStr, HelpFormatSTR, result2);
                caret2 := 'Next';
                caret3 := 'Set Type Style menu';
                GetIndString(firstStr, HelpFormatSTR, first1);
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                firstStr := concat(firstStr, selectStr);
                GetIndString(reminderStr, HelpFormatSTR, reminder1)
              end;
            H_textSpalette:
              begin
                GetIndString(resultStr, HelpFormatSTR, result2);
                caret2 := 'Next';
                caret3 := 'palette';
```

```
Otherwise, if they've done none of the above }
    if beatMethod = H_neverTried then
      begin
        reminderStr := '';
        GetIndString(firstStr, HelpCCSTR, reminder2);
        GetIndString(nextStr, HelpCCSTR, firstNext);
        GetIndString(resultStr, HelpFormatSTR, result3);
        caret1 := '';
        caret2 := 'First';
        if not PromiseAll then
          caret3 := 'click on the answers in the calendar'
        else
          caret3 := 'choose Quit from the Edit menu';
      end;

Get the dialog text }
    DLOGDone := FALSE;
    dPtr := GetNewDialog(CustDlogDLOG, nil, pointer(-1));
    SetPort(dPtr);
    TextFont(geneva);
    TextSize(10);
    caret0 := 'cut';
    ParamText(caret0, caret1, caret2, caret3);

Do the dialog text }
    GetDItem(dPtr, reminderItem, itemType, textHandle, box);
    SetIText(textHandle, reminderStr);

    GetDItem(dPtr, firstItem, itemType, textHandle, box);
    SetIText(textHandle, firstStr);

    GetDItem(dPtr, nextItem, itemType, textHandle, box);
    SetIText(textHandle, nextStr);

    GetDItem(dPtr, resultItem, itemType, textHandle, box);
    SetIText(textHandle, resultStr);

    ShowWindow(dPtr);
    FrameOval(dPtr, DoneButton);

Wait until Done is pressed }
    while not DLOGDone do
      begin
        ModalDialog(nil, hit);
        if (hit = DoneButton) then
          DLOGDone := TRUE;
      end;
    DisposDialog(dPtr);
  end;
end.
```

```
            otherwise

        end;
      end

{If not, check to see if they've changed the size: }
    else if HR^.^.HRnumSizes <> 0 then
      begin
        bestMethod := bestSize;
        caret1 := 'change the lettering size, except you will choose a style, rather than a size';
        case bestMethod of
          H_text1menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result2);
              caret2 := 'Next';
              caret3 := 'Set Type Style menu';
              GetIndString(firstStr, HelpFormatSTR, first1);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
              GetIndString(reminderStr, HelpFormatSTR, remainder1);
            end;
          H_text1palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result2);
              caret2 := 'Next';
              caret3 := 'palette';
              GetIndString(firstStr, HelpFormatSTR, first1);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
              GetIndString(reminderStr, HelpFormatSTR, remainder1);
            end;
          H_text4menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result1);
              caret2 := 'Next';
              caret3 := 'Set Type Style menu';
              GetIndString(firstStr, HelpFormatSTR, first2);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(reminderStr, HelpFormatSTR, remainder1);
            end;
          H_text4palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result1);
              caret2 := 'Next';
              caret3 := 'palette';
              GetIndString(firstStr, HelpFormatSTR, first2);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(reminderStr, HelpFormatSTR, remainder1);
            end;
          H_text7menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result3);
              caret2 := 'First';
              reminderStr := '';
              caret3 := 'Set Type Style menu';
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(firstStr, HelpFormatSTR, remainder1);
            end;
          H_text7palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result3);
              caret2 := 'First';
              caret3 := 'palette';
              reminderStr := '';
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(firstStr, HelpFormatSTR, remainder1);
            end;
          otherwise

        end;
      end

{If none of the above, see if they've cut before: }
    else if HR^.^.HRnumCut <> 0 then
      begin
        bestMethod := bestCut;
        caret1 := 'cut text, except you will choose a style, rather than ';
        case bestMethod of
          H_cut1menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result2);
              caret2 := 'Next';
              caret3 := 'Set Type Style menu';
              GetIndString(firstStr, HelpFormatSTR, first1);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
              GetIndString(reminderStr, HelpFormatSTR, remainder1);
              caret1 := concat(caret1, 'Cut from the Edit menu');
            end;
          H_cut1key:
            begin
              GetIndString(resultStr, HelpFormatSTR, result2);
              caret2 := 'Next';
              caret3 := 'Set Type Style menu';
              GetIndString(firstStr, HelpFormatSTR, first1);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
              GetIndString(reminderStr, HelpFormatSTR, remainder1);
              caret1 := concat(caret1, 'typing command-x');
            end;
          H_cut1palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result2);
              caret2 := 'Next';
              caret3 := 'palette';
              GetIndString(firstStr, HelpFormatSTR, first1);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
```

```
                    GetIndString(firstStr, HelpFormatSTR, first1);
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    firstStr := concat(firstStr, selectStr);
                    GetIndString(remainderStr, HelpFormatSTR, remainder1);
                  end;
                H_text6menu:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result1);
                    caret2 := 'Next';
                    caret3 := 'Set Type Style menu';
                    GetIndString(firstStr, HelpFormatSTR, first2);
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(remainderStr, HelpFormatSTR, remainder1);
                  end;
                H_text6palette:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result1);
                    caret2 := 'Next';
                    caret3 := 'palette';
                    GetIndString(firstStr, HelpFormatSTR, first2);
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(remainderStr, HelpFormatSTR, remainder1);
                  end;
                H_text9menu:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result3);
                    caret2 := 'First';
                    remainderStr := '';
                    caret3 := 'Set Type Style menu';
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(firstStr, HelpFormatSTR, remainder1);
                  end;
                H_text9palette:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result3);
                    caret2 := 'First';
                    caret3 := 'palette';
                    remainderStr := '';
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(firstStr, HelpFormatSTR, remainder1);
                  end;
                otherwise

              end;
            end

Otherwise, check to see if they've changed the font: }
          else if HR^^ HFrumFont <> 0 then
            begin
              beatMethod := bestFont;
              caret1 := 'change the typeface, except you will choose a style rather than a font';
              case beatMethod of
                H_text2menu:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result2);
                    caret2 := 'Next';
                    caret3 := 'Set Type Style menu';
                    GetIndString(firstStr, HelpFormatSTR, first1);
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    firstStr := concat(firstStr, selectStr);
                    GetIndString(remainderStr, HelpFormatSTR, remainder1);
                  end;
                H_text2palette:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result2);
                    caret2 := 'Next';
                    caret3 := 'palette';
                    GetIndString(firstStr, HelpFormatSTR, first1);
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    firstStr := concat(firstStr, selectStr);
                    GetIndString(remainderStr, HelpFormatSTR, remainder1);
                  end;
                H_text5menu:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result1);
                    caret2 := 'Next';
                    caret3 := 'Set Type Style menu';
                    GetIndString(firstStr, HelpFormatSTR, first2);
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(remainderStr, HelpFormatSTR, remainder1);
                  end;
                H_text5palette:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result1);
                    caret2 := 'Next';
                    caret3 := 'palette';
                    GetIndString(firstStr, HelpFormatSTR, first2);
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(remainderStr, HelpFormatSTR, remainder1);
                  end;
                H_text8menu:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result3);
                    caret2 := 'First';
                    remainderStr := '';
                    caret3 := 'Set Type Style menu';
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(firstStr, HelpFormatSTR, remainder1);
                  end;
                H_text8palette:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result3);
                    caret2 := 'First';
                    caret3 := 'palette';
                    remainderStr := '';
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(firstStr, HelpFormatSTR, remainder1);
                  end;
```

```
                GetIndString(reminderStr, HelpFormatSTR, reminder1);
                caret1 := concat(caret1, 'the scissors from the palette');
              end;
            H_cut2menu:
              begin
                GetIndString(resultStr, HelpFormatSTR, result3);
                caret2 := 'First';
                reminderStr := '';
                caret3 := 'Set Type Style menu';
                mustTutorPerf := TRUE;
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                GetIndString(firstStr, HelpFormatSTR, reminder1);
                caret1 := concat(caret1, 'Cut from the Edit menu');
              end;
            H_cut2key:
              begin
                GetIndString(resultStr, HelpFormatSTR, result3);
                caret2 := 'First';
                reminderStr := '';
                caret3 := 'Set Type Style menu';
                mustTutorPerf := TRUE;
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                GetIndString(firstStr, HelpFormatSTR, reminder1);
                caret1 := concat(caret1, 'typing command-x');
              end;
            H_cut2palette:
              begin
                GetIndString(resultStr, HelpFormatSTR, result3);
                caret2 := 'First';
                caret3 := 'palette';
                reminderStr := '';
                mustTutorPerf := TRUE;
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                GetIndString(firstStr, HelpFormatSTR, reminder1);
                caret1 := concat(caret1, 'the scissors from the palette');
              end;
            otherwise

          end
        end

{ not see if they've copied: }
    else if HR^^ HRnumCopy <> 0 then
      begin
        bestMethod := bestCopy;
        caret1 := 'copy text, except you will choose a style, rather than ';
        case bestMethod of
          H_copy1menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result2);
              caret2 := 'Next';
              caret3 := 'Set Type Style menu';
              GetIndString(firstStr, HelpFormatSTR, first1);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
              GetIndString(reminderStr, HelpFormatSTR, reminder1);
              caret1 := concat(caret1, 'Copy from the Edit menu');
            end;
          H_copy1key:
            begin
              GetIndString(resultStr, HelpFormatSTR, result2);
              caret2 := 'Next';
              caret3 := 'Set Type Style menu';
              GetIndString(firstStr, HelpFormatSTR, first1);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
              GetIndString(reminderStr, HelpFormatSTR, reminder1);
              caret1 := concat(caret1, 'typing command-c');
            end;
          H_copy1palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result2);
              caret2 := 'Next';
              caret3 := 'palette';
              GetIndString(firstStr, HelpFormatSTR, first1);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
              GetIndString(reminderStr, HelpFormatSTR, reminder1);
              caret1 := concat(caret1, 'Text->Clip from the palette');
            end;
          H_copy2menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result3);
              caret2 := 'First';
              reminderStr := '';
              caret3 := 'Set Type Style menu';
              mustTutorPerf := TRUE;
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(firstStr, HelpFormatSTR, reminder1);
              caret1 := concat(caret1, 'Copy from the Edit menu');
            end;
          H_copy2key:
            begin
              GetIndString(resultStr, HelpFormatSTR, result3);
              caret2 := 'First';
              reminderStr := '';
              caret3 := 'Set Type Style menu';
              mustTutorPerf := TRUE;
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(firstStr, HelpFormatSTR, reminder1);
              caret1 := concat(caret1, 'typing command-c');
            end;
          H_copy2palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result3);
              caret2 := 'First';
              caret3 := 'palette';
              reminderStr := '';
```

```
                    mustTutorFeat := TRUE;
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(firstStr, HelpFormatSTR, reminder1);
                    caret1 := concat(caret1, 'Text->Clip from the palette');
                  end;
                otherwise

              end
            end

{If not, see if they've deleted: }
        else if HR^^.HRnumDelete <> 0 then
          begin
            bestMethod := bestDelete;
            caret1 := 'delete text, except you will choose a style ';
            case bestMethod of
              H_delete1:
                begin
                  GetIndString(resultStr, HelpFormatSTR, result1);
                  caret2 := 'Next';
                  if menuPref then
                    caret3 := 'Set Type Style menu'
                  else
                    caret3 := 'palette';
                  GetIndString(firstStr, HelpFormatSTR, first2);
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  GetIndString(reminderStr, HelpFormatSTR, reminder1);
                  caret1 := concat(caret1, 'after you move the insertion point, then type');
                end;
              H_delete2:
                begin
                  GetIndString(resultStr, HelpFormatSTR, result2);
                  caret2 := 'Next';
                  if menuPref then
                    caret3 := 'Set Type Style menu'
                  else
                    caret3 := 'palette';
                  GetIndString(firstStr, HelpFormatSTR, first1);
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  firstStr := concat(firstStr, selectStr);
                  GetIndString(reminderStr, HelpFormatSTR, reminder1);
                  caret1 := concat(caret1, 'after selecting text, rather than pressing delete');
                end;
              H_delete3menu:
                begin
                  GetIndString(resultStr, HelpFormatSTR, result1);
                  caret2 := 'Next';
                  caret3 := 'Set Type Style menu';
                  GetIndString(firstStr, HelpFormatSTR, first2);
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  GetIndString(reminderStr, HelpFormatSTR, reminder1);
                  caret1 := concat(caret1, 'after selecting text, rather than choose Delete from the Edit menu');
                end;
              H_delete3palette:
                begin
                  GetIndString(resultStr, HelpFormatSTR, result1);
                  caret2 := 'Next';
                  caret3 := 'palette';
                  GetIndString(firstStr, HelpFormatSTR, first2);
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  GetIndString(reminderStr, HelpFormatSTR, reminder1);
                  caret1 := concat(caret1, 'after selecting text, rather than clicking the Delete icon from the palette');
                end;
              H_delete4menu:
                begin
                  GetIndString(resultStr, HelpFormatSTR, result3);
                  caret2 := 'First';
                  reminderStr := '';
                  caret3 := 'Set Type Style menu';
                  mustTutorFeat := TRUE;
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  GetIndString(firstStr, HelpFormatSTR, reminder1);
                  caret1 := concat(caret1, 'rather than choose Delete from the Edit menu');
                end;
              H_delete4palette:
                begin
                  GetIndString(resultStr, HelpFormatSTR, result3);
                  caret2 := 'First';
                  caret3 := 'palette';
                  reminderStr := '';
                  mustTutorFeat := TRUE;
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  GetIndString(firstStr, HelpFormatSTR, reminder1);
                  caret1 := concat(caret1, 'rather than clicking the Delete icon from the palette');
                end;
              otherwise

            end
          end

{If not, see if they've replaced: }
        else if HR^^.HRnumReplace <> 0 then
          begin
            bestMethod := bestReplace;
            caret1 := 'replace text, except you will choose a Style, rather than';
            case bestMethod of
              H_replace1:
                begin
                  GetIndString(resultStr, HelpFormatSTR, result1);
                  caret2 := 'Next';
                  if menuPref then
                    caret3 := 'Set Type Style menu'
                  else
                    caret3 := 'palette';
                  GetIndString(firstStr, HelpFormatSTR, first2);
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  GetIndString(reminderStr, HelpFormatSTR, reminder1);
                  caret1 := concat(caret1, 'pressing the delete key');
```

```
  end;
H_replace2:
  begin
    GetIndString(resultStr, HelpFormatSTR, result2);
    caret2 := 'Next';
    if menuPref then
      caret3 := 'Set Type Style menu'
    else
      caret3 := 'palette';
    GetIndString(firstStr, HelpFormatSTR, first1);
    GetIndString(nextStr, HelpFormatSTR, firstNext);
    firstStr := concat(firstStr, selectStr);
    GetIndString(remainderStr, HelpFormatSTR, remainder1);
    caret1 := concat(caret1, 'pressing the delete key after selecting text');
  end;
H_replace3menu:
  begin
    GetIndString(resultStr, HelpFormatSTR, result2);
    caret2 := 'Next';
    caret3 := 'Set Type Style menu';
    GetIndString(firstStr, HelpFormatSTR, first1);
    GetIndString(nextStr, HelpFormatSTR, firstNext);
    firstStr := concat(firstStr, selectStr);
    GetIndString(remainderStr, HelpFormatSTR, remainder1);
    caret1 := concat(caret1, 'choosing Delete from the menu after selecting text');
  end;
H_replace3palette:
  begin
    GetIndString(resultStr, HelpFormatSTR, result2);
    caret2 := 'Next';
    caret3 := 'palette';
    GetIndString(firstStr, HelpFormatSTR, first1);
    GetIndString(nextStr, HelpFormatSTR, firstNext);
    firstStr := concat(firstStr, selectStr);
    GetIndString(remainderStr, HelpFormatSTR, remainder1);
    caret1 := concat(caret1, 'clicking the Delete icon after selecting text');
  end;
H_replace4menu:
  begin
    GetIndString(resultStr, HelpFormatSTR, result3);
    caret2 := 'First';
    reminderStr := '';
    caret3 := 'Set Type Style menu';
    mustTutorFont := TRUE;
    GetIndString(nextStr, HelpFormatSTR, firstNext);
    GetIndString(firstStr, HelpFormatSTR, remainder1);
    caret1 := concat(caret1, 'choosing Delete from the menu');
  end;
H_replace4palette:
  begin
    GetIndString(resultStr, HelpFormatSTR, result3);
    caret2 := 'First';
    caret3 := 'palette';
    reminderStr := '';
    mustTutorFont := TRUE;
    GetIndString(nextStr, HelpFormatSTR, firstNext);
    GetIndString(firstStr, HelpFormatSTR, remainder1);
    caret1 := concat(caret1, 'clicking the Delete icon');
  end;
H_replace6:
  begin
    GetIndString(resultStr, HelpFormatSTR, result2);
    caret2 := 'Next';
    if menuPref then
      caret3 := 'Set Type Style menu'
    else
      caret3 := 'palette';
    GetIndString(firstStr, HelpFormatSTR, first1);
    GetIndString(nextStr, HelpFormatSTR, firstNext);
    firstStr := concat(firstStr, selectStr);
    GetIndString(remainderStr, HelpFormatSTR, remainder1);
    caret1 := concat(caret1, 'typing after selecting text');
  end;
H_replace7palette:
  begin
    GetIndString(resultStr, HelpFormatSTR, result2);
    caret2 := 'Next';
    caret3 := 'palette';
    GetIndString(firstStr, HelpFormatSTR, first1);
    GetIndString(nextStr, HelpFormatSTR, firstNext);
    firstStr := concat(firstStr, selectStr);
    GetIndString(remainderStr, HelpFormatSTR, remainder1);
    caret1 := concat(caret1, 'clicking the Replace icon after selecting text');
  end;
H_replace7menu:
  begin
    GetIndString(resultStr, HelpFormatSTR, result2);
    caret2 := 'Next';
    caret3 := 'Set Type Style menu';
    GetIndString(firstStr, HelpFormatSTR, first1);
    GetIndString(nextStr, HelpFormatSTR, firstNext);
    firstStr := concat(firstStr, selectStr);
    GetIndString(remainderStr, HelpFormatSTR, remainder1);
    caret1 := concat(caret1, 'choosing Replace from the Edit menu after selecting text');
  end;
H_replace8palette:
  begin
    GetIndString(resultStr, HelpFormatSTR, result3);
    caret2 := 'First';
    caret3 := 'palette';
    reminderStr := '';
    mustTutorFont := TRUE;
    GetIndString(nextStr, HelpFormatSTR, firstNext);
    GetIndString(firstStr, HelpFormatSTR, remainder1);
    caret1 := concat(caret1, 'clicking the Replace icon');
  end;
H_replace8menu:
  begin
```

```
                    GetIndString(resultStr, HelpFormatSTR, result3);
                    caret2 := 'First';
                    reminderStr := '';
                    caret3 := 'Set Type Style menu';
                    mustTutorFont := TRUE;
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(firstStr, HelpFormatSTR, reminder1);
                    caret1 := concat(caret1, 'choosing Replace from the Edit menu');
                  end;
                otherwise

              end
            end

  {If not, see if they've moved: }
        else if HR^^.HRnumMove <> 0 then
          begin
            bestMethod := bestMove78;
            if bestMethod <> H_neverTried then
              begin
                caret1 := 'move text, except you will choose a style. ';
                case bestMethod of
                  H_move7menu:
                    begin
                      GetIndString(resultStr, HelpFormatSTR, result1);
                      caret2 := 'Next';
                      caret3 := 'Set Type Style menu';
                      GetIndString(firstStr, HelpFormatSTR, first2);
                      GetIndString(nextStr, HelpFormatSTR, firstNext);
                      GetIndString(reminderStr, HelpFormatSTR, reminder1);
                      caret1 := concat(caret1, 'rather than Move from the Edit menu');
                    end;
                  H_move7palette:
                    begin
                      GetIndString(resultStr, HelpFormatSTR, result1);
                      caret2 := 'Next';
                      caret3 := 'palette';
                      GetIndString(firstStr, HelpFormatSTR, first2);
                      GetIndString(nextStr, HelpFormatSTR, firstNext);
                      GetIndString(reminderStr, HelpFormatSTR, reminder1);
                      caret1 := concat(caret1, 'rather than clicking on the Move icon from the palette');
                    end;
                  H_move8menu:
                    begin
                      GetIndString(resultStr, HelpFormatSTR, result3);
                      caret2 := 'First';
                      reminderStr := '';
                      caret3 := 'Set Type Style menu';
                      mustTutorFont := TRUE;
                      GetIndString(nextStr, HelpFormatSTR, firstNext);
                      GetIndString(firstStr, HelpFormatSTR, reminder1);
                      caret1 := concat(caret1, 'rather than Move from the Edit menu');
                    end;
                  H_move8palette:
                    begin
                      GetIndString(resultStr, HelpFormatSTR, result3);
                      caret2 := 'First';
                      caret3 := 'palette';
                      reminderStr := '';
                      mustTutorFont := TRUE;
                      GetIndString(nextStr, HelpFormatSTR, firstNext);
                      GetIndString(firstStr, HelpFormatSTR, reminder1);
                      caret1 := concat(caret1, 'rather than clicking on the Move icon from the palette');
                    end;
                  otherwise

                end
              end

  { If not, see if they've duplicated: }
            else if HR^^.HRnumDuplicate <> 0 then
              begin
                bestMethod := bestDuplicate;
                caret1 := 'duplicate text, except you will choose a style rather than Duplicate from the Edit menu';
                case bestMethod of
                  H_duplicate1:
                    begin
                      GetIndString(resultStr, HelpFormatSTR, result1);
                      caret2 := 'Next';
                      GetIndString(firstStr, HelpFormatSTR, first2);
                      caret3 := 'Set Type Style menu';
                      GetIndString(nextStr, HelpFormatSTR, firstNext);
                      GetIndString(reminderStr, HelpFormatSTR, reminder1);
                    end;
                  H_duplicate2:
                    begin
                      GetIndString(resultStr, HelpFormatSTR, result3);
                      caret2 := 'First';
                      caret3 := 'Set Type Style menu';
                      mustTutorFont := TRUE;
                      GetIndString(nextStr, HelpFormatSTR, firstNext);
                      reminderStr := '';
                      GetIndString(firstStr, HelpFormatSTR, reminder1);
                    end;
                  otherwise

                end
              end;

  { If they haven't done any of the above, create a standard message: }
        if bestMethod = H_neverTried then
          begin
            reminderStr := '';
            GetIndString(firstStr, HelpFormatSTR, reminder2);
            GetIndString(nextStr, HelpFormatSTR, firstNext);
            GetIndString(resultStr, HelpFormatSTR, result3);
            caret1 := '';
```

```
            caret2 := 'Pref';
            caret3 := 'palette';
            mustTutorFont := TRUE
        end;

Get the dialog box: }
    DLOGDone := FALSE;
    dPtr := GetNewDialog(StyleDLOG, nil, pointer(-1));
    SetPort(dPtr);
    TextFont(geneva);
    TextSize(10);
    caret0 := 'style of lettering';
    ParamText(caret0, caret1, caret2, caret3);

Do the dialog text: }
    GetDItem(dPtr, remainderItem, ItemType, textHandle, box);
    SetText(textHandle, remainderStr);

    GetDItem(dPtr, firstItem, ItemType, textHandle, box);
    SetText(textHandle, firstStr);

    GetDItem(dPtr, nextItem, ItemType, textHandle, box);
    SetText(textHandle, nextStr);

    GetDItem(dPtr, resultItem, ItemType, textHandle, box);
    SetText(textHandle, resultStr);

    ShowWindow(dPtr);
    FrameDItem(dPtr, DoneButton);

Wait until the user presses the Done button: }
    while not DLOGDone do
        begin
            ModalDialog(nil, hit);
            if hit = DoneButton then
                DLOGDone := TRUE;
        end;
    DisposDialog(dPtr);
end;
end
```

```
{
Margaret Stone
Sc.M. Project, Brown University

This unit contains the code to provide help to the user when the user asks for help changing
lettering size.
}

unit SizeHelp;

{
Interface Section
}

interface
   uses
      EditorHeapInit, EditorUtilities, HelpBasis;

   procedure helpSize (fromMenu: boolean);

{
Implementation Section
}

implementation

   const
{string indices in string list}
      reminder1 = 1;
      reminder2 = 2;
      first1 = 3;
      select1 = 4;
      select2 = 5;
      select3 = 6;
      select4 = 7;
      select5 = 8;
      firstNext = 9;
      first2 = 10;
      result1 = 11;
      result2 = 12;
      result3 = 13;

{item numbers in dialog box:}
      reminderItem = 3;
      firstItem = 4;
      nextItem = 5;
      resultItem = 6;

{
  Procedure helpSize provides help when the user has chosen Changing my document's
  appearance -> Enlarging/Reducing lettering size.
}

   procedure helpSize;      {(fromMenu:boolean)}
      const
         DoneButton = 1;
      var
         dPtr: DialogPtr;
         int, itemType: integer;
         DLOGDone: boolean;
         bestMethod: prefsType;
         textHandle: Handle;
         box: Rect;
         reminderStr, firstStr, nextStr, resultStr, caret0, caret1, caret2, caret3, selectStr: Str255;

   begin
{create help size message:}

      bestMethod := bestSize;
      case bestMethod of
         H_SelectMenu:
            GetIndString(selectStr, HelpFormatSTR, select3);
         H_SelectMouse:
            GetIndString(selectStr, HelpFormatSTR, select4);
         H_SelectKey:
            GetIndString(selectStr, HelpFormatSTR, select5);
         otherwise
            if fromMenu then
               GetIndString(selectStr, HelpFormatSTR, select1)
            else
               GetIndString(selectStr, HelpFormatSTR, select2)
      end;
      bestMethod := H_neverTried;

{Check to see if the user's changed the size previously:}
      if HRec.HRNumSizes <> 0 then
         begin
            bestMethod := bestSize;
            caret1 := 'change the lettering size previously';
            case bestMethod of
               H_textMenu:
                  begin
                     GetIndString(resultStr, HelpFormatSTR, result2);
                     caret2 := 'Next';
                     caret3 := 'Set Type Style menu';
                     GetIndString(firstStr, HelpFormatSTR, first1);
                     GetIndString(nextStr, HelpFormatSTR, firstNext);
                     firstStr := concat(firstStr, selectStr);
                     GetIndString(reminderStr, HelpFormatSTR, reminder1);
                  end;
               H_textPalette:
                  begin
                     GetIndString(resultStr, HelpFormatSTR, result2);
                     caret2 := 'Next';
                     caret3 := 'palette';
                     GetIndString(firstStr, HelpFormatSTR, first1);
                     GetIndString(nextStr, HelpFormatSTR, firstNext);
                     firstStr := concat(firstStr, selectStr);
```

```
                    GetIndString(remainderStr, HelpFormatSTR, remainder1);
                  end;
                H_text4menu:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result1);
                    caret2 := 'Next';
                    caret3 := 'Set Type Style menu';
                    GetIndString(firstStr, HelpFormatSTR, first2);
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(remainderStr, HelpFormatSTR, remainder1);
                  end;
                H_text4palette:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result1);
                    caret2 := 'Next';
                    caret3 := 'palette';
                    GetIndString(firstStr, HelpFormatSTR, first2);
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(remainderStr, HelpFormatSTR, remainder1);
                  end;
                H_text7menu:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result3);
                    caret2 := 'First';
                    remainderStr := '';
                    caret3 := 'Set Type Style menu';
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(firstStr, HelpFormatSTR, remainder1);
                  end;
                H_text7palette:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result3);
                    caret2 := 'First';
                    caret3 := 'palette';
                    remainderStr := '';
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(firstStr, HelpFormatSTR, remainder1);
                  end;
                otherwise

              end;
            end

{If not, see if they've changed the style: }
          else if HiWord(HPrivateStyle) <> 0 then
            begin
              bestMethod := bestStyle;
              caret1 := 'change the lettering style, except you will choose a size rather than a style';
              case bestMethod of
                H_text3menu:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result2);
                    caret2 := 'Next';
                    caret3 := 'Set Type Style menu';
                    GetIndString(firstStr, HelpFormatSTR, first1);
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    firstStr := concat(firstStr, selectStr);
                    GetIndString(remainderStr, HelpFormatSTR, remainder1);
                  end;
                H_text3palette:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result2);
                    caret2 := 'Next';
                    caret3 := 'palette';
                    GetIndString(firstStr, HelpFormatSTR, first1);
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    firstStr := concat(firstStr, selectStr);
                    GetIndString(remainderStr, HelpFormatSTR, remainder1);
                  end;
                H_text6menu:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result1);
                    caret2 := 'Next';
                    caret3 := 'Set Type Style menu';
                    GetIndString(firstStr, HelpFormatSTR, first2);
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(remainderStr, HelpFormatSTR, remainder1);
                  end;
                H_text6palette:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result1);
                    caret2 := 'Next';
                    caret3 := 'palette';
                    GetIndString(firstStr, HelpFormatSTR, first2);
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(remainderStr, HelpFormatSTR, remainder1);
                  end;
                H_text8menu:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result3);
                    caret2 := 'First';
                    remainderStr := '';
                    caret3 := 'Set Type Style menu';
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(firstStr, HelpFormatSTR, remainder1);
                  end;
                H_text8palette:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result3);
                    caret2 := 'First';
                    caret3 := 'palette';
                    remainderStr := '';
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(firstStr, HelpFormatSTR, remainder1);
                  end;
                otherwise

              end;
```

```
        end

{ if not, see if they've changed the font: }
    else if HR^^.HRmenuFont <> 0 then
        begin
          bestMethod := bestFont;
          caret1 := 'change the typeface, except you will choose a size rather than a font';
          case bestMethod of
            H_font2menu:
              begin
                GetIndString(resultStr, HelpFormatSTR, result2);
                caret2 := 'Next';
                caret3 := 'Set Type Style menu';
                GetIndString(firstStr, HelpFormatSTR, first1);
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                firstStr := concat(firstStr, selectStr);
                GetIndString(reminderStr, HelpFormatSTR, remender1);
              end;
            H_font2palette:
              begin
                GetIndString(resultStr, HelpFormatSTR, result2);
                caret2 := 'Next';
                caret3 := 'palette';
                GetIndString(firstStr, HelpFormatSTR, first1);
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                firstStr := concat(firstStr, selectStr);
                GetIndString(reminderStr, HelpFormatSTR, remender1);
              end;
            H_font5menu:
              begin
                GetIndString(resultStr, HelpFormatSTR, result1);
                caret2 := 'Next';
                caret3 := 'Set Type Style menu';
                GetIndString(firstStr, HelpFormatSTR, first2);
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                GetIndString(reminderStr, HelpFormatSTR, remender1);
              end;
            H_font5palette:
              begin
                GetIndString(resultStr, HelpFormatSTR, result1);
                caret2 := 'Next';
                caret3 := 'palette';
                GetIndString(firstStr, HelpFormatSTR, first2);
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                GetIndString(reminderStr, HelpFormatSTR, remender1);
              end;
            H_font3menu:
              begin
                GetIndString(resultStr, HelpFormatSTR, result3);
                caret2 := 'First';
                reminderStr := '';
                caret3 := 'Set Type Style menu';
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                GetIndString(firstStr, HelpFormatSTR, remender1);
              end;
            H_font3palette:
              begin
                GetIndString(resultStr, HelpFormatSTR, result3);
                caret2 := 'First';
                caret3 := 'palette';
                reminderStr := '';
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                GetIndString(firstStr, HelpFormatSTR, remender1);
              end;
            otherwise

          end;
        end

{ if not, see if they've cut: }
    else if HR^^.HRmenuCut <> 0 then
        begin
          bestMethod := bestCut;
          caret1 := 'cut text, except you will choose a style, rather than ';
          case bestMethod of
            H_cut1menu:
              begin
                GetIndString(resultStr, HelpFormatSTR, result2);
                caret2 := 'Next';
                caret3 := 'Set Type Style menu';
                GetIndString(firstStr, HelpFormatSTR, first1);
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                firstStr := concat(firstStr, selectStr);
                GetIndString(reminderStr, HelpFormatSTR, remender1);
                caret1 := concat(caret1, 'Cut from the Edit menu');
              end;
            H_cut1key:
              begin
                GetIndString(resultStr, HelpFormatSTR, result2);
                caret2 := 'Next';
                caret3 := 'Set Type Style menu';
                GetIndString(firstStr, HelpFormatSTR, first1);
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                firstStr := concat(firstStr, selectStr);
                GetIndString(reminderStr, HelpFormatSTR, remender1);
                caret1 := concat(caret1, 'typing command-x');
              end;
            H_cut1palette:
              begin
                GetIndString(resultStr, HelpFormatSTR, result2);
                caret2 := 'Next';
                caret3 := 'palette';
                GetIndString(firstStr, HelpFormatSTR, first1);
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                firstStr := concat(firstStr, selectStr);
                GetIndString(reminderStr, HelpFormatSTR, remender1);
                caret1 := concat(caret1, 'the scissors from the palette');
              end;
```

```
            H_cut2menu:
               begin
                  GetIndString(resultStr, HelpFormatSTR, result3);
                  caret2 := 'First';
                  reminderStr := '';
                  caret3 := 'Set Type Style menu';
                  mustTutorFent := TRUE;
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  GetIndString(firstStr, HelpFormatSTR, reminder1);
                  caret1 := concat(caret1, 'Cut from the Edit menu');
               end;
            H_cut2key:
               begin
                  GetIndString(resultStr, HelpFormatSTR, result3);
                  caret2 := 'First';
                  reminderStr := '';
                  caret3 := 'Set Type Style menu';
                  mustTutorFent := TRUE;
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  GetIndString(firstStr, HelpFormatSTR, reminder1);
                  caret1 := concat(caret1, 'typing command-x');
               end;
            H_cut2palette:
               begin
                  GetIndString(resultStr, HelpFormatSTR, result3);
                  caret2 := 'First';
                  caret3 := 'palette';
                  reminderStr := '';
                  mustTutorFent := TRUE;
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  GetIndString(firstStr, HelpFormatSTR, reminder1);
                  caret1 := concat(caret1, 'the scissors from the palette');
               end;
            otherwise
               ;
         end
      end

{ if not, see if they've copied: }
   else if HR^^.HRnumCopy <> 0 then
      begin
         bestMethod := bestCopy;
         caret1 := 'copy text, except you will choose a style, rather than ';
         case bestMethod of
            H_copy1menu:
               begin
                  GetIndString(resultStr, HelpFormatSTR, result2);
                  caret2 := 'Next';
                  caret3 := 'Set Type Style menu';
                  GetIndString(firstStr, HelpFormatSTR, first1);
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  firstStr := concat(firstStr, selectStr);
                  GetIndString(reminderStr, HelpFormatSTR, reminder1);
                  caret1 := concat(caret1, 'Copy from the Edit menu');
               end;
            H_copy1key:
               begin
                  GetIndString(resultStr, HelpFormatSTR, result2);
                  caret2 := 'Next';
                  caret3 := 'Set Type Style menu';
                  GetIndString(firstStr, HelpFormatSTR, first1);
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  firstStr := concat(firstStr, selectStr);
                  GetIndString(reminderStr, HelpFormatSTR, reminder1);
                  caret1 := concat(caret1, 'typing command-c');
               end;
            H_copy1palette:
               begin
                  GetIndString(resultStr, HelpFormatSTR, result2);
                  caret2 := 'Next';
                  caret3 := 'palette';
                  GetIndString(firstStr, HelpFormatSTR, first1);
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  firstStr := concat(firstStr, selectStr);
                  GetIndString(reminderStr, HelpFormatSTR, reminder1);
                  caret1 := concat(caret1, 'Text->Clip from the palette');
               end;
            H_copy2menu:
               begin
                  GetIndString(resultStr, HelpFormatSTR, result3);
                  caret2 := 'First';
                  reminderStr := '';
                  caret3 := 'Set Type Style menu';
                  mustTutorFent := TRUE;
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  GetIndString(firstStr, HelpFormatSTR, reminder1);
                  caret1 := concat(caret1, 'Copy from the Edit menu');
               end;
            H_copy2key:
               begin
                  GetIndString(resultStr, HelpFormatSTR, result3);
                  caret2 := 'First';
                  reminderStr := '';
                  caret3 := 'Set Type Style menu';
                  mustTutorFent := TRUE;
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  GetIndString(firstStr, HelpFormatSTR, reminder1);
                  caret1 := concat(caret1, 'typing command-c');
               end;
            H_copy2palette:
               begin
                  GetIndString(resultStr, HelpFormatSTR, result3);
                  caret2 := 'First';
                  caret3 := 'palette';
                  reminderStr := '';
                  mustTutorFent := TRUE;
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  GetIndString(firstStr, HelpFormatSTR, reminder1);
```

```
                caret1 := concat(caret1, 'Text->Clip from the palette');
              end;
            otherwise
              ;
          end
        end

if not, see if they've deleted: }
    else if HR^^.HRnumDelete <> 0 then
      begin
        testMethod := testDelete;
        caret1 := 'delete text, except you will choose a style ';
        case testMethod of
          H_delete1:
            begin
              GetIndString(resultStr, HelpFormatSTR, result1);
              caret2 := 'Next';
              if menuPref then
                caret3 := 'Set Type Style menu'
              else
                caret3 := 'palette';
              GetIndString(firstStr, HelpFormatSTR, first2);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
              GetIndString(reminderStr, HelpFormatSTR, remainder1);
              caret1 := concat(caret1, 'after you move the insertion point, then type');
            end;
          H_delete2:
            begin
              GetIndString(resultStr, HelpFormatSTR, result2);
              caret2 := 'Next';
              if menuPref then
                caret3 := 'Set Type Style menu'
              else
                caret3 := 'palette';
              GetIndString(firstStr, HelpFormatSTR, first1);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
              GetIndString(reminderStr, HelpFormatSTR, remainder1);
              caret1 := concat(caret1, 'after selecting text, rather than pressing delete');
            end;
          H_delete3menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result1);
              caret2 := 'Next';
              caret3 := 'Set Type Style menu';
              GetIndString(firstStr, HelpFormatSTR, first2);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(reminderStr, HelpFormatSTR, remainder1);
              caret1 := concat(caret1, 'after selecting text, rather than choose Delete from the Edit menu');
            end;
          H_delete4palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result1);
              caret2 := 'Next';
              caret3 := 'palette';
              GetIndString(firstStr, HelpFormatSTR, first2);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(reminderStr, HelpFormatSTR, remainder1);
              caret1 := concat(caret1, 'after selecting text, rather than clicking the Delete icon from the palette');
            end;
          H_delete5menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result3);
              caret2 := 'First';
              remainderStr := '';
              caret3 := 'Set Type Style menu';
              mustTheirPref := TRUE;
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(firstStr, HelpFormatSTR, remainder1);
              caret1 := concat(caret1, 'rather than choose Delete from the Edit menu');
            end;
          H_delete6palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result3);
              caret2 := 'First';
              caret3 := 'palette';
              remainderStr := '';
              mustTheirPref := TRUE;
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(firstStr, HelpFormatSTR, remainder1);
              caret1 := concat(caret1, 'rather than clicking the Delete icon from the palette');
            end;
          otherwise
            ;
        end
      end

if not, see if they've replaced: }
    else if HR^^.HRnumReplace <> 0 then
      begin
        testMethod := testReplace;
        caret1 := 'replace text, except you will choose a Style, rather than';
        case testMethod of
          H_replace1:
            begin
              GetIndString(resultStr, HelpFormatSTR, result1);
              caret2 := 'Next';
              if menuPref then
                caret3 := 'Set Type Style menu'
              else
                caret3 := 'palette';
              GetIndString(firstStr, HelpFormatSTR, first2);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(reminderStr, HelpFormatSTR, remainder1);
              caret1 := concat(caret1, 'pressing the delete key');
            end;
          H_replace2:
```

```
    begin
      GetIndString(resultStr, HelpFormatSTR, result2);
      caret2 := 'Next';
      if menuPref then
        caret3 := 'Set Type Style menu'
      else
        caret3 := 'palette';
      GetIndString(firstStr, HelpFormatSTR, first1);
      GetIndString(nextStr, HelpFormatSTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpFormatSTR, reminder1);
      caret1 := concat(caret1, 'pressing the delete key after selecting text');
    end;
  H_replace3menu:
    begin
      GetIndString(resultStr, HelpFormatSTR, result2);
      caret2 := 'Next';
      caret3 := 'Set Type Style menu';
      GetIndString(firstStr, HelpFormatSTR, first1);
      GetIndString(nextStr, HelpFormatSTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpFormatSTR, reminder1);
      caret1 := concat(caret1, 'choosing Delete from the menu after selecting text');
    end;
  H_replace3palette:
    begin
      GetIndString(resultStr, HelpFormatSTR, result2);
      caret2 := 'Next';
      caret3 := 'palette';
      GetIndString(firstStr, HelpFormatSTR, first1);
      GetIndString(nextStr, HelpFormatSTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpFormatSTR, reminder1);
      caret1 := concat(caret1, 'clicking the Delete icon after selecting text');
    end;
  H_replace4menu:
    begin
      GetIndString(resultStr, HelpFormatSTR, result3);
      caret2 := 'First';
      reminderStr := '';
      caret3 := 'Set Type Style menu';
      mustTutorFont := TRUE;
      GetIndString(nextStr, HelpFormatSTR, firstNext);
      GetIndString(firstStr, HelpFormatSTR, reminder1);
      caret1 := concat(caret1, 'choosing Delete from the menu');
    end;
  H_replace4palette:
    begin
      GetIndString(resultStr, HelpFormatSTR, result3);
      caret2 := 'First';
      caret3 := 'palette';
      reminderStr := '';
      mustTutorFont := TRUE;
      GetIndString(nextStr, HelpFormatSTR, firstNext);
      GetIndString(firstStr, HelpFormatSTR, reminder1);
      caret1 := concat(caret1, 'clicking the Delete icon');
    end;
  H_replace6:
    begin
      GetIndString(resultStr, HelpFormatSTR, result2);
      caret2 := 'Next';
      if menuPref then
        caret3 := 'Set Type Style menu'
      else
        caret3 := 'palette';
      GetIndString(firstStr, HelpFormatSTR, first1);
      GetIndString(nextStr, HelpFormatSTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpFormatSTR, reminder1);
      caret1 := concat(caret1, 'typing after selecting text');
    end;
  H_replace7palette:
    begin
      GetIndString(resultStr, HelpFormatSTR, result2);
      caret2 := 'Next';
      caret3 := 'palette';
      GetIndString(firstStr, HelpFormatSTR, first1);
      GetIndString(nextStr, HelpFormatSTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpFormatSTR, reminder1);
      caret1 := concat(caret1, 'clicking the Replace icon after selecting text');
    end;
  H_replace7menu:
    begin
      GetIndString(resultStr, HelpFormatSTR, result2);
      caret2 := 'Next';
      caret3 := 'Set Type Style menu';
      GetIndString(firstStr, HelpFormatSTR, first1);
      GetIndString(nextStr, HelpFormatSTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpFormatSTR, reminder1);
      caret1 := concat(caret1, 'choosing Replace from the Edit menu after selecting text');
    end;
  H_replace8palette:
    begin
      GetIndString(resultStr, HelpFormatSTR, result3);
      caret2 := 'First';
      caret3 := 'palette';
      reminderStr := '';
      mustTutorFont := TRUE;
      GetIndString(nextStr, HelpFormatSTR, firstNext);
      GetIndString(firstStr, HelpFormatSTR, reminder1);
      caret1 := concat(caret1, 'clicking the Replace icon');
    end;
  H_replace8menu:
    begin
      GetIndString(resultStr, HelpFormatSTR, result3);
      caret2 := 'First';
```

```
                reminderStr := '';
                caret3 := 'Set Type Style menu';
                mustTutorFont := TRUE;
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                GetIndString(firstStr, HelpFormatSTR, reminder1);
                caret1 := concat(caret1, 'choosing Replace from the Edit menu');
              end;
            otherwise

          end
        end

{If not, see if they've moved: }
      else if HR^^.HFnumMove <> 0 then
        begin
          bestMethod := bestMove78;
          if bestMethod <> H_neverTried then
            begin
              caret1 := 'move text, except you will choose a style. ';
              case bestMethod of
                H_move7menu:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result1);
                    caret2 := 'Next';
                    caret3 := 'Set Type Style menu';
                    GetIndString(firstStr, HelpFormatSTR, first2);
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(reminderStr, HelpFormatSTR, reminder1);
                    caret1 := concat(caret1, 'rather than Move from the Edit menu');
                  end;
                H_move7palette:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result1);
                    caret2 := 'Next';
                    caret3 := 'palette';
                    GetIndString(firstStr, HelpFormatSTR, first2);
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(reminderStr, HelpFormatSTR, reminder1);
                    caret1 := concat(caret1, 'rather than clicking on the Move icon from the palette');
                  end;
                H_move8menu:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result3);
                    caret2 := 'First';
                    reminderStr := '';
                    caret3 := 'Set Type Style menu';
                    mustTutorFont := TRUE;
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(firstStr, HelpFormatSTR, reminder1);
                    caret1 := concat(caret1, 'rather than Move from the Edit menu');
                  end;
                H_move8palette:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result3);
                    caret2 := 'First';
                    caret3 := 'palette';
                    reminderStr := '';
                    mustTutorFont := TRUE;
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(firstStr, HelpFormatSTR, reminder1);
                    caret1 := concat(caret1, 'rather than clicking on the Move icon from the palette');
                  end;
                otherwise

              end
            end

{Finally, see if they've duplicated: }
          else if HR^^.HFnumDuplicate <> 0 then
            begin
              bestMethod := bestDuplicate;
              caret1 := 'duplicate text, except you will choose a style rather than Duplicate from the Edit menu';
              case bestMethod of
                H_duplicate1:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result1);
                    caret2 := 'Next';
                    GetIndString(firstStr, HelpFormatSTR, first2);
                    caret3 := 'Set Type Style menu';
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    GetIndString(reminderStr, HelpFormatSTR, reminder1);
                  end;
                H_duplicate2:
                  begin
                    GetIndString(resultStr, HelpFormatSTR, result3);
                    caret2 := 'First';
                    caret3 := 'Set Type Style menu';
                    mustTutorFont := TRUE;
                    GetIndString(nextStr, HelpFormatSTR, firstNext);
                    reminderStr := '';
                    GetIndString(firstStr, HelpFormatSTR, reminder1);
                  end;
                otherwise

              end
            end;

{if they haven't done any of the above, create a standard message: }
      if bestMethod = H_neverTried then
        begin
          reminderStr := '';
          GetIndString(firstStr, HelpFormatSTR, reminder2);
          GetIndString(nextStr, HelpFormatSTR, firstNext);
          GetIndString(resultStr, HelpFormatSTR, result3);
          caret1 := '';
          caret2 := 'First';
          caret3 := 'palette';
```

```
            mustTutorForm := TRUE
        end;

{put up the dialog box: }
    DLOGDone := FALSE;
    dPtr := GetNewDialog(SizeDLOG, nil, pointer(-1));
    SetPort(dPtr);
    TextFont(geneva);
    TextSize(10);
    caret0 := 'size of lettering';
    ParamText(caret0, caret1, caret2, caret3);

{Do the dialog text: }
    GetDItem(dPtr, remainderItem, itemType, textHandle, box);
    SetIText(textHandle, reminderStr);

    GetDItem(dPtr, firstItem, itemType, textHandle, box);
    SetIText(textHandle, firstStr);

    GetDItem(dPtr, nextItem, itemType, textHandle, box);
    SetIText(textHandle, nextStr);

    GetDItem(dPtr, resultItem, itemType, textHandle, box);
    SetIText(textHandle, resultStr);

    ShowWindow(dPtr);
    FrameDItem(dPtr, DoneButton);

{Wait for the user to click in the Done button: }
    while not DLOGDone do
        begin
            ModalDialog(nil, hit);
            if (hit = DoneButton) then
                DLOGDone := TRUE;
        end;
    DisposDialog(dPtr);
  end;
end.
```

```
{ Margaret Stone
  Sc.M. Project, Brown University

  This unit contains the code to provide help to the user when the user asks for help changing
  typeface (font).}

unit FontHelp;

{ Interface Section }

interface
   uses
      EditorHelpInt, EditorUtilities, HelpBasics;

   procedure helpFont (fromMenu: boolean);

{ Implementation Section }

implementation
   const
{string indices in string list: }
      reminder1 = 1;
      reminder2 = 2;
      first1 = 3;
      select1 = 4;
      select2 = 5;
      select3 = 6;
      select4 = 7;
      select5 = 8;
      firstNext = 9;
      first2 = 10;
      result1 = 11;
      result2 = 12;
      result3 = 13;

{item numbers in dialog box: }
      reminderItem = 3;
      firstItem = 4;
      nextItem = 5;
      resultItem = 6;

{ Procedure helpFont provides help when the user has chosen Changing my document's
  appearance -> Changing typeface. }

   procedure helpFont;          {(fromMenu:boolean)}
      const
         DensButton = 1;
      var
         dPtr: DialogPtr;
         hit, itemType: integer;
         DLOGDone: boolean;
         bestMethod: profileType;
         textHandle: Handle;
         box: Rect;
         reminderStr, firstStr, nextStr, resultStr, caret0, caret1, caret2, caret3, selectStr: Str255;

   begin
{create help font message;}

      bestMethod := bestSelect;
      case bestMethod of
         H_SelectMenu:
            GetIndString(selectStr, HelpFormatsTR, select3);
         H_SelectMouse:
            GetIndString(selectStr, HelpFormatsTR, select4);
         H_SelectKey:
            GetIndString(selectStr, HelpFormatsTR, select5);
         otherwise
            if fromMenu then
               GetIndString(selectStr, HelpFormatsTR, select1)
            else
               GetIndString(selectStr, HelpFormatsTR, select2)
      end;
      bestMethod := H_neverTried;

{Check to see if they've changed the font previously: }
      if HRA^.HRrunFont <> 0 then
         begin
            bestMethod := bestFont;
            caret1 := 'change the typeface previously';
            case bestMethod of
               H_text2menu:
                  begin
                     GetIndString(result3Str, HelpFormatsTR, result2);
                     caret2 := 'Next';
                     caret3 := 'Set Type Style menu';
                     GetIndString(firstStr, HelpFormatsTR, first1);
                     GetIndString(nextStr, HelpFormatsTR, firstNext);
                     firstStr := concat(firstStr, selectStr);
                     GetIndString(reminderStr, HelpFormatsTR, reminder1);
                  end;
               H_text2palette:
                  begin
                     GetIndString(result3Str, HelpFormatsTR, result2);
                     caret2 := 'Next';
                     caret3 := 'palette';
                     GetIndString(firstStr, HelpFormatsTR, first1);
                     GetIndString(nextStr, HelpFormatsTR, firstNext);
                     firstStr := concat(firstStr, selectStr);
                     GetIndString(reminderStr, HelpFormatsTR, reminder1);
```

```
            end;
          H_text5menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result1);
              caret2 := 'Next';
              caret3 := 'Set Type Style menu';
              GetIndString(firstStr, HelpFormatSTR, first2);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(remainderStr, HelpFormatSTR, reminder1);
            end;
          H_text5palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result1);
              caret2 := 'Next';
              caret3 := 'palette';
              GetIndString(firstStr, HelpFormatSTR, first2);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(remainderStr, HelpFormatSTR, reminder1);
            end;
          H_text6menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result3);
              caret2 := 'First';
              remainderStr := '';
              caret3 := 'Set Type Style menu';
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(firstStr, HelpFormatSTR, reminder1);
            end;
          H_text6palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result3);
              caret2 := 'First';
              caret3 := 'palette';
              remainderStr := '';
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(firstStr, HelpFormatSTR, reminder1);
            end;
          otherwise

        end;
      end

{If not, see if they've changed the size: }
    else if HR^^.HRnumSizes <> 0 then
      begin
        besMethod := besSize;
        caret1 := 'change the lettering size, except you will choose a typeface, rather than a size';
        case besMethod of
          H_text1menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result2);
              caret2 := 'Next';
              caret3 := 'Set Type Style menu';
              GetIndString(firstStr, HelpFormatSTR, first1);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
              GetIndString(remainderStr, HelpFormatSTR, reminder1);
            end;
          H_text1palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result2);
              caret2 := 'Next';
              caret3 := 'palette';
              GetIndString(firstStr, HelpFormatSTR, first1);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
              GetIndString(remainderStr, HelpFormatSTR, reminder1);
            end;
          H_text4menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result1);
              caret2 := 'Next';
              caret3 := 'Set Type Style menu';
              GetIndString(firstStr, HelpFormatSTR, first2);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(remainderStr, HelpFormatSTR, reminder1);
            end;
          H_text4palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result1);
              caret2 := 'Next';
              caret3 := 'palette';
              GetIndString(firstStr, HelpFormatSTR, first2);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(remainderStr, HelpFormatSTR, reminder1);
            end;
          H_text7menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result3);
              caret2 := 'First';
              remainderStr := '';
              caret3 := 'Set Type Style menu';
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(firstStr, HelpFormatSTR, reminder1);
            end;
          H_text7palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result3);
              caret2 := 'First';
              caret3 := 'palette';
              remainderStr := '';
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(firstStr, HelpFormatSTR, reminder1);
            end;
          otherwise

        end;
      end
```

```
{if not, see if they've changed the style: }
    else if HR^^.HRminusStyle <> 0 then
      begin
        bestMethod := bestStyle;
        caret1 := 'change the lettering style, except you will choose a typeface, rather than a style';
        case bestMethod of
          H_text3menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result2);
              caret2 := 'Next';
              caret3 := 'Set Type Style menu';
              GetIndString(firstStr, HelpFormatSTR, first1);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
              GetIndString(reminderStr, HelpFormatSTR, reminder1);
            end;
          H_text3palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result2);
              caret2 := 'Next';
              caret3 := 'palette';
              GetIndString(firstStr, HelpFormatSTR, first1);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
              GetIndString(reminderStr, HelpFormatSTR, reminder1);
            end;
          H_text4menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result1);
              caret2 := 'Next';
              caret3 := 'Set Type Style menu';
              GetIndString(firstStr, HelpFormatSTR, first2);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(reminderStr, HelpFormatSTR, reminder1);
            end;
          H_text4palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result1);
              caret2 := 'Next';
              caret3 := 'palette';
              GetIndString(firstStr, HelpFormatSTR, first2);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(reminderStr, HelpFormatSTR, reminder1);
            end;
          H_text5menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result3);
              caret2 := 'First';
              reminderStr := '';
              caret3 := 'Set Type Style menu';
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(firstStr, HelpFormatSTR, reminder1);
            end;
          H_text5palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result3);
              caret2 := 'First';
              caret3 := 'palette';
              reminderStr := '';
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              GetIndString(firstStr, HelpFormatSTR, reminder1);
            end;
          otherwise

        end;
      end

{if not, see if they've cut: }
    else if HR^^.HRminusCut <> 0 then
      begin
        bestMethod := bestCut;
        caret1 := 'cut text, except you will choose a style, rather than ';
        case bestMethod of
          H_cut1menu:
            begin
              GetIndString(resultStr, HelpFormatSTR, result2);
              caret2 := 'Next';
              caret3 := 'Set Type Style menu';
              GetIndString(firstStr, HelpFormatSTR, first1);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
              GetIndString(reminderStr, HelpFormatSTR, reminder1);
              caret1 := concat(caret1, 'Cut from the Edit menu');
            end;
          H_cut1key:
            begin
              GetIndString(resultStr, HelpFormatSTR, result2);
              caret2 := 'Next';
              caret3 := 'Set Type Style menu';
              GetIndString(firstStr, HelpFormatSTR, first1);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
              GetIndString(reminderStr, HelpFormatSTR, reminder1);
              caret1 := concat(caret1, 'typing command-x');
            end;
          H_cut1palette:
            begin
              GetIndString(resultStr, HelpFormatSTR, result2);
              caret2 := 'Next';
              caret3 := 'palette';
              GetIndString(firstStr, HelpFormatSTR, first1);
              GetIndString(nextStr, HelpFormatSTR, firstNext);
              firstStr := concat(firstStr, selectStr);
              GetIndString(reminderStr, HelpFormatSTR, reminder1);
              caret1 := concat(caret1, 'the scissors from the palette');
            end;
          H_cut2menu:
```

```
                begin
                  GetIndString(resultStr, HelpFormatSTR, result3);
                  caret2 := 'First';
                  remainderStr := '';
                  caret3 := 'Set Type Style menu';
                  mustTutorFont := TRUE;
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  GetIndString(firstStr, HelpFormatSTR, remainder1);
                  caret1 := concat(caret1, 'Cut from the Edit menu');
                end;
              H_cut2key:
                begin
                  GetIndString(resultStr, HelpFormatSTR, result3);
                  caret2 := 'First';
                  remainderStr := '';
                  caret3 := 'Set Type Style menu';
                  mustTutorFont := TRUE;
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  GetIndString(firstStr, HelpFormatSTR, remainder1);
                  caret1 := concat(caret1, 'typing command-x');
                end;
              H_cut2palette:
                begin
                  GetIndString(resultStr, HelpFormatSTR, result3);
                  caret2 := 'First';
                  caret3 := 'palette';
                  remainderStr := '';
                  mustTutorFont := TRUE;
                  GetIndString(nextStr, HelpFormatSTR, firstNext);
                  GetIndString(firstStr, HelpFormatSTR, remainder1);
                  caret1 := concat(caret1, 'the scissors from the palette');
                end;
              otherwise

            end
          end

' not see if they've copied: }
      else if HRes HFirstCopy <> 0 then
        begin
          lastSelHist := lastCopy;
          caret1 := 'copy text, except you will choose a style, rather than ';
          case lastSelHist of
            H_copy1menu:
              begin
                GetIndString(resultStr, HelpFormatSTR, result2);
                caret2 := 'Next';
                caret3 := 'Set Type Style menu';
                GetIndString(firstStr, HelpFormatSTR, first1);
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                firstStr := concat(firstStr, selectStr);
                GetIndString(remainderStr, HelpFormatSTR, remainder1);
                caret1 := concat(caret1, 'Copy from the Edit menu');
              end;
            H_copy1key:
              begin
                GetIndString(resultStr, HelpFormatSTR, result2);
                caret2 := 'Next';
                caret3 := 'Set Type Style menu';
                GetIndString(firstStr, HelpFormatSTR, first1);
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                firstStr := concat(firstStr, selectStr);
                GetIndString(remainderStr, HelpFormatSTR, remainder1)
                caret1 := concat(caret1, 'typing command-c')
              end;
            H_copy1palette:
              begin
                GetIndString(resultStr,, HelpFormatSTR, result2);
                caret2 := 'Next';
                caret3 := 'palette';
                GetIndString(firstStr, HelpFormatSTR, first1);
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                firstStr := concat(firstStr, selectStr);
                GetIndString(remainderStr, HelpFormatSTR, remainder1);
                caret1 := concat(caret1, 'Text->Clip from the palette');
              end;
            H_copy2menu:
              begin
                GetIndString(resultStr, HelpFormatSTR, result3);
                caret2 := 'First';
                remainderStr := '';
                caret3 := 'Set Type Style menu';
                mustTutorFont := TRUE;
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                GetIndString(firstStr, HelpFormatSTR, remainder1);
                caret1 := concat(caret1, 'Copy from the Edit menu');
              end;
            H_copy2key:
              begin
                GetIndString(resultStr, HelpFormatSTR, result3);
                caret2 := 'First';
                remainderStr := '';
                caret3 := 'Set Type Style menu';
                mustTutorFont := TRUE;
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                GetIndString(firstStr, HelpFormatSTR, remainder1);
                caret1 := concat(caret1, 'typing command-c');
              end;
            H_copy2palette:
              begin
                GetIndString(resultStr, HelpFormatSTR, result3);
                caret2 := 'First';
                caret3 := 'palette';
                remainderStr := '';
                mustTutorFont := TRUE;
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                GetIndString(firstStr, HelpFormatSTR, remainder1);
                caret1 := concat(caret1, 'Text->Clip from the palette');
```

```
                end;
              otherwise

            end
          end

{ not, see if they've deleted: }
      else if HR** HRnumDelete <> 0 then
        begin
          bestMethod := bestDelete;
          caret1 := 'delete text, except you will choose a style ';
          case bestMethod of
            H_delete1:
              begin
                GetIndString(resultStr, HelpFormatSTR, result1);
                caret2 := 'Next';
                if menuPref then
                  caret3 := 'Set Type Style menu'
                else
                  caret3 := 'palette';
                GetIndString(firstStr, HelpFormatSTR, first2);
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                firstStr := concat(firstStr, selectStr);
                GetIndString(reminderStr, HelpFormatSTR, remainder1);
                caret1 := concat(caret1, 'after you move the insertion point, then type');
              end;
            H_delete2:
              begin
                GetIndString(resultStr, HelpFormatSTR, result2);
                caret2 := 'Next';
                if menuPref then
                  caret3 := 'Set Type Style menu'
                else
                  caret3 := 'palette';
                GetIndString(firstStr, HelpFormatSTR, first1);
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                firstStr := concat(firstStr  selectStr);
                GetIndString(reminderStr, HelpFormatSTR, remainder1);
                caret1 := concat(caret1, 'after selecting text, rather than pressing delete');
              end;
            H_delete3menu:
              begin
                GetIndString(resultStr, HelpFormatSTR, result1);
                caret2 := 'Next';
                caret3 := 'Set Type Style menu';
                GetIndString(firstStr, HelpFormatSTR, first2);
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                GetIndString(reminderStr, HelpFormatSTR, remainder1);
                caret1 := concat(caret1, 'after selecting text, rather than choose Delete from the Edit menu');
              end;
            H_delete3palette:
              begin
                GetIndString(resultStr, HelpFormatSTR, result1);
                caret2 := 'Next';
                caret3 := 'palette';
                GetIndString(firstStr, HelpFormatSTR, first2);
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                GetIndString(reminderStr, HelpFormatSTR, remainder1);
                caret1 := concat(caret1, 'after selecting text, rather than clicking the Delete icon from the palette');
              end;
            H_delete4menu:
              begin
                GetIndString(resultStr, HelpFormatSTR  result3);
                caret2 := 'First';
                reminderStr := '';
                caret3 := 'Set Type Style menu'
                mustTutorFont := TRUE;
                GetIndString(nextStr, HelpFormatSTR  firstNext);
                GetIndString(firstStr, HelpFormatSTR, remainder1);
                caret1 := concat(caret1, 'rather than choose Delete from the Edit menu');
              end;
            H_delete4palette:
              begin
                GetIndString(resultStr, HelpFormatSTR, result3);
                caret2 := 'First';
                caret3 := 'palette';
                reminderStr := '';
                mustTutorFont := TRUE;
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                GetIndString(firstStr, HelpFormatSTR, remainder1);
                caret1 := concat(caret1, 'rather than clicking the Delete icon from the palette');
              end;
            otherwise

          end
        end

{ not, see if they've replaced: }
      else if HR** HRnumReplace <> 0 then
        begin
          bestMethod := bestReplace;
          caret1 := 'replace text, except you will choose a Style, rather than';
          case bestMethod of
            H_replace1:
              begin
                GetIndString(resultStr, HelpFormatSTR, result1);
                caret2 := 'Next';
                if menuPref then
                  caret3 := 'Set Type Style menu'
                else
                  caret3 := 'palette';
                GetIndString(firstStr, HelpFormatSTR, first2);
                GetIndString(nextStr, HelpFormatSTR, firstNext);
                GetIndString(reminderStr, HelpFormatSTR, remainder1);
                caret1 := concat(caret1, 'pressing the delete key');
              end;
            H_replace2:
              begin
```

```
      GetIndString(resultStr, HelpFormatsTR, result2);
      caret2 := 'Next';
      if menuPref then
        caret3 := 'Set Type Style menu'
      else
        caret3 := 'palette';
      GetIndString(firstStr, HelpFormatsTR, first1);
      GetIndString(nextStr, HelpFormatsTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpFormatsTR, reminder1);
      caret1 := concat(caret1, 'pressing the delete key after selecting text');
    end;
  H_replace3menu:
    begin
      GetIndString(resultStr, HelpFormatsTR, result2);
      caret2 := 'Next';
      caret3 := 'Set Type Style menu';
      GetIndString(firstStr, HelpFormatsTR, first1);
      GetIndString(nextStr, HelpFormatsTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpFormatsTR, reminder1);
      caret1 := concat(caret1, 'choosing Delete from the menu after selecting text');
    end;
  H_replace3palette:
    begin
      GetIndString(resultStr, HelpFormatsTR, result2);
      caret2 := 'Next';
      caret3 := 'palette';
      GetIndString(firstStr, HelpFormatsTR, first1);
      GetIndString(nextStr, HelpFormatsTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpFormatsTR, reminder1);
      caret1 := concat(caret1, 'clicking the Delete icon after selecting text');
    end;
  H_replace4menu:
    begin
      GetIndString(resultStr, HelpFormatsTR, result3);
      caret2 := 'First';
      reminderStr := '';
      caret3 := 'Set Type Style menu';
      mustTutorFeel := TRUE;
      GetIndString(nextStr, HelpFormatsTR, firstNext);
      GetIndString(firstStr, HelpFormatsTR, reminder1);
      caret1 := concat(caret1, 'choosing Delete from the menu');
    end;
  H_replace4palette:
    begin
      GetIndString(resultStr, HelpFormatsTR, result3);
      caret2 := 'First';
      caret3 := 'palette';
      reminderStr := '';
      mustTutorFeel := TRUE;
      GetIndString(nextStr, HelpFormatsTR, firstNext);
      GetIndString(firstStr, HelpFormatsTR, reminder1);
      caret1 := concat(caret1, 'clicking the Delete icon');
    end;
  H_replace6:
    begin
      GetIndString(resultStr, HelpFormatsTR, result2);
      caret2 := 'Next';
      if menuPref then
        caret3 := 'Set Type Style menu'
      else
        caret3 := 'palette';
      GetIndString(firstStr, HelpFormatsTR, first1);
      GetIndString(nextStr, HelpFormatsTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpFormatsTR, reminder1);
      caret1 := concat(caret1, 'typing after selecting text');
    end;
  H_replace7palette:
    begin
      GetIndString(resultStr, HelpFormatsTR, result2);
      caret2 := 'Next';
      caret3 := 'palette';
      GetIndString(firstStr, HelpFormatsTR, first1);
      GetIndString(nextStr, HelpFormatsTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpFormatsTR, reminder1);
      caret1 := concat(caret1, 'clicking the Replace icon after selecting text');
    end;
  H_replace7menu:
    begin
      GetIndString(resultStr, HelpFormatsTR, result2);
      caret2 := 'Next';
      caret3 := 'Set Type Style menu';
      GetIndString(firstStr, HelpFormatsTR, first1);
      GetIndString(nextStr, HelpFormatsTR, firstNext);
      firstStr := concat(firstStr, selectStr);
      GetIndString(reminderStr, HelpFormatsTR, reminder1);
      caret1 := concat(caret1, 'choosing Replace from the Edit menu after selecting text');
    end;
  H_replace4palette:
    begin
      GetIndString(resultStr, HelpFormatsTR, result3);
      caret2 := 'First';
      caret3 := 'palette';
      reminderStr := '';
      mustTutorFeel := TRUE;
      GetIndString(nextStr, HelpFormatsTR, firstNext);
      GetIndString(firstStr, HelpFormatsTR, reminder1);
      caret1 := concat(caret1, 'clicking the Replace icon');
    end;
  H_replace8menu:
    begin
      GetIndString(resultStr, HelpFormatsTR, result3);
      caret2 := 'First';
      reminderStr := '';
```

```
                    caret3 := 'Set Type Style menu';
                    mustTutorFent := TRUE;
                    GetIndString(nextStr, HelpFormatsTR, firstNext);
                    GetIndString(firstStr, HelpFormatsTR, reminder1);
                    caret1 := concat(caret1, 'choosing Replace from the Edit menu');
                  end;
                otherwise

              end
            end

{ If not, see if they've moved: }
        else if HR^^ HRnumMove <> 0 then
          begin
            bestMethod := bestMove78;
            if bestMethod <> H_neverTried then
              begin
                caret1 := 'move text, except you will choose a style, ';
                case bestMethod of
                  H_move78menu:
                    begin
                      GetIndString(resultStr, HelpFormatsTR, result1);
                      caret2 := 'Next';
                      caret3 := 'Set Type Style menu';
                      GetIndString(firstStr, HelpFormatsTR, first2);
                      GetIndString(nextStr, HelpFormatsTR, firstNext);
                      GetIndString(reminderStr, HelpFormatsTR, reminder1);
                      caret1 := concat(caret1, 'rather than Move from the Edit menu');
                    end;
                  H_more7palette:
                    begin
                      GetIndString(resultStr, HelpFormatsTR, result1);
                      caret2 := 'Next';
                      caret3 := 'palette';
                      GetIndString(firstStr, HelpFormatsTR, first2);
                      GetIndString(nextStr, HelpFormatsTR, firstNext);
                      GetIndString(reminderStr, HelpFormatsTR, reminder1);
                      caret1 := concat(caret1, 'rather than clicking on the Move icon from the palette');
                    end;
                  H_move8menu:
                    begin
                      GetIndString(resultStr, HelpFormatsTR, result3);
                      caret2 := 'First';
                      reminderStr := '';
                      caret3 := 'Set Type Style menu';
                      mustTutorFent := TRUE;
                      GetIndString(nextStr, HelpFormatsTR, firstNext);
                      GetIndString(firstStr, HelpFormatsTR, reminder1);
                      caret1 := concat(caret1, 'rather than Move from the Edit menu');
                    end;
                  H_move8palette:
                    begin
                      GetIndString(resultStr, HelpFormatsTR, result3);
                      caret2 := 'First';
                      caret3 := 'palette';
                      reminderStr := '';
                      mustTutorFent := TRUE;
                      GetIndString(nextStr, HelpFormatsTR, firstNext);
                      GetIndString(firstStr, HelpFormatsTR, reminder1);
                      caret1 := concat(caret1, 'rather than clicking on the Move icon from the palette');
                    end;
                  otherwise

                end
              end

{ Finally, see if they've duplicated: }
            else if HR^^ HRnumDuplicate <> 0 then
              begin
                bestMethod := bestDuplicate;
                caret1 := 'duplicate text, except you will choose a style rather than Duplicate from the Edit menu';
                case bestMethod of
                  H_duplicate1:
                    begin
                      GetIndString(resultStr, HelpFormatsTR, result1);
                      caret2 := 'Next';
                      GetIndString(firstStr, HelpFormatsTR, first2);
                      caret3 := 'Set Type Style menu';
                      GetIndString(nextStr, HelpFormatsTR, firstNext);
                      GetIndString(reminderStr, HelpFormatsTR, reminder1);
                    end;
                  H_duplicate2:
                    begin
                      GetIndString(resultStr, HelpFormatsTR, result3);
                      caret2 := 'First';
                      caret3 := 'Set Type Style menu';
                      mustTutorFent := TRUE;
                      GetIndString(nextStr, HelpFormatsTR, firstNext);
                      reminderStr := '';
                      GetIndString(firstStr, HelpFormatsTR, reminder1);
                    end;
                  otherwise

                end
              end;
          end;

{ If they haven't done any of the above, create a standard message: }
        if bestMethod = H_neverTried then
          begin
            reminderStr := '';
            GetIndString(firstStr, HelpFormatsTR, reminder2);
            GetIndString(nextStr, HelpFormatsTR, firstNext);
            GetIndString(resultStr, HelpFormatsTR, result3);
            caret1 := '';
            caret2 := 'First';
            caret3 := 'palette';
            mustTutorFent := TRUE
```

```
        end;

{Get the dialog box: }
    DLOGDone := FALSE;
    dPtr := GetNewDialog(TypefaceDLOG, nil, pointer(-1));
    SetPort(dPtr);
    TextFont(geneva);
    TextSize(10);
    caret0 := 'typeface';
    ParamText(caret0, caret1, caret2, caret3);

{Do the dialog text: }
    GetDItem(dPtr, reminderItem, itemType, textHandle, box);
    SetIText(textHandle, reminderStr);

    GetDItem(dPtr, firstItem, itemType, textHandle, box);
    SetIText(textHandle, firstStr);

    GetDItem(dPtr, nextItem, itemType, textHandle, box);
    SetIText(textHandle, nextStr);

    GetDItem(dPtr, resultItem, itemType, textHandle, box);
    SetIText(textHandle, resultStr);

    ShowWindow(dPtr);
    FrameDItem(dPtr, DoneButton);

{Wait for the user to press the Done button: }
    while not DLOGDone do
        begin
            ModalDialog(nil, hit);
            if (hit = DoneButton) then
                DLOGDone := TRUE;
        end;
    DisposDialog(dPtr);
  end;
end
```

```
{
  Margaret Stone
  Sc.M. Project, Brown University

  This unit contains the code to handle a press in the Help button or a choice from the Help menu, plus
  the code to handle intrusive help messages, and help selecting and help viewing.
}

unit HelpMessages;

{
  Interface Section
}

interface

  uses
    EditorGlobals, EditorHelper4, EditorUtilities, HelpBeeps, StyleHelp, SizeHelp, FontHelp, EditHelp, EditHelp2, EditHelp3, EditHelp4, EditHelp5;

  procedure DoHelpButton;
  procedure DoHelp (fromMenu: boolean; item: integer);
  procedure YouCantNotDoThat (chooseMenu: boolean; typeText: boolean; which: integer);
  procedure doChangeTo;

{
  Implemenation Section
}

implementation

  const
{string indices in string list HelpBalectSTR: }
    Sreminder1 = 1;
    Sreminder2 = 2;
    Smenu1 = 3;
    Smenu2 = 4;
    Smenu3 = 5;
    Smouse1 = 6;
    Smouse2 = 7;
    Skey1 = 8;
    Skey2 = 9;

{ string indices in string list HelpViewSTR: }
    Vreminder1 = 1;
    Vreminder2 = 2;
    Varrow1 = 3;
    Varrow2 = 4;
    Vmenu1 = 5;
    Vmenu2 = 6;
    Vmenu3 = 7;
    Velevator1 = 8;
    Velevator2 = 9;

{item numbers in dialog box: }
    reminderItem = 3;
    firstItem = 4;
    nextItem = 5;
    resultItem = 6;

{
  Procedure YouCantNotDoThat displays intrusive help messages when the user tries to select from a
  menu or choose from the palette while the instructions window is instructing him to do otherwise.
}

  procedure YouCantNotDoThat     (chooseMenu: boolean; typeText: boolean; which:integer);
    const
      DoneButton = 1;
    var
      dPtr: DialogPtr;
      hit, itemType: integer;
      DLOGDone: boolean;
      theItem: Handle;
      box: Rect;
  begin
    DLOGDone := FALSE;
    dPtr := GetNewDialog(UntilDLOG, nil, pointer(-1));

    if chooseMenu then
      begin
        case which of
          FileMenuID:
            ParamText('choose from the File Menu', '', '', '');
          EditMenuID:
            ParamText('choose from the Edit Menu', '', '', '');
          SelectMenuID:
            ParamText('choose from the Select Menu', '', '', '');
          ViewMenuID:
            ParamText('choose from the View Menu', '', '', '');
          TextMenuID:
            ParamText('choose from the Set Type Style Menu', '', '', '');
          otherwise
            ParamText('choose from that menu', '', '', '');
        end
      end
    else if typeText then
      ParamText('type text', '', '', '')
    else
      ParamText('choose from the Palette', '', '', '');

    ShowWindow(dPtr);
    SetPort(dPtr);
    FrameDItem(dPtr, DoneButton);

    while not DLOGDone do
      begin
        ModalDialog(nil, hit);
        if (hit = DoneButton) then
          DLOGDone := TRUE;
```

```
    end;
  DisposDialog(dPtr);
end;


{ Procedure doChangeTo instructs the user that the "Change to" in the palette is not a command.  }
{ The Change To in the palette will most likely be removed in a future version of the software.  }


procedure doChangeTo;
  const
    DoneButton = 1;
  var
    dPtr:  DialogPtr;
    hit, itemType:  integer;
    DLOGDone: boolean;
    theItem: Handle;
    box:  Rect;
begin
  DLOGDone := FALSE;
  dPtr := GetNewDialog(ChangeToDLOG, nil, pointer(-1));

  ShowWindow(dPtr);
  SetPort(dPtr);
  FrameDitem(dPtr, DoneButton);

  while not DLOGDone do
    begin
      ModalDialog(nil, hit);
      if (hit = DoneButton) then
        DLOGDone := TRUE;
    end;
  DisposDialog(dPtr);
end;


{ DoFormatHelp calls the procedure which will provide the proper help based on the user's request.  }


procedure DoFormatHelp (fromMenu: boolean);
  const
    StyleButton = 3;
    SizeButton = 4;
    FontButton = 5;
    OkayButton = 1;
    CancelButton = 2;
  var
    dPtr:  DialogPtr;
    hit, radio, itemType:  integer;
    DLOGDone: boolean;
    theItem: Handle;
    box:  Rect;
begin
{Show the format help dialog box: }
  DLOGDone := FALSE;
  dPtr := GetNewDialog(FormatDLOG, nil, pointer(-1));
  radio := 0;

  ShowWindow(dPtr);
  SetPort(dPtr);
  TextFont(geneva);
  TextSize(10);
  FrameOitem(dPtr, OkayButton);

{Wait for the user to make a format help choice: }
  while not DLOGDone do
    begin
      ModalDialog(nil, hit);
      if ((hit = StyleButton) or (hit = SizeButton) or (hit = FontButton)) and (hit <> radio) then
        begin
          GetDItem(dPtr, hit, itemType, theItem, box);
          SetCtlValue(ControlHandle(theItem), 1);
          if radio <> 0 then
            begin
              GetDItem(dPtr, radio, itemType, theItem, box);
              SetCtlValue(ControlHandle(theItem), 0);
            end;
          radio := hit;
          ShowWindow(dPtr)
        end;
      if ((hit = OkayButton) or (hit = CancelButton)) then
        DLOGDone := TRUE;
    end;

  DisposDialog(dPtr);

{Call the appropriate format help procedure: }
  if hit <> CancelButton then
    case radio of
      StyleButton:
        helpStyle(fromMenu);
      SizeButton:
        helpSize(fromMenu);
      FontButton:
        helpFont(fromMenu);
      otherwise

    end
end;


{ DoEditHelp calls the procedure which will provide the proper help based on the user's request.  }


procedure DoEditHelp (fromMenu: boolean);
  const
    SpellingButton = 3;
```

```
        RewordingButton = 4;
        DeletingButton = 5;
        SwitchingButton = 6;
        MovingButton = 7;
        ReplacingButton = 8;
        DuplicatingButton = 9;
        CuttingButton = 10;
        CopyingButton = 11;
        PastingButton = 12;
        OkayButton = 1;
        CancelButton = 2;
    var
        dPtr: DialogPtr;
        hit, radio, itemType: integer;
        DLOGDone: boolean;
        theItem: Handle;
        box: Rect;
  begin
{Create the edit help dialog box: }
    DLOGDone := FALSE;
    dPtr := GetNewDialog(EditDLOG, nil, pointer(-1));
    radio := 0;

    ShowWindow(dPtr);
    SetPort(dPtr);
    TextFont(geneva);
    TextSize(10);
    FrameDItem(dPtr, OkayButton);

{Wait for the user to make a choice: }
    while not DLOGDone do
      begin
        ModalDialog(nil, hit);
        if ((hit >= SpellingButton) and (hit <= PastingButton)) and (hit <> radio) then
          begin
            GetDItem(dPtr, hit, itemType, theItem, box);
            SetCtlValue(ControlHandle(theItem), 1);
            if radio <> 0 then
              begin
                GetDItem(dPtr, radio, itemType, theItem, box);
                SetCtlValue(ControlHandle(theItem), 0);
              end;
            radio := hit;
            ShowWindow(dPtr)
          end;
        if ((hit = OkayButton) or (hit = CancelButton)) then
          DLOGDone := TRUE;
      end;

    DisposDialog(dPtr);

{Provide the help the user requested: }
    if hit <> CancelButton then
      case radio of
        SpellingButton:
          helpReplacing(fromMenu, TRUE, FALSE);
        RewordingButton:
          helpReplacing(fromMenu, FALSE, TRUE);
        DeletingButton:
          helpDeleting(fromMenu);
        SwitchingButton:
          helpMoving(fromMenu, FALSE);
        MovingButton:
          helpMoving(fromMenu, TRUE);
        ReplacingButton:
          helpReplacing(fromMenu, FALSE, FALSE);
        DuplicatingButton:
          helpDuplicating(fromMenu);
        CuttingButton:
          helpCutting(fromMenu);
        PastingButton:
          helpPasting(fromMenu);
        CopyingButton:
          helpCopying(fromMenu);
        otherwise

      end;

  end;

  {-------------------------------------------------------------------------------}
  {DoSelectHelp provides the appropriate help when the user chooses Help -> Select.}
  {-------------------------------------------------------------------------------}

  procedure DoSelectHelp;
    const
      DoneButton = 1;
    var
      dPtr: DialogPtr;
      hit, itemType: integer;
      DLOGDone: boolean;
      helpSelected: profileType;
      textHandle: Handle;
      box: Rect;
      reminderStr, firstStr, nextStr, resultStr, caret0, selectStr: Str255;

    procedure SelectMenuHelp;
    begin
      GetIndString(firstStr, HelpSelectSTR, $menu1);
      GetIndString(nextStr, HelpSelectSTR, $menu2);
      GetIndString(resultStr, HelpSelectSTR, $menu3);
    end;

    procedure SelectMouseHelp;
    begin
      reminderStr := '';
      GetIndString(nextStr, HelpSelectSTR, $mouse1);
      GetIndString(resultStr, HelpSelectSTR, $mouse2);
```

```
    end;

    procedure SelectKeyMess;
    begin
      reminderStr := '';
      GetIndString(nextStr, HelpSelectSTR, SKey1);
      GetIndString(resultStr, HelpSelectSTR, SKey2);
    end;

  begin
create help selecting message: }
    bestMethod := H_neverTried;
    bestMethod := bestSelect;
    case bestMethod of
      H_SelectMenu:
        begin
          SelectMenuMess;
          GetIndString(reminderStr, HelpSelectSTR, Sreminder1);
          caret0 := 'by choosing an option from the Select menu';
        end;
      H_SelectMouse:
        begin
          SelectMouseMess;
          GetIndString(firstStr, HelpSelectSTR, Sreminder1);
          caret0 := 'by pressing on the mouse button while moving the mouse';
        end;
      H_SelectKey:
        begin
          SelectKeyMess;
          GetIndString(firstStr, HelpSelectSTR, Sreminder1);
          caret0 := 'by pressing the shift key and clicking to select the text between the insertion point and the shift key';
        end;
      H_neverTried:
        if (tutorPaste or mustTutorPaste or tutorFontSet or mustTutorFont) then
          begin
            SelectMenuMess;
            caret0 := 'the Select menu';
            GetIndString(reminderStr, HelpSelectSTR, Sreminder2);
          end
        else
          begin
            SelectMouseMess;
            caret0 := 'mouse';
            GetIndString(firstStr, HelpSelectSTR, Sreminder2);
          end;
      otherwise

    end;

{Get the dialog box: }
    DLOGDone := FALSE;
    dPtr := GetNewDialog(SelectDLOG, nil, pointer(-1));
    SetPort(dPtr);
    TextFont(geneva);
    TextSize(10);
    ParamText(caret0, '', '', '');

{Do the dialog text: }
    GetDItem(dPtr, reminderItem, itemType, textHandle, box);
    SetIText(textHandle, reminderStr);

    GetDItem(dPtr, firstItem, itemType, textHandle, box);
    SetIText(textHandle, firstStr);

    GetDItem(dPtr, nextItem, itemType, textHandle, box);
    SetIText(textHandle, nextStr);

    GetDItem(dPtr, resultItem, itemType, textHandle, box);
    SetIText(textHandle, resultStr);

    ShowWindow(dPtr);
    FrameDItem(dPtr, DoneButton);

{Wait for the user to press the Done button: }
    while not DLOGDone do
      begin
        ModalDialog(nil, hit);
        if (hit = DoneButton) then
          DLOGDone := TRUE;
      end;
    DisposDialog(dPtr);
  end;

{-----------------------------------------------------------------------------}
{DoViewHelp provides the appropriate help when the user chooses Help->seeing above/below screen }
{-----------------------------------------------------------------------------}

  procedure DoViewHelp;
    const
      DoneButton = 1;
    var
      dPtr: DialogPtr;
      hit, itemType: integer;
      DLOGDone: boolean;
      bestMethod: probeType;
      textHandle: Handle;
      box: Rect;
      reminderStr, firstStr, nextStr, resultStr, caret0, selectStr: Str255;

    procedure ViewMenu;
    begin
      GetIndString(firstStr, HelpViewSTR, Vmenu1);
      GetIndString(nextStr, HelpViewSTR, Vmenu2);
      GetIndString(resultStr, HelpViewSTR, Vmenu3);
    end;

    procedure ViewThumbDrag;
    begin
```

```
        reminderStr := '';
        GetIndString(nextStr, HelpViewSTR, Velevator1);
        GetIndString(resultStr, HelpViewSTR, Velevator2);
      end;

      procedure ViewScrollArrows;
      begin
        reminderStr := '';
        GetIndString(nextStr, HelpViewSTR, Varrow1);
        GetIndString(resultStr, HelpViewSTR, Varrow2);
      end;

  begin
{create help viewing message: }
      bestMethod := H_neverTried;
      bestMethod := bestView;
      case bestMethod of
        H_ScreenPMouse, H_ScreenNMouse, H_ClickUpArrow, H_ClickUpArrowRepeat, H_PressUpArrow, H_ClickDnArrow, H_ClickDnArrowRepeat,
H_PressDnArrow:
          begin
            ViewScrollArrows;
            GetIndString(firstStr, HelpViewSTR, Vreminder1);
            caret0 := 'the scroll bar, the rectangle on the side of the window with the arrows at the ends'
          end;
        H_DragThumb:
          begin
            ViewThumbDrag;
            GetIndString(firstStr, HelpViewSTR, Vreminder1);
            caret0 := 'the scroll bar, the rectangle on the side of the window with the arrows at the ends'
          end;
        H_ScreenPKey, H_ScreenNKey, H_ScreenPMenu, H_ScreenNMenu, H_LineNMenu, H_LineNKey, H_LinePMenu, H_LinePKey:
          begin
            ViewMenu;
            GetIndString(reminderStr, HelpViewSTR, Vreminder1);
            caret0 := 'the View menu'
          end;
        H_neverTried:
          if tutorPause or mustTutorPause or tutorFontSet or mustTutorFont then
            begin
              ViewMenu;
              caret0 := 'the View menu';
              GetIndString(reminderStr, HelpViewSTR, Vreminder2);
            end
          else
            begin
              ViewScrollArrows;
              caret0 := 'the scroll bar, the rectangle on the side of the window with the arrows at the ends';
              GetIndString(firstStr, HelpViewSTR, Vreminder2);
            end;
        otherwise

      end;

  {Get the dialog box: }
      DLOGDone := FALSE;
      dPtr := GetNewDialog(ViewDLOG, nil, pointer(-1));
      SetPort(dPtr);
      TextFont(geneva);
      TextSize(10);
      ParamText(caret0, '', '', '');

  {Do the dialog text: }
      GetDItem(dPtr, reminderItem, itemType, textHandle, box);
      SetIText(textHandle, reminderStr);

      GetDItem(dPtr, firstItem, itemType, textHandle, box);
      SetIText(textHandle, firstStr);

      GetDItem(dPtr, nextItem, itemType, textHandle, box);
      SetIText(textHandle, nextStr);

      GetDItem(dPtr, resultItem, itemType, textHandle, box);
      SetIText(textHandle, resultStr);

      ShowWindow(dPtr);
      FrameDItem(dPtr, DoneButton);

  {Wait for the user to press the Done button: }
      while not DLOGDone do
        begin
          ModalDialog(nil, hit);
          if (hit = DoneButton) then
            DLOGDone := TRUE;
        end;
      DisposDialog(dPtr);
    end;

  {-----------------------------------------------------------------------------}
  {DoHelp dispatches the user's help request to the appropriate handling procedure.}
  {-----------------------------------------------------------------------------}

  procedure DoHelp(      fromMenu: boolean; item: integer);
    const
      FormatButton = 3;
      EditButton = 4;
      SelectButton = 5;
      ViewButton = 6;

    begin
      if fromMenu then
        case item of
          FormatItem:
            DoFormatHelp(fromMenu);
          EditItem:
            DoEditHelp(fromMenu);
          SelectItem:
            DoSelectHelp;
          ViewItem:
```

```
            DoViewHelp;
          otherwise

        end
      else
        case item of
          FormatButton:
            DoFormatHelp(frameMenu);
          EditButton:
            DoEditHelp(frameMenu);
          SelectButton:
            DoSelectHelp;
          ViewButton:
            DoViewHelp;
          otherwise

        end;

      UnLoadSeg(@boolSelect);          {Segment 6}
      UnLoadSeg(@helpStyle);           {Segment 7}
      UnLoadSeg(@helpDeleting);        {Segment 8}
      UnLoadSeg(@helpFont);            {Segment 10}
      UnLoadSeg(@helpCopying);         {Segment 11}
  end;

{-----------------------------------------------------------------------------}
{ DoHelpButton handles the user pressing in the Help button.                  }
{-----------------------------------------------------------------------------}

  procedure DoHelpButton;
    const
      FormatButton = 3;
      EditButton = 4;
      SelectButton = 5;
      ViewButton = 6;
      OkayButton = 1;
      CancelButton = 2;
    var
      dPtr: DialogPtr;
      hit, radio, itemType: integer;
      DLOGDone: boolean;
      theItem: Handle;
      box: Rect;
  begin
{Get the help button dialog box.}
    DLOGDone := FALSE;
    dPtr := GetNewDialog(HelpButtonDLOG, nil, pointer(-1));
    radio := 0;

    ShowWindow(dPtr);
    SetPort(dPtr);
    TextFont(geneva);
    TextSize(10);
    FrameDItem(dPtr, OkayButton);

{Wait for the user to make a choice.}
    while not DLOGDone do
      begin
        ModalDialog(nil, hit);
        if ((hit = FormatButton) or (hit = EditButton) or (hit = SelectButton) or (hit = ViewButton)) and (hit <> radio) then
          begin
            GetDItem(dPtr, hit, itemType, theItem, box);
            SetCtlValue(ControlHandle(theItem), 1);
            if radio <> 0 then
              begin
                GetDItem(dPtr, radio, itemType, theItem, box);
                SetCtlValue(ControlHandle(theItem), 0);
              end;
            radio := hit;
            ShowWindow(dPtr)
          end;
        if ((hit = OkayButton) or (hit = CancelButton)) then
          DLOGDone := TRUE;
      end;

    DisposDialog(dPtr);

    if hit <> CancelButton then
      DoHelp(FALSE, radio);
  end;

end.
```

```
{
  Margaret Stone
  Sc.M. Project, Brown University

  This unit contains some of the compiler-like code to determine if the latest word-processor event
  will add a user strategy to the user help profile.
}

unit EditorHelp2;

{
  Interface Section
}

interface

  uses
    EditorGlobals, EditorHelpInt;

  procedure AssignValues (WPERTo: WPERHandle; WPERFrom: WPERPtr);
  procedure AssignValuesBack (WPERTo: WPERPtr; WPERFrom: WPERHandle);
  procedure RemoveIPSelect;
  procedure RemoveKeys;
  procedure HRHandlePaste;
  procedure HRHandleReplace;
  procedure HRHandleDuplicate;
  procedure HRHandleMove;

{
  Implementation Section
}

implementation

{
  AssignValues assigns the values from a handle to a pointer
}

  procedure AssignValues;  {(WPERTo: WPERHandle; WPERFrom: WPERPtr)}
  begin
    WPERTo^^.theEvent := WPERFrom^.theEvent;
    WPERTo^^.from := WPERFrom^.from;
    WPERTo^^.numChars := WPERFrom^.numChars;
    WPERTo^^.next := WPERFrom^.next
  end;

{
  AssignValuesBack assigns the values from a pointer to a handle.
}

  procedure AssignValuesBack;     {(WPERTo: WPERPtr; WPERFrom: WPERHandle)}
  begin
    WPERTo^.theEvent := WPERFrom^^.theEvent;
    WPERTo^.from := WPERFrom^^.from;
    WPERTo^.numChars := WPERFrom^^.numChars;
    WPERTo^.next := WPERFrom^^.next
  end;

{
  Procedure RemoveIPSelect handles situations in which there are a number of records of
  type MoveIP or Select before a PressOK or PressCancel which should be removed, but if
  simply important to the help profile, should be recorded there first.
  It should be noted that if Select was from a menu, there will be no PressOK or PressCancel.
  This procedure is called by HRHandleSizeSet, HRHandleFontSet, and HRHandleStyleSet
  HandleSelect
}

  procedure RemoveIPSelect;
    var
      temp, deletePtr: WPERPtr;
      done: boolean;
  begin
    temp := WPEvents^^.next;
    done := FALSE;
    while (temp <> nil) and (not done) do
      begin
        if temp^.theEvent = Select then
          begin
            HR^^.HRnumSelects := HR^^.HRnumSelects + 1;
            if temp^.from = menu then
              begin
                HR^^.HRSelectFromMenu := HR^^.HRSelectFromMenu + 1;
                done := TRUE
              end
            else if temp^.from = mouse then
              HR^^.HRSelectFromMouse := HR^^.HRSelectFromMouse + 1
            else if temp^.from = key then
              HR^^.HRSelectFromKey := HR^^.HRSelectFromKey + 1
            TOC^^.IPMoved := TRUE;
            TOC^^.EventSince := TRUE
          end
        else if temp^.theEvent = MoveIP then
          begin
            TOC^^.IPMoved := TRUE;
            TOC^^.EventSince := TRUE
          end
        else
          done := TRUE;
        deletePtr := temp;
        temp := temp^.next;
        DisposPtr(Ptr(deletePtr));
      end;
    if temp <> nil then
      begin
        deletePtr := temp;
        AssignValues(WPEvents, temp);
        DisposPtr(Ptr(deletePtr));
      end
```

```
    else
        begin
            DisposHandle(Handle(WPEvents));
            WPEvents := nil
        end
end;

{ Procedure RemoveKeys handles situations in which there are a number of records of type
  UserType or DeleteType before a PressOK or PressCancel which should be removed, but if
  singly important to the help profile, should be recorded there first.
  This procedure is called by HandleSelect }

procedure RemoveKeys;
    var
        temp, deletePtr: WPERPtr;
        done, already: boolean;
begin
    temp := WPEvents^^.next;
    done := FALSE;
    while (temp <> nil) and (not done) do
        begin
            already := FALSE;
            if temp^.theEvent = DeleteType then
                begin
                    if temp^.next <> nil then
                        if temp^.next^.theEvent = UserType then
                            begin
                                HR^^.HRnumReplace := HR^^.HRnumReplace + 1;
                                HR^^.HRReplace1 := HR^^.HRReplace1 + 1
                                if temp^.next^.next <> nil then
                                    begin
                                        already := TRUE;
                                        deletePtr := temp;
                                        temp := temp^.next^.next;
                                        DisposPtr(Ptr(deletePtr^.next));
                                        DisposPtr(Ptr(deletePtr));
                                    end
                            end

                end
            else if (temp^.theEvent <> UserType) and (temp^.theEvent <> DeleteType) then
                done := TRUE;
            if not already then
                begin
                    deletePtr := temp;
                    temp := temp^.next;
                    DisposPtr(Ptr(deletePtr));
                end;
        end;

    if temp <> nil then
        begin
            deletePtr := temp;
            AssignValues(WPEvents, temp);
            DisposPtr(Ptr(deletePtr));
        end
    else
        begin
            DisposHandle(Handle(WPEvents));
            WPEvents := nil
        end
end;

{ Procedure HRHandlePaste handles the first event = Paste. }

procedure HRHandlePaste;
    var
        temp, nextEvent, traverse, previous, deletePtr: WPERPtr;
        selectFlag: boolean;
begin
    TOC^^.EventSince := TRUE;
    temp := WPEvents^;
    { TutorPaste or MustTutorPaste then
        begin
            if WPEvents^^.next <> nil then
                begin
                    nextEvent := WPEvents^^.next;
                    if (nextEvent^.theEvent = MoveIP) or (nextEvent^.theEvent = Select) then
                        begin
                            TOC^^.IPMoved := TRUE;
                            selectFlag := FALSE;
                            traverse := nextEvent;
                            previous := traverse;
                            while (traverse^.next <> nil) and (not ((traverse^.theEvent = Select) and (traverse^.from = menu))) and ((traverse^.theEvent =
MoveIP) or (traverse^.theEvent = Select)) do
                                begin
                                    previous := traverse;
                                    traverse := traverse^.next;
                                end;
                            { (traverse^.theEvent = Select) and (traverse^.from = menu) then
                                selectFlag := TRUE;
                            { (traverse^.theEvent = PressOK) or selectFlag then
                                begin
                                    case TOC^^.method of
                                        CopyDelete:
                                            begin
                                                HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
                                                HR^^.HRnumDuplicate := HR^^.HRnumDuplicate + 1
                                                HR^^.HRnumMove := HR^^.HRnumMove + 1
                                                if temp^.from = menu then
                                                    begin
                                                        HR^^.HRPaste2Menu := HR^^.HRPaste2Menu + 1;
                                                        HR^^.HRDuplicate4Menu := HR^^.HRDuplicate4Menu + 1;
                                                        HR^^.HRMovedMenu := HR^^.HRMovedMenu + 1;
```

```
                    end
                  else if temp^.from = key then
                    begin
                      HR^^.HRPaste2Key := HR^^.HRPaste2Key + 1;
                      HR^^.HRDuplicate4Key := HR^^.HRDuplicate4Key + 1;
                      HR^^.HRMove6Key := HR^^.HRMove6Key + 1;
                    end
                  else if temp^.from = palette then
                    begin
                      HR^^.HRPaste2Palette := HR^^.HRPaste2Palette + 1;
                      HR^^.HRDuplicate4Palette := HR^^.HRDuplicate4Palette + 1;
                      HR^^.HRMove4Palette := HR^^.HRMove4Palette + 1;
                    end
                end;
              CopyDeleteType:
                begin
                  HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
                  HR^^.HRnumDuplicate := HR^^.HRnumDuplicate + 1;
                  HR^^.HRnumMove := HR^^.HRnumMove + 1;
                  if temp^.from = menu then
                    begin
                      HR^^.HRPaste2Menu := HR^^.HRPaste2Menu + 1;
                      HR^^.HRDuplicate4Menu := HR^^.HRDuplicate4Menu + 1;
                      HR^^.HRMove4Menu := HR^^.HRMove4Menu + 1;
                    end
                  else if temp^.from = key then
                    begin
                      HR^^.HRPaste2Key := HR^^.HRPaste2Key + 1;
                      HR^^.HRDuplicate4Key := HR^^.HRDuplicate4Key + 1;
                      HR^^.HRMove4Key := HR^^.HRMove4Key + 1;
                    end
                  else if temp^.from = palette then
                    begin
                      HR^^.HRPaste2Palette := HR^^.HRPaste2Palette + 1;
                      HR^^.HRDuplicate4Palette := HR^^.HRDuplicate4Palette + 1;
                      HR^^.HRMove4Palette := HR^^.HRMove4Palette + 1;
                    end
                end;
              CutTOC:
                begin
                  HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
                  HR^^.HRnumDuplicate := HR^^.HRnumDuplicate + 1;
                  HR^^.HRnumMove := HR^^.HRnumMove + 1;
                  if temp^.from = menu then
                    begin
                      HR^^.HRPaste2Menu := HR^^.HRPaste2Menu + 1;
                      HR^^.HRDuplicate4Menu := HR^^.HRDuplicate4Menu + 1;
                      HR^^.HRMove2Menu := HR^^.HRMove2Menu + 1;
                    end
                  else if temp^.from = key then
                    begin
                      HR^^.HRPaste2Key := HR^^.HRPaste2Key + 1;
                      HR^^.HRDuplicate4Key := HR^^.HRDuplicate4Key + 1;
                      HR^^.HRMove2Key := HR^^.HRMove2Key + 1;
                    end
                  else if temp^.from = palette then
                    begin
                      HR^^.HRPaste2Palette := HR^^.HRPaste2Palette + 1;
                      HR^^.HRDuplicate4Palette := HR^^.HRDuplicate4Palette + 1;
                      HR^^.HRMove2Palette := HR^^.HRMove2Palette + 1;
                    end
                end;
              noToc:
                begin
                  HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
                  if temp^.from = menu then
                    HR^^.HRPaste2Menu := HR^^.HRPaste2Menu + 1
                  else if temp^.from = key then
                    HR^^.HRPaste2Key := HR^^.HRPaste2Key + 1
                  else if temp^.from = palette then
                    HR^^.HRPaste2Palette := HR^^.HRPaste2Palette + 1
                end;
              otherwise
                begin
                  HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
                  HR^^.HRnumDuplicate := HR^^.HRnumDuplicate + 1;
                  if temp^.from = menu then
                    begin
                      HR^^.HRPaste2Menu := HR^^.HRPaste2Menu + 1;
                      HR^^.HRDuplicate4Menu := HR^^.HRDuplicate4Menu + 1;
                    end
                  else if temp^.from = key then
                    begin
                      HR^^.HRPaste2Key := HR^^.HRPaste2Key + 1;
                      HR^^.HRDuplicate4Key := HR^^.HRDuplicate4Key + 1;
                    end
                  else if temp^.from = palette then
                    begin
                      HR^^.HRPaste2Palette := HR^^.HRPaste2Palette + 1;
                      HR^^.HRDuplicate4Palette := HR^^.HRDuplicate4Palette + 1;
                    end
                end;
            end;
            RemoveIPSelect
          end
        else if traverse^.next <> nil then
          begin
            TOC^^.IPMoved := TRUE;
            TOC^^.EventSince := TRUE;
            RemoveIPSelect
          end
      end
  else if nextEvent^.theEvent = PressOK then
    begin
      case TOC^^.method of
        CopyDelete:
          begin
            if TOC^^.IPMoved then
```

```
            begin
               HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
               HR^^.HRnumDuplicate := HR^^.HRnumDuplicate + 1;
               HR^^.HRnumMove := HR^^.HRnumMove + 1;
            end
         else
            HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
         if TOC^^.IFMoved then
            begin
               if temp^.from = menu then
                  begin
                     HR^^.HRPaste2Menu := HR^^.HRPaste2Menu + 1;
                     HR^^.HRDuplicate4Menu := HR^^.HRDuplicate4Menu + 1;
                     HR^^.HRMove4Menu := HR^^.HRMove4Menu + 1;
                  end
               else if temp^.from = key then
                  begin
                     HR^^.HRPaste2Key := HR^^.HRPaste2Key + 1;
                     HR^^.HRDuplicate4Key := HR^^.HRDuplicate4Key + 1;
                     HR^^.HRMove4Key := HR^^.HRMove4Key + 1;
                  end
               else if temp^.from = palette then
                  begin
                     HR^^.HRPaste2Palette := HR^^.HRPaste2Palette + 1;
                     HR^^.HRDuplicate4Palette := HR^^.HRDuplicate4Palette + 1;
                     HR^^.HRMove4Palette := HR^^.HRMove4Palette + 1;
                  end
            end
         else if temp^.from = menu then
            HR^^.HRPaste2Menu := HR^^.HRPaste2Menu + 1
         else if temp^.from = key then
            HR^^.HRPaste2Key := HR^^.HRPaste2Key + 1
         else if temp^.from = palette then
            HR^^.HRPaste2Palette := HR^^.HRPaste2Palette + 1
      end;
   CopyDeleteType:
      begin
         if TOC^^.IFMoved then
            begin
               HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
               HR^^.HRnumDuplicate := HR^^.HRnumDuplicate + 1;
               HR^^.HRnumMove := HR^^.HRnumMove + 1
            end
         else
            HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
         if TOC^^.IFMoved then
            begin
               if temp^.from = menu then
                  begin
                     HR^^.HRPaste2Menu := HR^^.HRPaste2Menu + 1;
                     HR^^.HRDuplicate4Menu := HR^^.HRDuplicate4Menu + 1;
                     HR^^.HRMove4Menu := HR^^.HRMove4Menu + 1;
                  end
               else if temp^.from = key then
                  begin
                     HR^^.HRPaste2Key := HR^^.HRPaste2Key + 1;
                     HR^^.HRDuplicate4Key := HR^^.HRDuplicate4Key + 1;
                     HR^^.HRMove4Key := HR^^.HRMove4Key + 1;
                  end
               else if temp^.from = palette then
                  begin
                     HR^^.HRPaste2Palette := HR^^.HRPaste2Palette + 1;
                     HR^^.HRDuplicate4Palette := HR^^.HRDuplicate4Palette + 1;
                     HR^^.HRMove4Palette := HR^^.HRMove4Palette + 1;
                  end
            end
         else if temp^.from = menu then
            HR^^.HRPaste2Menu := HR^^.HRPaste2Menu + 1
         else if temp^.from = key then
            HR^^.HRPaste2Key := HR^^.HRPaste2Key + 1
         else if temp^.from = palette then
            HR^^.HRPaste2Palette := HR^^.HRPaste2Palette + 1
      end;
   CutTOC:
      begin
         if TOC^^.IFMoved then
            begin
               HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
               HR^^.HRnumDuplicate := HR^^.HRnumDuplicate + 1;
               HR^^.HRnumMove := HR^^.HRnumMove + 1;
            end
         else
            HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
         if TOC^^.IFMoved then
            begin
               if temp^.from = menu then
                  begin
                     HR^^.HRPaste2Menu := HR^^.HRPaste2Menu + 1;
                     HR^^.HRDuplicate4Menu := HR^^.HRDuplicate4Menu + 1;
                     HR^^.HRMove2Menu := HR^^.HRMove2Menu + 1;
                  end
               else if temp^.from = key then
                  begin
                     HR^^.HRPaste2Key := HR^^.HRPaste2Key + 1;
                     HR^^.HRDuplicate4Key := HR^^.HRDuplicate4Key + 1;
                     HR^^.HRMove2Key := HR^^.HRMove2Key + 1;
                  end
               else if temp^.from = palette then
                  begin
                     HR^^.HRPaste2Palette := HR^^.HRPaste2Palette + 1;
                     HR^^.HRDuplicate4Palette := HR^^.HRDuplicate4Palette + 1;
                     HR^^.HRMove2Palette := HR^^.HRMove2Palette + 1;
                  end
            end
         else if temp^.from = menu then
            HR^^.HRPaste2Menu := HR^^.HRPaste2Menu + 1
         else if temp^.from = key then
            HR^^.HRPaste2Key := HR^^.HRPaste2Key + 1
```

```
                    else if temp^.from = palette then
                      HR^^.HRPaste2Palette := HR^^.HRPaste2Palette + 1
                  end;
                noTec:
                  begin
                    HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
                    if temp^.from = menu then
                      HR^^.HRPaste2Menu := HR^^.HRPaste2Menu + 1
                    else if temp^.from = key then
                      HR^^.HRPaste2Key := HR^^.HRPaste2Key + 1
                    else if temp^.from = palette then
                      HR^^.HRPaste2Palette := HR^^.HRPaste2Palette + 1
                  end;
                otherwise
                  begin
                    HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
                    HR^^.HRnumDuplicate := HR^^.HRnumDuplicate + 1;
                    if temp^.from = menu then
                      begin
                        HR^^.HRPaste2Menu := HR^^.HRPaste2Menu + 1;
                        HR^^.HRDuplicate4Menu := HR^^.HRDuplicate4Menu + 1;
                      end
                    else if temp^.from = key then
                      begin
                        HR^^.HRPaste2Key := HR^^.HRPaste2Key + 1;
                        HR^^.HRDuplicate4Key := HR^^.HRDuplicate4Key + 1;
                      end
                    else if temp^.from = palette then
                      begin
                        HR^^.HRPaste2Palette := HR^^.HRPaste2Palette + 1;
                        HR^^.HRDuplicate4Palette := HR^^.HRDuplicate4Palette + 1;
                      end
                  end;
              end;
              RemoveInSelect
            end
          else if nextEvent^.theEvent = PressCancel then
            begin
              if nextEvent^.next = nil then
                begin
                  DisposeHandle(Handle(WPEvents));
                  WPEvents := nil
                end
              else
                begin
                  deletePtr := nextEvent^.next;
                  AssignValues(WPEvents, nextEvent^.next);
                  DisposPtr(Ptr(deletePtr));
                end;
              DisposPtr(Ptr(nextEvent));
            end
        end
    end
  else  {if not TutorPaste or MustTutorPaste}
    begin
      case TOC^^.method of
        noTOC:

        CurTOC:
          begin
            if TOC^^.IfMoved then
              begin
                HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
                HR^^.HRnumDuplicate := HR^^.HRnumDuplicate + 1;
                HR^^.HRnumMove := HR^^.HRnumMove + 1;
              end
            else
              HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
            if TOC^^.IfMoved then
              begin
                if temp^.from = menu then
                  begin
                    HR^^.HRPaste1Menu := HR^^.HRPaste1Menu + 1;
                    HR^^.HRDuplicate3Menu := HR^^.HRDuplicate3Menu + 1;
                    HR^^.HRMove1Menu := HR^^.HRMove1Menu + 1;
                  end
                else if temp^.from = key then
                  begin
                    HR^^.HRPaste1Key := HR^^.HRPaste1Key + 1;
                    HR^^.HRDuplicate3Key := HR^^.HRDuplicate3Key + 1;
                    HR^^.HRMove1Key := HR^^.HRMove1Key + 1;
                  end
                else if temp^.from = palette then
                  begin
                    HR^^.HRPaste1Palette := HR^^.HRPaste1Palette + 1;
                    HR^^.HRDuplicate3Palette := HR^^.HRDuplicate3Palette + 1;
                    HR^^.HRMove1Palette := HR^^.HRMove1Palette + 1;
                  end
              end
            else if temp^.from = menu then
              HR^^.HRPaste1Menu := HR^^.HRPaste1Menu + 1
            else if temp^.from = key then
              HR^^.HRPaste1Key := HR^^.HRPaste1Key + 1
            else if temp^.from = palette then
              HR^^.HRPaste1Palette := HR^^.HRPaste1Palette + 1
          end;
        CopyDelete:
          begin
            if TOC^^.IfMoved then
              begin
                HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
                HR^^.HRnumDuplicate := HR^^.HRnumDuplicate + 1;
                HR^^.HRnumMove := HR^^.HRnumMove + 1;
              end
            else
              HR^^.HRnumPaste := HR^^.HRnumPaste + 1;
            if TOC^^.IfMoved then
              begin
```

```
                    if temp^.from = menu then
                        begin
                            HR^^.HRPaste1Menu := HR^^.HRPaste1Menu + 1;
                            HR^^.HRDuplicate3Menu := HR^^.HRDuplicate3Menu + 1;
                            HR^^.HRMove6Menu := HR^^.HRMove6Menu + 1;
                        end
                    else if temp^.from = key then
                        begin
                            HR^^.HRPaste1Key := HR^^.HRPaste1Key + 1;
                            HR^^.HRDuplicate3Key := HR^^.HRDuplicate3Key + 1;
                            HR^^.HRMove6Key := HR^^.HRMove6Key + 1;
                        end
                    else if temp^.from = palette then
                        begin
                            HR^^.HRPaste1Palette := HR^^.HRPaste1Palette + 1;
                            HR^^.HRDuplicate3Palette := HR^^.HRDuplicate3Palette + 1;
                            HR^^.HRMove6Palette := HR^^.HRMove6Palette + 1;
                        end
                end
            else if temp^.from = menu then
                HR^^.HRPaste1Menu := HR^^.HRPaste1Menu + 1
            else if temp^.from = key then
                HR^^.HRPaste1Key := HR^^.HRPaste1Key + 1
            else if temp^.from = palette then
                HR^^.HRPaste1Palette := HR^^.HRPaste1Palette + 1
        end;

    CopyDeleteType:
        begin
            if TOC^^.IFMoved then
                begin
                    HR^^.HRnumPaste := HR^^.HRnumPaste + 1
                    HR^^.HRnumDuplicate := HR^^.HRnumDuplicate + 1
                    HR^^.HRnumMove := HR^^.HRnumMove + 1;
                end
            else
                HR^^.HRnumPaste := HR^^.HRnumPaste + 1
            if TOC^^.IFMoved then
                begin
                    if temp^.from = menu then
                        begin
                            HR^^.HRPaste1Menu := HR^^.HRPaste1Menu + 1;
                            HR^^.HRDuplicate3Menu := HR^^.HRDuplicate3Menu + 1;
                            HR^^.HRMove3Menu := HR^^.HRMove3Menu + 1;
                        end
                    else if temp^.from = key then
                        begin
                            HR^^.HRPaste1Key := HR^^.HRPaste1Key + 1
                            HR^^.HRDuplicate3Key := HR^^.HRDuplicate3Key + 1;
                            HR^^.HRMove3Key := HR^^.HRMove3Key + 1;
                        end
                    else if temp^.from = palette then
                        begin
                            HR^^.HRPaste1Palette := HR^^.HRPaste1Palette + 1;
                            HR^^.HRDuplicate3Palette := HR^^.HRDuplicate3Palette + 1;
                            HR^^.HRMove3Palette := HR^^.HRMove3Palette + 1;
                        end
                end
            else if temp^.from = menu then
                HR^^.HRPaste1Menu := HR^^.HRPaste1Menu + 1
            else if temp^.from = key then
                HR^^.HRPaste1Key := HR^^.HRPaste1Key + 1
            else if temp^.from = palette then
                HR^^.HRPaste1Palette := HR^^.HRPaste1Palette + 1
        end;
    otherwise
        begin
            HR^^.HRnumPaste := HR^^.HRnumPaste + 1
            if TOC^^.IFMoved then
                HR^^.HRnumDuplicate := HR^^.HRnumDuplicate + 1;
            if temp^.from = menu then
                begin
                    if TOC^^.IFMoved then
                        HR^^.HRDuplicate3Menu := HR^^.HRDuplicate3Menu + 1;
                    HR^^.HRPaste1Menu := HR^^.HRPaste1Menu + 1
                end
            else if temp^.from = key then
                begin
                    if TOC^^.IFMoved then
                        HR^^.HRDuplicate3Key := HR^^.HRDuplicate3Key + 1;
                    HR^^.HRPaste1Key := HR^^.HRPaste1Key + 1
                end
            else if temp^.from = palette then
                begin
                    if TOC^^.IFMoved then
                        HR^^.HRDuplicate3Palette := HR^^.HRDuplicate3Palette + 1;
                    HR^^.HRPaste1Palette := HR^^.HRPaste1Palette + 1
                end
        end;
    end;
    if temp^.next <> nil then
        begin
            deletePtr := temp^.next;
            AssignValues(WPEvents, deletePtr);
            DisposPtr(Ptr(deletePtr));
        end
    else
        begin
            DisposHandle(Handle(WPEvents));
            WPEvents := nil
        end;
end
end;
```

Procedure HRHandleReplace handles the first event = Replace.

```
procedure HRHandleReplace;
    var
        temp, nextEvent, nextNextEvent, traverse, previous, nextTraverse, nextPrevious, deletePtr: WPERPtr;
        selectFlag: boolean;
begin
    temp := WPEvents^;
    nextEvent := WPEvents^^.next;
    selectFlag := FALSE;
    TOC^^ EventSince := TRUE;
    if nextEvent <> nil then
        begin
            if (nextEvent^.theEvent = Select) or (nextEvent^.theEvent = MoveIP) then
                begin
                    traverse := nextEvent;
                    previous := traverse;
                    while (traverse^.next <> nil) and (not ((traverse^.theEvent = Select) and (traverse^.from = menu))) and ((traverse^.theEvent = MoveIP) or
traverse^.theEvent = Select)) do
                        begin
                            previous := traverse;
                            traverse := traverse^.next;
                        end;
                    if (traverse^.theEvent = Select) and (traverse^.from = menu) then
                        selectFlag := TRUE;
                    if (traverse^.theEvent = PressOK) or selectFlag then
                        if (previous^.theEvent = Select) or selectFlag then
                            begin
                                nextNextEvent := traverse^.next;
                                if nextNextEvent <> nil then
                                    begin
                                        if (nextNextEvent^.theEvent = UserType) or (nextNextEvent^.theEvent = DeleteType) then
                                            begin
                                                nextTraverse := nextNextEvent;
                                                nextPrevious := nextTraverse;
                                                while (nextTraverse^.next <> nil) do
                                                    begin
                                                        nextPrevious := nextTraverse;
                                                        nextTraverse := nextTraverse^.next;
                                                    end;
                                                if (nextTraverse^.theEvent = PressOK) then
                                                    begin
                                                        HR^^.HRReturnReplace := HR^^.HRReturnReplace + 1;
                                                        if temp^.from = menu then
                                                            HR^^.HRReplacedFromMenu := HR^^.HRReplacedFromMenu + 1
                                                        else
                                                            HR^^.HRReplacedFromPalette := HR^^.HRReplacedFromPalette + 1;
                                                        RemoveIPSelect;
                                                        RemoveKeys;
                                                    end
                                                else if (nextTraverse^.next <> nil) then    {nextTraverse <> PressOK }
                                                    begin
                                                        RemoveIPSelect;
                                                        RemoveKeys;
                                                    end
                                            end
                                        else    {nextNextEvent<>UserType and nextNextEvent<>DeleteType}
                                            begin
                                                RemoveIPSelect;
                                                if WPEvents <> nil then
                                                    begin
                                                        nextEvent := WPEvents^^.next;
                                                        if nextEvent = nil then
                                                            begin
                                                                DisposHandle(Handle(WPEvents));
                                                                WPEvents := nil;
                                                            end
                                                        else            {nextEvent <> nil}
                                                            begin
                                                                deletePtr := nextEvent;
                                                                AssignValues(WPEvents, nextEvent);
                                                                DisposePtr(Ptr(deletePtr));
                                                            end
                                                    end;
                                            end     {else nextNextEvent <> Select and nextNextEvent <> MoveIP}
                                    end         {nextNextEvent <> nil}
                            end             {previous event = select}
                        else            {previous event <> select, but there was a PressOK or SelectFlag}
                            begin
                                nextNextEvent := traverse^.next;
                                if nextNextEvent <> nil then
                                    begin
                                        if (nextNextEvent^.theEvent = PressOK) or (nextNextEvent^.theEvent = PressCancel) then
                                            begin
{Get rid of the whole thing}
                                                RemoveIPSelect;
                                                if WPEvents <> nil then
                                                    begin
                                                        nextEvent := WPEvents^^.next;
                                                        if nextEvent = nil then
                                                            begin
                                                                DisposHandle(Handle(WPEvents));
                                                                WPEvents := nil;
                                                            end
                                                        else            {nextEvent <> nil}
                                                            begin
                                                                deletePtr := nextEvent;
                                                                AssignValues(WPEvents, nextEvent);
                                                                DisposePtr(Ptr(deletePtr));
                                                            end
                                                    end;
                                            end
                                        else        {nextNextEvent <> PressOK or PressCancel }
                                            begin
{Keep going until PressOK or PressCancel, get rid of the whole thing}
                                                nextTraverse := nextNextEvent;
                                                nextPrevious := nextTraverse;
                                                while (nextTraverse^.next <> nil) and ((nextTraverse^.theEvent = UserType) or (nextTraverse^.theEvent =
DeleteType)) do
```

```
                                    begin
                                        nextPrevious := nextTraverse;
                                        nextTraverse := nextTraverse^.next;
                                    end;
                                if (nextTraverse^.theEvent = PressOK) or (nextTraverse^.theEvent = PressCancel) then
                                    begin
                                        RemoveIPSelect;
                                        RemoveKeys;
                                    end
                                end
                            end
                        end
                    else if (traverse^.next <> nil) then            {traverse <> PressOK}
                        RemoveIPSelect
                end
            else if nextEvent^.theEvent = PressCancel then
                begin
{Get rid of Duplicate and PressCancel}
                if nextEvent^.next <> nil then
                    begin
                        deletePtr := nextEvent^.next;
                        AssignValues(WPEvents, nextEvent^.next);
                        DisposPtr(Ptr(deletePtr));
                    end
                else
                    begin
                        DisposHandle(Handle(WPEvents));
                        WPEvents := nil
                    end;
                DisposPtr(Ptr(nextEvent));
                end
            else if nextEvent^.theEvent = PressOK then
                begin
{Keep going until another pressOK or pressCancel, and delete everything but selects}
                traverse := nextEvent^.next;
                if traverse <> nil then
                    if (traverse^.theEvent = UserType) or (traverse^.theEvent = DeleteType) then
                        begin
{Get rid of Replace,PressOK,UserType/DeleteType,PressOK or PressCancel}
                        previous := traverse;
                        while (traverse^.next <> nil) and ((traverse^.theEvent = UserType) or (traverse^.theEvent = DeleteType)) do
                            begin
                                previous := traverse;
                                traverse := traverse^.next
                            end;
                        if (traverse^.theEvent = PressOK) or (traverse^.theEvent = PressCancel) then
                            begin
                                deletePtr := nextEvent;
                                AssignValues(WPEvents, nextEvent);
                                DisposPtr(Ptr(deletePtr));
                                RemoveKeys;
                            end
                        end
                    else        {traverse^.theEvent<>DeleteType or UserType}
                        begin
{Get rid of Replace,PressOK, PressOK or PressCancel}
                        if traverse^.next <> nil then
                            begin
                                deletePtr := traverse^.next;
                                AssignValues(WPEvents, traverse^.next);
                                DisposPtr(Ptr(deletePtr));
                            end
                        else
                            begin
                                DisposHandle(Handle(WPEvents));
                                WPEvents := nil
                            end;
                        DisposPtr(Ptr(traverse));
                        DisposPtr(Ptr(nextEvent));
                        end
                end
            end
    end;

    {-----------------------------------------------------------------------------}
    {Procedure HRHandleMove handles the first event = Move.                       }
    {-----------------------------------------------------------------------------}

    procedure HRHandleMove;
        var
            temp, nextEvent, nextNextEvent, traverse, previous, nextTraverse, nextPrevious, deletePtr: WPERPtr;
            selectFlag, nextSelectFlag: boolean;
    begin
        temp := WPEvents^;
        nextEvent := WPEvents^.next;
        selectFlag := FALSE;
        if nextEvent <> nil then
            begin
                if (nextEvent^.theEvent = Select) or (nextEvent^.theEvent = MoveIP) then
                    begin
                        traverse := nextEvent;
                        previous := traverse;
                        while (traverse^.next <> nil) and (not ((traverse^.theEvent = Select) and (traverse^.from = menu))) and ((traverse^.theEvent = MoveIP) or
    (traverse^.theEvent = Select)) do
                            begin
                                previous := traverse;
                                traverse := traverse^.next;
                            end;
                        if (traverse^.theEvent = Select) and (traverse^.from = menu) then
                            selectFlag := TRUE;
                        if (traverse^.theEvent = PressOK) or selectFlag then
                            if (previous^.theEvent = Select) or selectFlag then
                                begin
                                    TOC^^.method := MoveTOC;
                                    TOC^^.IPMoved := FALSE;
                                    TOC^^.eventSince := FALSE;

                                    nextNextEvent := traverse^.next;
```

```
                    if nextNextEvent <> nil then
                      begin
                        if (nextNextEvent^.theEvent = Select) or (nextNextEvent^.theEvent = MoveIP) then
                          begin
                            nextSelectFlag := FALSE;
                            nextTraverse  := nextNextEvent;
                            nextPrevious  := nextTraverse;
                            while (nextTraverse^.next <> nil) and (not ((nextTraverse^.theEvent = Select) and (nextTraverse^.from = menu)))
and ((nextTraverse^.theEvent = MoveIP) or (nextTraverse^.theEvent = Select)) do
                              begin
                                nextPrevious  := nextTraverse;
                                nextTraverse  := nextTraverse^.next;
                              end;
                            if (nextTraverse^.theEvent = Select) and (nextTraverse^.from = menu) then
                              nextSelectFlag := TRUE;
                            if (nextTraverse^.theEvent = PressOK) or nextSelectFlag then
                              begin
                                HR^^.HRnumMove := HR^^.HRnumMove + 1;
                                if temp^.from = menu then
                                  HR^^.HRMovesMenu := HR^^.HRMovesMenu + 1
                                else
                                  HR^^.HRMovesPalette := HR^^.HRMovesPalette + 1;
                                RemoveIPSelect;
                                if (nextPrevious = nextTraverse) and nextSelectFlag then
                                  begin
                                    if WPEvents <> nil then
                                      if WPEvents^^.next <> nil then
                                        begin
                                          temp := WPEvents^^.next;
                                          AssignValues(WPEvents, temp);
                                          DisposPtr(Ptr(temp));
                                        end
                                      else
                                        begin
                                          DisposHandle(Handle(WPEvents));
                                          WPEvents := nil;
                                        end
                                  end
                                else
                                  RemoveIPSelect;
                              end
                            else if (nextTraverse^.next <> nil) then    {nextTraverse <> PressOK and not nextSelectFlag}
                              begin
                                RemoveIPSelect;
                                RemoveIPSelect;
                              end
                          end
                        else    {nextNextEvent<>Select and nextNextEvent<>MoveIP}
                          begin
                            RemoveIPSelect;
                            if WPEvents <> nil then
                              begin
                                nextEvent := WPEvents^^.next;
                                if nextEvent = nil then
                                  begin
                                    DisposHandle(Handle(WPEvents));
                                    WPEvents := nil;
                                  end
                                else    {nextEvent <> nil}
                                  begin
                                    deletePtr  := nextEvent;
                                    AssignValues(WPEvents, nextEvent);
                                    DisposPtr(Ptr(deletePtr));
                                  end
                              end;
                          end    {else nextNextEvent <> Select and nextNextEvent <> MoveIP}
                      end    {nextNextEvent <> nil}
                  end    {previous event = select}
                else    {previous event <> select, but there was a PressOK or SelectFlag}
                  begin
                    nextNextEvent := traverse^.next;
                    if nextNextEvent <> nil then
                      begin
                        if (nextNextEvent^.theEvent = PressOK) or (nextNextEvent^.theEvent = PressCancel) or ((nextNextEvent^.theEvent =
Select) and (nextNextEvent^.from = menu)) then
                          begin
{Get rid of the whole thing}
                            RemoveIPSelect;
                            if WPEvents <> nil then
                              begin
                                nextEvent := WPEvents^^.next;
                                if nextEvent = nil then
                                  begin
                                    DisposHandle(Handle(WPEvents));
                                    WPEvents := nil;
                                  end
                                else    {nextEvent <> nil}
                                  begin
                                    deletePtr  := nextEvent;
                                    AssignValues(WPEvents, nextEvent);
                                    DisposPtr(Ptr(deletePtr));
                                  end
                              end;
                          end
                        else    {nextNextEvent <> PressOK or PressCancel or MenuSelect}
                          begin
{Keep going until PressOK or PressCancel, get rid of the whole thing}
                            nextSelectFlag := FALSE;
                            nextTraverse  := nextNextEvent;
                            nextPrevious  := nextTraverse;
                            while (nextTraverse^.next <> nil) and (not ((nextTraverse^.theEvent = Select) and (nextTraverse^.from = menu)))
and ((nextTraverse^.theEvent = MoveIP) or (nextTraverse^.theEvent = Select)) do
                              begin
                                nextPrevious  := nextTraverse;
                                nextTraverse  := nextTraverse^.next;
                              end;
                            if (nextTraverse^.theEvent = select) and (nextTraverse^.from = menu) then
                              nextSelectFlag := TRUE;
```

```
                                    if (nextTraverse^.theEvent = PressOK) or (nextTraverse^.theEvent = PressCancel) then
                                       begin
                                          RemoveIPSelect;
                                          if (nextPrevious = nextTraverse) and nextSelectFlag then
                                             begin
                                                if WPEvents <> nil then
                                                   if WPEvents^^.next <> nil then
                                                      begin
                                                         temp := WPEvents^^.next;
                                                         AssignValues(WPEvents, temp);
                                                         DisposPtr(Ptr(temp));
                                                      end
                                                   else
                                                      begin
                                                         DisposHandle(Handle(WPEvents));
                                                         WPEvents := nil;
                                                      end
                                             end
                                          else
                                             RemoveIPSelect;
                                       end
                                 end
                              end
                           end
                        else if (traverse^.next <> nil) then            traverse <> PressOK)
                           RemoveIPSelect
                     end
                  else if nextEvent^.theEvent = PressCancel then
                     begin
Get rid of Duplicate and PressCancel)
                        if nextEvent^.next <> nil then
                           begin
                              deletePtr := nextEvent^.next;
                              AssignValues(WPEvents, nextEvent^.next);
                              DisposPtr(Ptr(deletePtr));
                           end
                        else
                           begin
                              DisposHandle(Handle(WPEvents));
                              WPEvents := nil
                           end;
                        DisposPtr(Ptr(nextEvent));
                     end
                  else if nextEvent^.theEvent = PressOK then
                     begin
Keep going until another pressOK or pressCancel, and delete everything but selects)
                        traverse := nextEvent^.next;
                        if traverse <> nil then
                           if (traverse^.theEvent = Select) or (traverse^.theEvent = MoveIP) then
                              begin
Get rid of Duplicate,PressOK,MoveIPSelect,PressOK or PressCancel
                                 previous := traverse;
                                 while (traverse^.next <> nil) and (not ((traverse^.theEvent = Select) and (traverse^.from = menu))) and ((traverse^.theEvent =
MoveIP) or (traverse^.theEvent = Select)) do
                                    begin
                                       previous := traverse;
                                       traverse := traverse^.next
                                    end;
                                 if (traverse^.theEvent = PressOK) or (traverse^.theEvent = PressCancel) or ((traverse^.theEvent = Select) and (traverse^.from
= menu)) then
                                    begin
                                       deletePtr := nextEvent;
                                       AssignValues(WPEvents, nextEvent);
                                       DisposPtr(Ptr(deletePtr));
                                       RemoveIPSelect;
                                    end
                              end
                           else      traverse^.theEvent=Select or MoveIP\
                              begin
Get rid of Duplicate,PressOK, PressOK or PressCancel
                                 if traverse^.next <> nil then
                                    begin
                                       deletePtr := traverse^.next;
                                       AssignValues(WPEvents, traverse^.next);
                                       DisposPtr(Ptr(deletePtr));
                                    end
                                 else
                                    begin
                                       DisposHandle(Handle(WPEvents));
                                       WPEvents := nil
                                    end;
                                 DisposPtr(Ptr(traverse));
                                 DisposPtr(Ptr(nextEvent));
                              end
                     end
               end
         end;

   {.....
   Procedure HRHandleDuplicate handles the first event = Duplicate.
   .....}

   procedure HRHandleDuplicate;
      var
         temp, nextEvent, nextNextEvent, traverse, previous, nextTraverse, nextPrevious, deletePtr: WPERPtr;
         selectFlag, nextSelectFlag: boolean;
   begin
      temp := WPEvents^;
      nextEvent := WPEvents^^.next;
      selectFlag := FALSE;
      if nextEvent <> nil then
         begin
            if (nextEvent^.theEvent = Select) or (nextEvent^.theEvent = MoveIP) then
               begin
                  traverse := nextEvent;
                  previous := traverse;
                  while (traverse^.next <> nil) and (not ((traverse^.theEvent = Select) and (traverse^.from = menu))) and ((traverse^.theEvent = MoveIP) or
(traverse^.theEvent = Select)) do
```

```
                    begin
                      previous := traverse;
                      traverse := traverse^.next;
                    end;
                  if (traverse^.theEvent = Select) and (traverse^.from = menu) then
                    selectFlag := TRUE;
                  if (traverse^.theEvent = PressOK) or selectFlag then
                    if (previous^.theEvent = Select) or selectFlag then
                      begin
                        TOC^^.method := DuplicateTOC;
                        TOC^^.IPMoved := FALSE;
                        TOC^^.eventSince := FALSE;

                        nextNextEvent := traverse^.next;
                        if nextNextEvent <> nil then
                          begin
                            if (nextNextEvent^.theEvent = Select) or (nextNextEvent^.theEvent = MoveIP) then
                              begin
                                nextSelectFlag := FALSE;
                                nextTraverse := nextNextEvent;
                                nextPrevious := nextTraverse;
                                while (nextTraverse^.next <> nil) and (not ((nextTraverse^.theEvent = Select) and (nextTraverse^.from = menu))
and ((nextTraverse^.theEvent = MoveIP) or (nextTraverse^.theEvent = Select)) do
                                  begin
                                    nextPrevious := nextTraverse;
                                    nextTraverse := nextTraverse^.next;
                                  end;
                                if (nextTraverse^.theEvent = Select) and (nextTraverse^.from = menu) then
                                  nextSelectFlag := TRUE;
                                if (nextTraverse^.theEvent = PressOK) or nextSelectFlag then
                                  begin
                                    HR^^.HRnumDuplicate := HR^^.HRnumDuplicate + 1;
                                    HR^^.HRDuplicate2 := HR^^.HRDuplicate2 + 1;
                                    RemoveIPSelect;
                                    if (nextPrevious = nextTraverse) and nextSelectFlag then
                                      begin
                                        if WPEvents <> nil then
                                          if WPEvents^^.next <> nil then
                                            begin
                                              temp := WPEvents^^.next;
                                              AssignValues(WPEvents, temp);
                                              DisposPtr(Ptr(temp));
                                            end
                                          else
                                            begin
                                              DisposHandle(Handle(WPEvents));
                                              WPEvents := nil;
                                            end
                                      end
                                    else
                                      RemoveIPSelect;
                                  end
                                else if (nextTraverse^.next <> nil) then    {nextTraverse <> PressOK and not nextSelectFlag}
                                  begin
                                    RemoveIPSelect;
                                    RemoveIPSelect;
                                  end
                              end
                            else    {nextNextEvent<>Select and nextNextEvent<>MoveIP}
                              begin
                                RemoveIPSelect;
                                if WPEvents <> nil then
                                  begin
                                    nextEvent := WPEvents^^.next;
                                    if nextEvent = nil then
                                      begin
                                        DisposHandle(Handle(WPEvents));
                                        WPEvents := nil;
                                      end
                                    else          {nextEvent <> nil}
                                      begin
                                        deletePtr := nextEvent;
                                        AssignValues(WPEvents, nextEvent);
                                        DisposPtr(Ptr(deletePtr));
                                      end
                                  end;
                              end          {else nextNextEvent <> Select and nextNextEvent <> MoveIP}
                          end          {nextNextEvent <> nil}
                      end          {previous event = select}
                    else          {previous event <> select, but there was a PressOK or SelectFlag}
                      begin
                        nextNextEvent := traverse^.next;
                        if nextNextEvent <> nil then
                          begin
                            if (nextNextEvent^.theEvent = PressOK) or (nextNextEvent^.theEvent = PressCancel) or ((nextNextEvent^.theEvent =
Select) and (nextNextEvent^.from = menu)) then
                              begin
{Get rid of the whole thing}
                                RemoveIPSelect;
                                if WPEvents <> nil then
                                  begin
                                    nextEvent := WPEvents^^.next;
                                    if nextEvent = nil then
                                      begin
                                        DisposHandle(Handle(WPEvents));
                                        WPEvents := nil;
                                      end
                                    else          {nextEvent <> nil}
                                      begin
                                        deletePtr := nextEvent;
                                        AssignValues(WPEvents, nextEvent);
                                        DisposPtr(Ptr(deletePtr));
                                      end
                                  end;
                              end
                            else          {nextNextEvent <> PressOK or PressCancel or MenuSelect}
                              begin
{Keep going until PressOK or PressCancel, get rid of the whole thing}
```

```
                                    nextSelectFlag := FALSE;
                                    nextTraverse  := nextNextEvent;
                                    nextPrevious  := nextTraverse;
                                    while (nextTraverse^.next <> nil) and (not ((nextTraverse^.theEvent = Select) and (nextTraverse^.from = menu)))
and ((nextTraverse^.theEvent = MoveIP) or (nextTraverse^.theEvent = Select)) do
                                       begin
                                          nextPrevious  := nextTraverse;
                                          nextTraverse  := nextTraverse^.next;
                                       end;
                                    if (nextTraverse^.theEvent = select) and (nextTraverse^.from = menu) then
                                       nextSelectFlag := TRUE;
                                    if (nextTraverse^.theEvent = PressOK) or (nextTraverse^.theEvent = PressCancel) then
                                       begin
                                          RemoveIPSelect;
                                          if (nextPrevious = nextTraverse) and nextSelectFlag then
                                             begin
                                                if WPEvents <> nil then
                                                   if WPEvents^^.next <> nil then
                                                      begin
                                                         temp := WPEvents^^.next;
                                                         AssignValues(WPEvents, temp);
                                                         DisposPtr(Ptr(temp));
                                                      end
                                                   else
                                                      begin
                                                         DisposHandle(Handle(WPEvents));
                                                         WPEvents := nil;
                                                      end
                                             end
                                          else
                                             RemoveIPSelect;
                                       end
                                 end
                              end
                           else if (traverse^.next <> nil) then            traverse <> PressOK
                              RemoveIPSelect
                        end
                     else if nextEvent^.theEvent = PressCancel then
                        begin
  Get rid of Duplicate and PressCancel
                           if nextEvent^.next <> nil then
                              begin
                                 deletePtr := nextEvent^.next;
                                 AssignValues(WPEvents, nextEvent^.next);
                                 DisposPtr(Ptr(deletePtr));
                              end
                           else
                              begin
                                 DisposHandle(Handle(WPEvents));
                                 WPEvents := nil
                              end;
                           DisposPtr(Ptr(nextEvent));
                        end
                     else if nextEvent^.theEvent = PressOK then
                        begin
  Keep going until another pressOK or pressCancel, and delete everything but selects)
                           traverse  := nextEvent^.next;
                           if traverse <> nil then
                              if (traverse^.theEvent = Select) or (traverse^.theEvent = MoveIP) then
                                 begin
  Get rid of Duplicate,PressOK,MoveIPSelect,PressOK  or  PressCancel
                                    previous  := traverse;
                                    while (traverse^.next <> nil) and (not ((traverse^.theEvent = Select) and (traverse^.from = menu))) and ((traverse^.theEvent =
MoveIP) or (traverse^.theEvent = Select)) do
                                       begin
                                          previous  := traverse;
                                          traverse  := traverse^.next
                                       end;
                                    if (traverse^.theEvent = PressOK) or (traverse^.theEvent = PressCancel) or ((traverse^.theEvent = Select) and (traverse^.from
= menu)) then
                                       begin
                                          deletePtr  := nextEvent;
                                          AssignValues(WPEvents, nextEvent);
                                          DisposPtr(Ptr(deletePtr));
                                          RemoveIPSelect;
                                       end
                                 end
                              else      traverse^.theEvent=Select or MoveIP)
                                 begin
  Get rid of Duplicate,PressOK, PressOK or PressCancel)
                                    if traverse^.next <> nil then
                                       begin
                                          deletePtr := traverse^.next;
                                          AssignValues(WPEvents, traverse^.next);
                                          DisposPtr(Ptr(deletePtr));
                                       end
                                    else
                                       begin
                                          DisposHandle(Handle(WPEvents));
                                          WPEvents := nil
                                       end;
                                    DisposPtr(Ptr(traverse));
                                    DisposPtr(Ptr(nextEvent));
                                 end
                           end
                     end
                  end;
               end.
```

```
{
  Margaret Stone
  Sc.M. Project, Brown University

  This unit contains some of the compiler-like code to determine if the latest word-processor event
  will add a user strategy to the user help profile.
}

unit EditorHelp3;

{
  Interface Section
}

interface

  uses
    EditorGlobals, EditorHelpIntf, EditorHelp2;

  procedure HRDeleteEvent (DeleteEvent: WPERPtr; Previous: WPERPtr);
  procedure RemoveToHR;
  procedure HRHandleSelect (TextSelected: boolean);

{
  Implementation Section
}

implementation

{
  Procedure HRDeleteEvent deletes an event from the queue.
}

  procedure HRDeleteEvent;        {DeleteEvent: WPERPtr; Previous: WPERPtr}
    var
      nextEvent, deletePtr: WPERPtr;
  begin
    nextEvent := DeleteEvent^.next;
    if Previous = nil then
      begin
        if nextEvent <> nil then
          begin
            deletePtr := nextEvent;
            AssignValues(WPEvents, nextEvent);
            DisposPtr(Ptr(deletePtr))
          end
        else
          begin
            DisposHandle(Handle(WPEvents));
            WPEvents := nil
          end
      end
    else      {previous<>nil}
      Previous^.next := nextEvent;
    DisposPtr(Ptr(DeleteEvent));
  end;

{
  Procedure RemoveToHR handles events which singly can be extracted into the user help
  profile.
}

  procedure RemoveToHR;
    var
      nextEvent, deletePtr, temp: WPERPtr;
  begin
    temp := WPEvents^;
    case temp^.theEvent of
      ClickDnArrow:
        begin
          HR^^.HRnumViewN := HR^^.HRnumViewN + 1;
          HR^^.HRClickDnArrow := HR^^.HRClickDnArrow + 1
        end;
      ClickUpArrow:
        begin
          HR^^.HRnumViewP := HR^^.HRnumViewP + 1;
          HR^^.HRClickUpArrow := HR^^.HRClickUpArrow + 1;
        end;
      ClickDnArrowRepeat:
        begin
          HR^^.HRnumViewN := HR^^.HRnumViewN + 1;
          HR^^.HRClickDnArrowRepeat := HR^^.HRClickDnArrowRepeat + 1;
        end;
      ClickUpArrowRepeat:
        begin
          HR^^.HRnumViewP := HR^^.HRnumViewP + 1;
          HR^^.HRClickUpArrowRepeat := HR^^.HRClickUpArrowRepeat + 1;
        end;
      PressDnArrow:
        begin
          HR^^.HRnumViewN := HR^^.HRnumViewN + 1;
          HR^^.HRPressDnArrow := HR^^.HRPressDnArrow + 1
        end;
      PressUpArrow:
        begin
          HR^^.HRnumViewP := HR^^.HRnumViewP + 1;
          HR^^.HRPressUpArrow := HR^^.HRPressUpArrow + 1;
        end;
      DragThumb:
        HR^^.HRDragThumb := HR^^.HRDragThumb + 1;
      ScreenP:
        begin
          HR^^.HRnumViewP := HR^^.HRnumViewP + 1
          if temp^.from = menu then
            HR^^.HRScreenPMenu := HR^^.HRScreenPMenu + 1
          else if temp^.from = key then
            HR^^.HRScreenPKey := HR^^.HRScreenPKey + 1
          else if temp^.from = mouse then
```

```
                HR^^.HRScreenPtMouse := HR^^.HRScreenPtMouse + 1
            end;
        ScreenN:
            begin
                HR^^.HRnumViewN := HR^^.HRnumViewN + 1;
                if temp^.from = menu then
                    HR^^.HRScreenNMenu := HR^^.HRScreenNMenu + 1
                else if temp^.from = key then
                    HR^^.HRScreenNKey := HR^^.HRScreenNKey + 1
                else if temp^.from = mouse then
                    HR^^.HRScreenNMouse := HR^^.HRScreenNMouse + 1
            end;
        LineP:
            begin
                HR^^.HRnumViewP := HR^^.HRnumViewP + 1
                if temp^.from = menu then
                    HR^^.HRLinePMenu := HR^^.HRLinePMenu + 1
                else if temp^.from = key then
                    HR^^.HRLinePKey := HR^^.HRLinePKey + 1
            end;
        LineN:
            begin
                HR^^.HRnumViewN := HR^^.HRnumViewN + 1;
                if temp^.from = menu then
                    HR^^.HRLineNMenu := HR^^.HRLineNMenu + 1
                else if temp^.from = key then
                    HR^^.HRLineNKey := HR^^.HRLineNKey + 1
            end;
        SizeChange:
            begin
                TutorPaste := FALSE;
                TutorFontSet := FALSE;
                HR^^.HRnumSize := HR^^.HRnumSize + 1;
                if temp^.from = menu then
                    HR^^.HRText1Menu := HR^^.HRText1Menu + 1
                else if temp^.from = palette then
                    HR^^.HRText1Palette := HR^^.HRText1Palette + 1
            end;
        FontChange:
            begin
                TutorPaste := FALSE;
                TutorFontSet := FALSE;
                HR^^.HRnumFont := HR^^.HRnumFont + 1;
                if temp^.from = menu then
                    HR^^.HRText2Menu := HR^^.HRText2Menu + 1
                else if temp^.from = palette then
                    HR^^.HRText2Palette := HR^^.HRText2Palette + 1
            end;
        StyleChange:
            begin
                TutorPaste := FALSE;
                TutorFontSet := FALSE;
                HR^^.HRnumStyle := HR^^.HRnumStyle + 1;
                if temp^.from = menu then
                    HR^^.HRText3Menu := HR^^.HRText3Menu + 1
                else if temp^.from = palette then
                    HR^^.HRText3Palette := HR^^.HRText3Palette + 1
            end;
        otherwise

    end;
    nextEvent := temp^.next;
    if nextEvent <> nil then
        begin
            deletePtr := nextEvent;
            AssignValues(WPEvents, nextEvent);
            DisposPtr(Ptr(deletePtr));
        end
    else
        begin
            DisposHandle(Handle(WPEvents));
            WPEvents := nil
        end;
end;

{-------------------------------------------------------------------------------
Procedure HRHandleSelect handles the first event = Select.
-------------------------------------------------------------------------------}

procedure HRHandleSelect        (TextSelected: boolean));
    var
        nextEvent, nextNextEvent, temp, traverse, previous, deletePtr: WPERPtr;
        TOCFlag: boolean;
begin
    if TextSelected then
        begin
            TOCFlag := TRUE;
            nextEvent := WPEvents^;
            temp := WPEvents^
        end
    else
        begin
            TOCFlag := FALSE;
            nextEvent := WPEvents^.next;
            temp := WPEvents^;
        end;

    if nextEvent <> nil then
        begin
            case nextEvent^.theEvent of
                Cut:
                    begin
                        TutorFontSet := FALSE;
                        TutorPaste := FALSE;

                        HR^^.HRnumCut := HR^^.HRnumCut + 1;
                        if nextEvent^.from = menu then
                            HR^^.HRCut1Menu := HR^^.HRCut1Menu + 1
```

```
      else if nextEvent^.from = key then
         HR^^.HRCut1Key := HR^^.HRCut1Key + 1
      else if nextEvent^.from = palette then
         HR^^.HRCut1Palette := HR^^.HRCut1Palette + 1;

      TOC^^.method := CutTOC;
      TOC^^.IPMoved := FALSE;
      TOC^^.eventSince := FALSE;

      if not TOCFlag then
         begin
            HR^^.HRnumSelects := HR^^.HRnumSelects + 1;
            if temp^.from = menu then
               HR^^.HRSelectFromMenu := HR^^.HRSelectFromMenu + 1
            else if temp^.from = key then
               HR^^.HRSelectFromKey := HR^^.HRSelectFromKey + 1
            else if temp^.from = mouse then
               HR^^.HRSelectFromMouse := HR^^.HRSelectFromMouse + 1;
         end;

      if nextEvent^.next <> nil then
         begin
            deletePtr := nextEvent^.next;
            AssignWValues(WPEvents, nextEvent^.next);
            DisposePtr(Ptr(deletePtr));
         end
      else
         begin
            DisposeHandle(Handle(WPEvents));
            WPEvents := nil
         end;
      DisposePtr(ptr(nextEvent))
   end;

Copy:
   begin
      TUserFontSet := FALSE;
      TUserPaste := FALSE;

      HR^^.HRnumCopy := HR^^.HRnumCopy + 1;
      if nextEvent^.from = menu then
         HR^^.HRCopy1Menu := HR^^.HRCopy1Menu + 1
      else if nextEvent^.from = key then
         HR^^.HRCopy1Key := HR^^.HRCopy1Key + 1
      else if nextEvent^.from = paste then
         HR^^.HRCopy1Paste := HR^^.HRCopy1Paste + 1;

      TOC^^.method := CopyTOC;
      TOC^^.IPMoved := FALSE;
      TOC^^.EventSince := FALSE;

      if not TOCFlag then
         begin
            HR^^.HRnumSelects := HR^^.HRnumSelects + 1;
            if temp^.from = menu then
               HR^^.HRSelectFromMenu := HR^^.HRSelectFromMenu + 1
            else if temp^.from = key then
               HR^^.HRSelectFromKey := HR^^.HRSelectFromKey + 1
            else if temp^.from = mouse then
               HR^^.HRSelectFromMouse := HR^^.HRSelectFromMouse + 1;
         end;

      if nextEvent^.next <> nil then
         begin
            deletePtr := nextEvent^.next;
            AssignWValues(WPEvents, nextEvent^.next);
            DisposePtr(Ptr(deletePtr));
         end
      else
         begin
            DisposeHandle(Handle(WPEvents));
            WPEvents := nil;
         end;
      DisposePtr(ptr(nextEvent));
   end;

UserType:
   begin
      TOC^^.IPMoved := TRUE;
      TOC^^.EventSince := TRUE;
      HR^^.HRnumReplace := HR^^.HRnumReplace + 1
      HR^^.HRReplaced := HR^^.HRReplaced + 1;

      if not TOCFlag then
         begin
            HR^^.HRnumSelects := HR^^.HRnumSelects + 1;
            if temp^.from = menu then
               HR^^.HRSelectFromMenu := HR^^.HRSelectFromMenu + 1
            else if temp^.from = key then
               HR^^.HRSelectFromKey := HR^^.HRSelectFromKey + 1
            else if temp^.from = mouse then
               HR^^.HRSelectFromMouse := HR^^.HRSelectFromMouse + 1;
         end;

      if nextEvent^.next <> nil then
         begin
            deletePtr := nextEvent^.next;
            AssignWValues(WPEvents, nextEvent^.next);
            DisposePtr(Ptr(deletePtr));
         end
      else
         begin
            DisposeHandle(Handle(WPEvents));
            WPEvents := nil
         end;
      DisposePtr(ptr(nextEvent));
   end;
```

```
DeleteType:
  begin
    if nextEvent^.next <> nil then
      begin
        nextNextEvent := nextEvent^.next;

        if TOCFlag then
          if (TOC^^.method = CopyTOC) and (not TOC^^.EventSince) then
            begin
              TOC^^.method := CopyDeleteType;
              TOC^^.EventSince := TRUE;
              if nextEvent^.next <> nil then
                begin
                  deletePtr := nextEvent^.next;
                  AssignValues(WPEvents, nextEvent^.next);
                  DisposPtr(Ptr(deletePtr))
                end
              else
                begin
                  DisposHandle(Handle(WPEvents));
                  WPEvents := nil
                end
            end
          else
            begin
              TOC^^.IPMoved := TRUE;
              TOC^^.EventSince := TRUE;
            end;

        if not TOCFlag then
          begin
            HR^^.HRnumSelects := HR^^.HRnumSelects + 1;
            if temp^.from = menu then
              HR^^.HRSelectFromMenu := HR^^.HRSelectFromMenu + 1
            else if temp^.from = key then
              HR^^.HRSelectFromKey := HR^^.HRSelectFromKey + 1
            else if temp^.from = mouse then
              HR^^.HRSelectFromMouse := HR^^.HRSelectFromMouse + 1;
          end;

        if nextNextEvent^.theEvent = UserType then
          begin
            HR^^.HRnumReplace := HR^^.HRnumReplace + 1;
            HR^^.HRReplace2 := HR^^.HRReplace2 + 1;
            HR^^.HRnumDelete := HR^^.HRnumDelete + 1;
            HR^^.HRDelete2 := HR^^.HRDelete2 + 1;
            if nextNextEvent^.next <> nil then
              begin
                deletePtr := nextNextEvent^.next;
                AssignValues(WPEvents, nextNextEvent^.next);
                DisposPtr(Ptr(deletePtr));
              end
            else
              begin
                DisposHandle(Handle(WPEvents));
                WPEvents := nil;
              end;
            DisposPtr(ptr(nextEvent));
            DisposPtr(ptr(nextNextEvent));
          end
        else
          begin
            HR^^.HRnumDelete := HR^^.HRnumDelete + 1;
            HR^^.HRDelete2 := HR^^.HRDelete2 + 1;
            deletePtr := nextNextEvent;
            AssignValues(WPEvents, nextNextEvent);
            DisposPtr(Ptr(deletePtr));
            DisposPtr(ptr(nextEvent));
          end
      end
  end;

Delete:
  begin
    UserFontSet := FALSE;
    UserPaste := FALSE;

    if nextEvent^.next <> nil then
      begin
        nextNextEvent := nextEvent^.next;

        if TOCFlag then
          if (TOC^^.method = CopyTOC) and (not TOC^^.EventSince) then
            begin
              TOC^^.method := CopyDelete;
              TOC^^.EventSince := TRUE;
              if nextEvent^.next <> nil then
                begin
                  deletePtr := nextEvent^.next;
                  AssignValues(WPEvents, nextEvent^.next);
                  DisposPtr(Ptr(deletePtr))
                end
              else
                begin
                  DisposHandle(Handle(WPEvents));
                  WPEvents := nil
                end
            end
          else
            begin
              TOC^^.IPMoved := TRUE;
              TOC^^.EventSince := TRUE
            end;

        if not TOCFlag then
          begin
            HR^^.HRnumSelects := HR^^.HRnumSelects + 1;
            if temp^.from = menu then
```

```
                        HR^^.HRSelectFromMenu := HR^^.HRSelectFromMenu + 1
                      else if temp^.from = key then
                        HR^^.HRSelectFromKey := HR^^.HRSelectFromKey + 1
                      else if temp^.from = mouse then
                        HR^^.HRSelectFromMouse := HR^^.HRSelectFromMouse + 1;
                    end;

                  if nextNextEvent^.theEvent = UserType then
                    begin
                      HR^^.HRnumReplace := HR^^.HRnumReplace + 1;
                      if nextEvent^.from = menu then
                        HR^^.HRReplace3FromMenu := HR^^.HRReplace3FromMenu + 1
                      else
                        HR^^.HRReplace3FromPalette := HR^^.HRReplace3FromPalette + 1;
                      HR^^.HRnumDelete := HR^^.HRnumDelete + 1;
                      if nextEvent^.from = menu then
                        HR^^.HRDelete3FromMenu := HR^^.HRDelete3FromMenu + 1
                      else
                        HR^^.HRDelete3FromPalette := HR^^.HRDelete3FromPalette + 1;
                      if nextNextEvent^.next <> nil then
                        begin
                          deletePtr := nextNextEvent^.next;
                          AssignValues(WPEvents, nextNextEvent^.next);
                          DisposePtr(Ptr(deletePtr));
                        end
                      else
                        begin
                          DisposeHandle(Handle(WPEvents));
                          WPEvents := nil;
                        end;
                      DisposePtr(ptr(nextEvent));
                      DisposePtr(ptr(nextNextEvent));
                    end
                  else
                    begin
                      HR^^.HRnumDelete := HR^^.HRnumDelete + 1;
                      if nextEvent^.from = menu then
                        HR^^.HRDelete3FromMenu := HR^^.HRDelete3FromMenu + 1
                      else
                        HR^^.HRDelete3FromPalette := HR^^.HRDelete3FromPalette + 1;
                      deletePtr := nextNextEvent;
                      AssignValues(WPEvents, nextNextEvent);
                      DisposePtr(Ptr(deletePtr));
                      DisposePtr(ptr(nextEvent));
                    end
                end
            end;

          Duplicate:
            begin
              TutorFontSet := FALSE;
              TutorPaste := FALSE;

              nextNextEvent := nextEvent^.next;
              if nextNextEvent <> nil then
                begin
                  if ((nextNextEvent^.theEvent = MoveP) or (nextNextEvent^.theEvent = Select)) then
                    begin
                      traverse := nextNextEvent;
                      previous := traverse;
                      while (traverse^.next <> nil) and (not ((traverse^.theEvent = Select) and (traverse^.from = menu))) and
traverse^.theEvent = MoveP) or (traverse^.theEvent = Select) do
                        begin
                          previous := traverse;
                          traverse := traverse^.next;
                        end;
                      if (traverse^.theEvent = PressOK) or ((previous^.theEvent = Select) and (previous^.from = menu)) then
                        begin
                          if not TOCFlag then
                            begin
                              HR^^.HRnumSelects := HR^^.HRnumSelects + 1;
                              if temp^.from = menu then
                                HR^^.HRSelectFromMenu := HR^^.HRSelectFromMenu + 1
                              else if temp^.from = key then
                                HR^^.HRSelectFromKey := HR^^.HRSelectFromKey + 1
                              else if temp^.from = mouse then
                                HR^^.HRSelectFromMouse := HR^^.HRSelectFromMouse + 1;
                            end;
                          TOC^^.method := DuplicateTOC;
                          TOC^^.ifMoved := FALSE;
                          TOC^^.eventsince := FALSE;
shouldn't actually be a menu select here)
                          if not TOCFlag then
                            begin
                              deletePtr := nextEvent;
                              AssignValues(WPEvents, nextEvent);
                              DisposePtr(Ptr(deletePtr));
                            end;
                          RemovePSelect;
                          HR^^.HRnumDuplicate := HR^^.HRnumDuplicate + 1;
                          HR^^.HRDuplicate1 := HR^^.HRDuplicate1 + 1
                        end
                      else if traverse^.next <> nil then
                        begin
                          if not TOCFlag then
                            begin
                              HR^^.HRnumSelects := HR^^.HRnumSelects + 1;
                              if temp^.from = menu then
                                HR^^.HRSelectFromMenu := HR^^.HRSelectFromMenu + 1
                              else if temp^.from = key then
                                HR^^.HRSelectFromKey := HR^^.HRSelectFromKey + 1
                              else if temp^.from = mouse then
                                HR^^.HRSelectFromMouse := HR^^.HRSelectFromMouse + 1
                              deletePtr := nextEvent;
                              AssignValues(WPEvents, nextEvent);
                              DisposePtr(Ptr(deletePtr));
                            end;
                          RemovePSelect;
```

```
                    end;
                end
            else if (nextNextEvent^.theEvent = PressOK) or (nextNextEvent^.theEvent = PressCancel) then
                begin
                    if not TOCFlag then
                        begin
                            HR^^ HRnumSelects := HR^^ HRnumSelects + 1;
                            if temp^.from = menu then
                                HR^^ HRSelectFromMenu := HR^^ HRSelectFromMenu + 1
                            else if temp^.from = key then
                                HR^^ HRSelectFromKey := HR^^ HRSelectFromKey + 1
                            else if temp^.from = mouse then
                                HR^^ HRSelectFromMouse := HR^^ HRSelectFromMouse + 1;
                        end;
                    if nextNextEvent^.next <> nil then
                        begin
                            deletePtr := nextNextEvent^.next;
                            AssignValues(WPEvents, nextNextEvent^.next);
                            DisposPtr(Ptr(deletePtr));
                        end
                    else
                        begin
                            DisposHandle(Handle(WPEvents));
                            WPEvents := nil;
                        end;
                    DisposPtr(Ptr(nextNextEvent));
                    DisposPtr(Ptr(nextEvent));
                end
        end
    end;

Replace:
    begin
        UserFontSet := FALSE;
        UserPaste := FALSE;

        TOC^^.IPMoved := TRUE;
        nextNextEvent := nextEvent^.next;
        if nextNextEvent <> nil then
            begin
                if ((nextNextEvent^.theEvent = UserType) or (nextNextEvent^.theEvent = DeleteType)) then
                    begin
                        traverse := nextNextEvent;
                        previous := traverse;
                        while (traverse^.next <> nil) and ((traverse^.theEvent = DeleteType) or (traverse^.theEvent = UserType)) do
                            begin
                                previous := traverse;
                                traverse := traverse^.next;
                            end;
                        if (traverse^.theEvent = PressOK) then
                            begin
                                if not TOCFlag then
                                    begin
                                        HR^^ HRnumSelects := HR^^ HRnumSelects + 1;
                                        if temp^.from = menu then
                                            HR^^ HRSelectFromMenu := HR^^ HRSelectFromMenu + 1
                                        else if temp^.from = key then
                                            HR^^ HRSelectFromKey := HR^^ HRSelectFromKey + 1
                                        else if temp^.from = mouse then
                                            HR^^ HRSelectFromMouse := HR^^ HRSelectFromMouse + 1;
                                        deletePtr := nextEvent;
                                        AssignValues(WPEvents, nextEvent);
                                        DisposPtr(Ptr(deletePtr));
                                    end;
                                RemoveKeys;
                                HR^^ HRnumReplace := HR^^ HRnumReplace + 1;
                                if temp^.from = menu then
                                    HR^^ HRReplace7FromMenu := HR^^ HRReplace7FromMenu + 1
                                else if temp^.from = palette then
                                    HR^^ HRReplace7FromPalette := HR^^ HRReplace7FromPalette + 1;
                            end
                        else if traverse^.next <> nil then
                            begin
                                if not TOCFlag then
                                    begin
                                        HR^^ HRnumSelects := HR^^ HRnumSelects + 1;
                                        if temp^.from = menu then
                                            HR^^ HRSelectFromMenu := HR^^ HRSelectFromMenu + 1
                                        else if temp^.from = key then
                                            HR^^ HRSelectFromKey := HR^^ HRSelectFromKey + 1
                                        else if temp^.from = mouse then
                                            HR^^ HRSelectFromMouse := HR^^ HRSelectFromMouse + 1;
                                        deletePtr := nextEvent;
                                        AssignValues(WPEvents, nextEvent);
                                        DisposPtr(Ptr(deletePtr));
                                    end;
                                RemoveKeys;
                            end;
                    end
                else if (nextNextEvent^.theEvent = PressOK) or (nextNextEvent^.theEvent = PressCancel) then
                    begin
                        if nextNextEvent^.next <> nil then
                            begin
                                deletePtr := nextNextEvent^.next;
                                AssignValues(WPEvents, nextNextEvent^.next);
                                DisposPtr(Ptr(deletePtr));
                            end
                        else
                            begin
                                DisposHandle(Handle(WPEvents));
                                WPEvents := nil;
                            end;
                        DisposPtr(Ptr(nextNextEvent));
                        DisposPtr(Ptr(nextEvent));
                    end
            end
    end;
```

```
            MoveP:
               begin
                  TutorFormSet := FALSE;
                  TutorPaste := FALSE;

                  nextNextEvent := nextEvent^.next;
                  if nextNextEvent <> nil then
                     begin
                        if ((nextNextEvent^.theEvent = MoveP) or (nextNextEvent^.theEvent = Select)) then
                           begin
                              traverse := nextNextEvent;
                              previous := traverse;
                              while (traverse^.next <> nil) and (not ((traverse^.theEvent = Select) and (traverse^.from = menu))) and
(traverse^.theEvent = MoveP) or (traverse^.theEvent = Select) do
                                 begin
                                    previous := traverse;
                                    traverse := traverse^.next;
                                 end;
                              if (traverse^.theEvent = PressOK) or ((previous^.theEvent = Select) and (previous^.from = menu)) then
                                 begin
                                    if not TOCFlag then
                                       begin
                                          HR^^.HRnumSelects := HR^^.HRnumSelects + 1;
                                          if temp^.from = menu then
                                             HR^^.HRSelectFromMenu := HR^^.HRSelectFromMenu + 1
                                          else if temp^.from = key then
                                             HR^^.HRSelectFromKey := HR^^.HRSelectFromKey + 1
                                          else if temp^.from = mouse then
                                             HR^^.HRSelectFromMouse := HR^^.HRSelectFromMouse + 1;
                                       end;
                                    TOC^^.method := MoveTOC;
                                    TOC^^.IPMoved := FALSE;
                                    TOC^^.eventSince := FALSE;
shouldn't actually be a menu select here)
                                    if not TOCFlag then
                                       begin
                                          deletePtr := nextEvent;
                                          AssignValues(WPEvents, nextEvent);
                                          DisposPtr(Ptr(deletePtr));
                                       end;
                                    RemoveIPSelect;
                                    HR^^.HRnumMove := HR^^.HRnumMove + 1;
                                    if temp^.from = menu then
                                       HR^^.HRMove7Menu := HR^^.HRMove7Menu + 1
                                    else if temp^.from = palette then
                                       HR^^.HRMove7Palette := HR^^.HRMove7Palette + 1;
                                 end
                              else if traverse^.next <> nil then
                                 begin
                                    if not TOCFlag then
                                       begin
                                          HR^^.HRnumSelects := HR^^.HRnumSelects + 1;
                                          if temp^.from = menu then
                                             HR^^.HRSelectFromMenu := HR^^.HRSelectFromMenu + 1
                                          else if temp^.from = key then
                                             HR^^.HRSelectFromKey := HR^^.HRSelectFromKey + 1
                                          else if temp^.from = mouse then
                                             HR^^.HRSelectFromMouse := HR^^.HRSelectFromMouse + 1;
                                          deletePtr := nextEvent;
                                          AssignValues(WPEvents, nextEvent);
                                          DisposPtr(Ptr(deletePtr));
                                       end;
                                    RemoveIPSelect;
                                 end;
                           end
                        else if (nextNextEvent^.theEvent = PressOK) or (nextNextEvent^.theEvent = PressCancel) then
                           begin
                              if nextNextEvent^.next <> nil then
                                 begin
                                    deletePtr := nextNextEvent^.next;
                                    AssignValues(WPEvents, nextNextEvent^.next);
                                    DisposPtr(Ptr(deletePtr));
                                 end
                              else
                                 begin
                                    DisposHandle(Handle(WPEvents));
                                    WPEvents := nil;
                                 end;
                              DisposPtr(Ptr(nextNextEvent));
                              DisposPtr(Ptr(nextEvent));
                           end
                     end;
               end;

            otherwise
               begin
                  if not TOCFlag then
                     begin
                        TOC^^.IPMoved := TRUE;
                        TOC^^.EventSince := TRUE;
                        HR^^.HRnumSelects := HR^^.HRnumSelects + 1;
                        if temp^.from = menu then
                           HR^^.HRSelectFromMenu := HR^^.HRSelectFromMenu + 1
                        else if temp^.from = key then
                           HR^^.HRSelectFromKey := HR^^.HRSelectFromKey + 1
                        else if temp^.from = mouse then
                           HR^^.HRSelectFromMouse := HR^^.HRSelectFromMouse + 1;
                        deletePtr := nextEvent;
                        AssignValues(WPEvents, nextEvent);
                        DisposPtr(Ptr(deletePtr));
                     end;
               end;
         end
      end
   end
end
```

```
{
  Margaret Stone
  Sc.M. Project, Brown University

  This unit contains some of the computer-like code to determine if the latest word-processor event
  will add a user strategy to the user help profile.
}

unit EditorHelp;

{
  Interface Section
}

interface

  uses
    EditorGlobals, HelpFiling, EditorHelpInit, EditorUtilities, EditorHelp2, EditorHelp3;


  procedure doEvtToHelp;

{
  Implementation Section
}

implementation


{
  Procedure HRHandleSizeSet handles the first event = SizeSet.  In this case, the help system
  may be applying the "naive user" approach to setting the size (giving instructions).  This
  procedure checks for the naive approach, and adjusts the user help profile accordingly.
}

  procedure HRHandleSizeSet;
    var
      nextEvent, temp, traverse, previous, deletePtr: WPERPtr;
  begin
    nextEvent := WPEvents^^.next;
    temp := WPEvents^;
    if nextEvent <> nil then
      begin
        if not (TutorFontSet or MustTutorFont) then
          begin
            HR^^.HRnumSize := HR^^.HRnumSize + 1;
            if temp^.from = menu then
              HR^^.HRText4Menu := HR^^.HRText4Menu + 1
            else if temp^.from = palette then
              HR^^.HRText4Palette := HR^^.HRText4Palette + 1;
            if nextEvent^.theEvent = UserType then
              begin
                if nextEvent^.next <> nil then
                  begin
                    deletePtr := nextEvent^.next;
                    AssignValues(WPEvents, nextEvent^.next);
                    DisposPtr(Ptr(deletePtr));
                  end
                else
                  begin
                    DisposHandle(Handle(WPEvents));
                    WPEvents := nil
                  end;
                DisposPtr(Ptr(nextEvent));
              end
            else
              begin
                if nextEvent = nil then
                  begin
                    DisposHandle(Handle(WPEvents));
                    WPEvents := nil
                  end
                else
                  begin
                    deletePtr := nextEvent;
                    AssignValues(WPEvents, nextEvent);
                    DisposPtr(Ptr(deletePtr));
                  end
              end;
          end
        else {TutorFontSet = TRUE}
          begin
            if ((nextEvent^.theEvent = MoveIP) or (nextEvent^.theEvent = Select)) then
              begin
                TOC^^.IPMoved := TRUE;
                TOC^^.EventSince := TRUE;
                traverse := nextEvent;
                previous := traverse;
                while (traverse^.next <> nil) and (not ((traverse^.theEvent = Select) and (traverse^.from = menu))) and ((traverse^.theEvent =
MoveIP) or (traverse^.theEvent = Select)) do
                  begin
                    previous := traverse;
                    traverse := traverse^.next;
                  end;
                if traverse <> nil then
                  if (traverse^.theEvent = PressOK) or ((previous^.theEvent = Select) and (previous^.from = menu)) then
                    begin
                      RemoveIPSelect;
                      HR^^.HRnumSize := HR^^.HRnumSize + 1;
                      if temp^.from = menu then
                        HR^^.HRText7Menu := HR^^.HRText7Menu + 1
                      else if temp^.from = palette then
                        HR^^.HRTex(7Palette := HR^^.HRTex(7Palette + 1
                    end
                  else if traverse^.next <> nil then
                    RemoveIPSelect
              end
            else if nextEvent^.theEvent = PressOK then
```

```
                    begin
                        HR^^.HRnumSize := HR^^.HRnumSize + 1;
                        if temp^.from = menu then
                            HR^^.HRText7Menu := HR^^.HRText7Menu + 1
                        else if temp^.from = palette then
                            HR^^.HRText7Palette := HR^^.HRText7Palette + 1;
                        if nextEvent^.next <> nil then
                            begin
                                deletePtr := nextEvent^.next;
                                AssignValues(WPEvents, nextEvent^.next);
                                DisposePtr(Ptr(deletePtr));
                            end
                        else
                            begin
                                DisposeHandle(Handle(WPEvents));
                                WPEvents := nil;
                            end;
                        DisposePtr(Ptr(nextEvent));
                    end
                else if nextEvent^.theEvent = PressCancel then
                    begin
                        if nextEvent^.next <> nil then
                            begin
                                deletePtr := nextEvent^.next;
                                AssignValues(WPEvents, nextEvent^.next);
                                DisposePtr(Ptr(deletePtr));
                            end
                        else
                            begin
                                DisposeHandle(Handle(WPEvents));
                                WPEvents := nil;
                            end;
                        DisposePtr(Ptr(nextEvent));
                    end
            end
        end;
end;

{ ------------------------------------------------------------------------------
Procedure HRHandleFontSet handles the first event = FontSet.  In this case, the help system
may be applying the "naive user" approach to setting the font (giving instructions).  This
procedure checks for the naive approach, and adjusts the user help profile accordingly.
------------------------------------------------------------------------------ }

procedure HRHandleFontSet;
    var
        nextEvent, temp, traverse, previous, deletePtr : WPERPtr;
begin
    nextEvent := WPEvents^^.next;
    temp := WPEvents^;
    if nextEvent <> nil then
        begin
            if not (TutorFontSet or MustTutorFont) then
                begin
                    HR^^.HRnumFont := HR^^.HRnumFont + 1;
                    if temp^.from = menu then
                        HR^^.HRText6Menu := HR^^.HRText6Menu + 1
                    else if temp^.from = palette then
                        HR^^.HRText6Palette := HR^^.HRText6Palette + 1;
                    if nextEvent^.theEvent = UserType then
                        begin
                            if nextEvent^.next <> nil then
                                begin
                                    deletePtr := nextEvent^.next;
                                    AssignValues(WPEvents, nextEvent^.next);
                                    DisposePtr(Ptr(deletePtr));
                                end
                            else
                                begin
                                    DisposeHandle(Handle(WPEvents));
                                    WPEvents := nil;
                                end;
                            DisposePtr(Ptr(nextEvent));
                        end
                    else
                        begin
                            if nextEvent = nil then
                                begin
                                    DisposeHandle(Handle(WPEvents));
                                    WPEvents := nil;
                                end
                            else
                                begin
                                    deletePtr := nextEvent;
                                    AssignValues(WPEvents, nextEvent);
                                    DisposePtr(Ptr(deletePtr));
                                end;
                        end;
                end
            else {TutorFontSet = TRUE}
                begin
                    if ((nextEvent^.theEvent = MoveIP) or (nextEvent^.theEvent = Select)) then
                        begin
                            TOC^^.IPMoved := TRUE;
                            TOC^^.EventSince := TRUE;
                            traverse := nextEvent;
                            previous := traverse;
                            while (traverse^.next <> nil) and (not ((traverse^.theEvent = Select) and (traverse^.from = menu))) and ((traverse^.theEvent =
MoveIP) or (traverse^.theEvent = Select)) do
                                begin
                                    previous := traverse;
                                    traverse := traverse^.next;
                                end;
                            if traverse <> nil then
                                if (traverse^.theEvent = PressOK) or ((previous^.theEvent = Select) and (previous^.from = menu)) then
                                    begin
                                        RemoveIPSelect;
                                        HR^^.HRnumFont := HR^^.HRnumFont + 1;
```

```
                        if temp^.from = menu then
                            HR^^.HRTextStMenu := HR^^.HRTextStMenu + 1
                        else if temp^.from = palette then
                            HR^^.HRTextStPalette := HR^^.HRTextStPalette + 1;
                    end
                else if traverse^.next <> nil then
                    RemoveIPSelect;
            end
        else if nextEvent^.theEvent = PressOK then
            begin
                HR^^.HRnumFont := HR^^.HRnumFont + 1;
                if temp^.from = menu then
                    HR^^.HRTextStMenu := HR^^.HRTextStMenu + 1
                else if temp^.from = palette then
                    HR^^.HRTextStPalette := HR^^.HRTextStPalette + 1;
                if nextEvent^.next <> nil then
                    begin
                        deletePtr := nextEvent^.next;
                        AssignValues(WPEvents, nextEvent^.next);
                        DisposePtr(Ptr(deletePtr))
                    end
                else
                    begin
                        DisposeHandle(Handle(WPEvents));
                        WPEvents := nil
                    end;
                DisposePtr(Ptr(nextEvent));
            end
        else if nextEvent^.theEvent = PressCancel then
            begin
                if nextEvent^.next <> nil then
                    begin
                        deletePtr := nextEvent^.next;
                        AssignValues(WPEvents, nextEvent^.next);
                        DisposePtr(Ptr(deletePtr));
                    end
                else
                    begin
                        DisposeHandle(Handle(WPEvents));
                        WPEvents := nil;
                    end;
                DisposePtr(Ptr(nextEvent));
            end
    end
end;

{ Procedure HRHandleStyleSet handles the first event = StyleSet.  In this case, the help
system may be applying the "naive user" approach to sensing the font (giving instructions).
This procedure checks for the naive approach, and adjusts the user help profile accordingly. }

procedure HRHandleStyleSet;
    var
        nextEvent, temp, traverse, previous, deletePtr: WPERPtr;
begin
    nextEvent := WPEvents^^.next;
    temp := WPEvents^;
    if nextEvent <> nil then
        begin
            if not (TutorFontSet or MustTutorFont) then
                begin
                    HR^^.HRnumStyle := HR^^.HRnumStyle + 1;
                    if temp^.from = menu then
                        HR^^.HRTextStMenu := HR^^.HRTextStMenu + 1
                    else if temp^.from = palette then
                        HR^^.HRTextStPalette := HR^^.HRTextStPalette + 1;
                    if nextEvent^.theEvent = UserType then
                        begin
                            if nextEvent^.next <> nil then
                                begin
                                    deletePtr := nextEvent^.next;
                                    AssignValues(WPEvents, nextEvent^.next);
                                    DisposePtr(Ptr(deletePtr));
                                end
                            else
                                begin
                                    DisposeHandle(Handle(WPEvents));
                                    WPEvents := nil
                                end;
                            DisposePtr(Ptr(nextEvent));
                        end
                    else
                        begin
                            if nextEvent = nil then
                                begin
                                    DisposeHandle(Handle(WPEvents));
                                    WPEvents := nil
                                end
                            else
                                begin
                                    deletePtr := nextEvent;
                                    AssignValues(WPEvents, nextEvent);
                                    DisposePtr(Ptr(deletePtr));
                                end;
                        end;
                end
            else {TutorFontSet or MustTutorFont = TRUE}
                begin
                    if (nextEvent^.theEvent = MoveIP) or (nextEvent^.theEvent = Select) then
                        begin
                            TOC^^.IPMoved := TRUE;
                            TOC^^.EventSince := TRUE;
                            traverse := nextEvent;
                            previous := traverse;
                            while (traverse^.next <> nil) and (not ((traverse^.theEvent = Select) and (traverse^.from = menu))) and ((traverse^.theEvent =
MoveIP) or (traverse^.theEvent = Select)) do
```

```
                    begin
                        previous := traverse;
                        traverse := traverse^.next;
                    end;
                if traverse <> nil then
                    if (traverse^.theEvent = PressOK) or ((previous^.theEvent = Select) and (previous^.from = menu)) then
                        begin
                            RemoveIPSelect;
                            HR^^.HRnumStyle := HR^^.HRnumStyle + 1;
                            if temp^.from = menu then
                                HR^^.HRText8Menu := HR^^.HRText8Menu + 1
                            else if temp^.from = palette then
                                HR^^.HRText8Palette := HR^^.HRText8Palette + 1;
                        end
                    else if traverse^.next <> nil then
                        RemoveIPSelect;
                end
            else if nextEvent^.theEvent = PressOK then
                begin
                    HR^^.HRnumStyle := HR^^.HRnumStyle + 1;
                    if temp^.from = menu then
                        HR^^.HRText8Menu := HR^^.HRText8Menu + 1
                    else if temp^.from = palette then
                        HR^^.HRText8Palette := HR^^.HRText8Palette + 1;
                    if nextEvent^.next <> nil then
                        begin
                            deletePtr := nextEvent^.next;
                            AssignValues(WPEvents, nextEvent^.next);
                            DisposPtr(Ptr(deletePtr));
                        end
                    else
                        begin
                            DisposHandle(Handle(WPEvents));
                            WPEvents := nil;
                        end;
                    DisposPtr(Ptr(nextEvent));
                end
            else if nextEvent^.theEvent = PressCancel then
                begin
                    if nextEvent^.next <> nil then
                        begin
                            deletePtr := nextEvent^.next;
                            AssignValues(WPEvents, nextEvent^.next);
                            DisposPtr(Ptr(deletePtr));
                        end
                    else
                        begin
                            DisposHandle(Handle(WPEvents));
                            WPEvents := nil;
                        end;
                    DisposPtr(Ptr(nextEvent));
                end
            end
        end;
end;

{ ..... }
{ Procedure HRHandleCut handles the first event = Cut }
{ ..... }

procedure HRHandleCut;
    var
        temp, nextEvent, traverse, previous, deletePtr : WPERPtr;
begin
    temp := WPEvents^;
    nextEvent := WPEvents^.next;
    if nextEvent <> nil then
        begin
            if ((nextEvent^.theEvent = MoveIP) or (nextEvent^.theEvent = Select)) then
                begin
                    traverse := nextEvent;
                    previous := traverse;
                    while (traverse^.next <> nil) and (not ((traverse^.theEvent = Select) and (traverse^.from = menu))) and ((traverse^.theEvent = MoveIP) or
(traverse^.theEvent = Select)) do
                        begin
                            previous := traverse;
                            traverse := traverse^.next;
                        end;
                    if (traverse^.theEvent = PressOK) or ((previous^.theEvent = Select) and (previous^.from = menu)) then
                        begin
                            HR^^.HRnumCut := HR^^.HRnumCut + 1;
                            if temp^.from = menu then
                                HR^^.HRCut2Menu := HR^^.HRCut2Menu + 1
                            else if temp^.from = key then
                                HR^^.HRCut2Key := HR^^.HRCut2Key + 1
                            else if temp^.from = palette then
                                HR^^.HRCut2Palette := HR^^.HRCut2Palette + 1;
                            TOC^^.method := CutTOC;
                            TOC^^.IPMoved := FALSE;
                            TOC^^.EventSince := FALSE;

                            RemoveIPSelect;
                        end
                    else if traverse^.next <> nil then
                        RemoveIPSelect;
                end
            else if (nextEvent^.theEvent = PressOK) or (nextEvent^.theEvent = PressCancel) then
                begin
                    if nextEvent^.next <> nil then
                        begin
                            deletePtr := nextEvent^.next;
                            AssignValues(WPEvents, nextEvent^.next);
                            DisposPtr(Ptr(deletePtr));
                        end
                    else
                        begin
                            DisposHandle(Handle(WPEvents));
                            WPEvents := nil;
```

```
                end;
              DisposePtr(Ptr(nextEvent));
            end
        end
    end;


    {-------------------------------------------------------------------------------}
    Procedure HRHandleCopy handles the first event = Copy.
    {-------------------------------------------------------------------------------}

    procedure HRHandleCopy;
      var
        temp, nextEvent, traverse, previous, deletePtr: WPERPtr;
    begin
      temp := WPEvents^;
      nextEvent := WPEvents^^.next;
      if nextEvent <> nil then
        begin
          if ((nextEvent^.theEvent = MoveIP) or (nextEvent^.theEvent = Select)) then
            begin
              traverse := nextEvent;
              previous := traverse;
              while (traverse^.next <> nil) and (not ((traverse^.theEvent = Select) and (traverse^.from = menu))) and ((traverse^.theEvent = MoveIP) or
traverse^.theEvent = Select)) do
                begin
                  previous := traverse;
                  traverse := traverse^.next;
                end;
              if (traverse^.theEvent = PressOK) or ((previous^.theEvent = Select) and (previous^.from = menu)) then
                begin
                  HR^^.HRnumCopy := HR^^.HRnumCopy + 1;
                  if temp^.from = menu then
                    HR^^.HRCopy2Menu := HR^^.HRCopy2Menu + 1
                  else if temp^.from = key then
                    HR^^.HRCopy2Key := HR^^.HRCopy2Key + 1
                  else if temp^.from = palette then
                    HR^^.HRCopy2Palette := HR^^.HRCopy2Palette + 1
                  RemoveIPBase;
                  TOC^^.method := CopyTOC;
                  TOC^^.IPMoved := FALSE;
                  TOC^^.eventSince := FALSE;
                end
              else if traverse^.next <> nil then
                RemoveIPSelect;
            end
          else if (nextEvent^.theEvent = PressOK) or (nextEvent^.theEvent = PressCancel) then
            begin
              if nextEvent^.next <> nil then
                begin
                  deletePtr := nextEvent^.next;
                  AssignValues(WPEvents, nextEvent^.next);
                  DisposePtr(Ptr(deletePtr));
                end
              else
                begin
                  DisposeHandle(Handle(WPEvents));
                  WPEvents := nil;
                end;
              DisposePtr(Ptr(nextEvent));
            end
        end
    end;


    {-------------------------------------------------------------------------------}
    Procedure HRHandleDelete handles the first event = Delete.
    {-------------------------------------------------------------------------------}

    procedure HRHandleDelete;
      var
        temp, nextEvent, nextNextEvent, traverse, previous, deletePtr: WPERPtr;
    begin
      temp := WPEvents^;
      nextEvent := WPEvents^^.next;
      if (TOC^^.method = CopyTOC) and (not TOC^^.EventSince) then
        begin
          TOC^^.method := CopyDelete;
          TOC^^.EventSince := TRUE;
          if nextEvent <> nil then
            begin
              deletePtr := nextEvent;
              AssignValues(WPEvents, nextEvent);
              DisposePtr(Ptr(deletePtr))
            end
          else
            begin
              DisposeHandle(Handle(WPEvents));
              WPEvents := nil
            end
        end
      else if nextEvent <> nil then
        begin
          if (nextEvent^.theEvent = Select) or (nextEvent^.theEvent = MoveIP) then
            begin
              TOC^^.EventSince := TRUE;
              TOC^^.IPMoved := TRUE;
              traverse := nextEvent;
              previous := traverse;
              while (traverse^.next <> nil) and (not ((traverse^.theEvent = Select) and (traverse^.from = menu))) and ((traverse^.theEvent = MoveIP) or
traverse^.theEvent = Select)) do
                begin
                  previous := traverse;
                  traverse := traverse^.next;
                end;
              if (traverse^.theEvent = PressOK) or ((traverse^.theEvent = Select) and (traverse^.from = menu)) then
                begin
                  nextNextEvent := traverse^.next;
                  if nextNextEvent <> nil then
```

```
                    begin
                      if nextNextEvent^.theEvent = UserType then
                        begin
                          HR^^.HRnumReplace := HR^^.HRnumReplace + 1;
                          if temp^.from = menu then
                            HR^^.HRReplace4FromMenu := HR^^.HRReplace4FromMenu + 1
                          else
                            HR^^.HRReplace4FromPalette := HR^^.HRReplace4FromPalette + 1
                        end;
                      RemoveEPSelect;
                    end
                  else
                    begin
                      HR^^.HRnumDelete := HR^^.HRnumDelete + 1;
                      if temp^.from = menu then
                        HR^^.HRDelete4FromMenu := HR^^.HRDelete4FromMenu + 1
                      else
                        HR^^.HRDelete4FromPalette := HR^^.HRDelete4FromPalette + 1
                    end
                end
            end
          else if (nextEvent^.theEvent = PressOK) or (nextEvent^.theEvent = PressCancel) then
            begin
              if nextEvent^.next <> nil then
                begin
                  deletePtr := nextEvent^.next;
                  AssignValues(WPEvents, nextEvent^.next);
                  DisposPtr(Ptr(deletePtr));
                end
              else
                begin
                  DisposHandle(Handle(WPEvents));
                  WPEvents := nil;
                end;
              DisposPtr(Ptr(nextEvent));
            end
        end
end;


{Procedure HRHandleDeleteType handles the first event = DeleteType.  In this case, the user
could simply be deleting, or if the next event is typing, this could be a method of replacing
text}

procedure HRHandleDeleteType;
  var
    nextEvent, deletePtr: WPERPtr;
begin
  nextEvent := WPEvents^^.next;
  if nextEvent <> nil then
    begin
      if nextEvent^.theEvent = UserType then
        begin
          HR^^.HRnumReplace := HR^^.HRnumReplace + 1
          HR^^.HRReplace1 := HR^^.HRReplace1 + 1;
          if nextEvent^.next <> nil then
            begin
              deletePtr := nextEvent^.next;
              AssignValues(WPEvents, nextEvent^.next);
              DisposPtr(Ptr(deletePtr));
            end
          else
            begin
              DisposHandle(Handle(WPEvents));
              WPEvents := nil;
            end;
          DisposPtr(Ptr(nextEvent))
        end
      else  {user is simply deleting}
        begin
          if nextEvent = nil then
            begin
              DisposHandle(Handle(WPEvents));
              WPEvents := nil
            end
          else
            begin
              deletePtr := nextEvent;
              AssignValues(WPEvents, nextEvent);
              DisposPtr(Ptr(deletePtr));
            end;
        end;
    end;
  TOC^^.IPMoved := TRUE;
  TOC^^.EventSince := TRUE
end;


{Procedure HRHandleUserType handles the first event = UserType.  In this case, the user
could simply be typing, or if the user has just cut text, this could be a method for replacing
the text.}

procedure HRHandleUserType;
  var
    nextEvent, deletePtr: WPERPtr;
begin
  nextEvent := WPEvents^^.next;
  if ((TOC^^.Method = CutTOC) and (not TOC^^.EventSince)) then
    begin
      HR^^.HRnumReplace := HR^^.HRnumReplace + 1;
      HR^^.HRReplace6 := HR^^.HRReplace6 + 1;
      if nextEvent = nil then
        begin
          DisposHandle(Handle(WPEvents));
          WPEvents := nil;
```

```
            end
         else
            begin
               deletePtr := nextEvent;
               AssignValues(WPEvents, nextEvent);
               DisposPtr(Ptr(deletePtr));
            end
      end
   else if nextEvent <> nil then      {user is simply typing}
      begin
         deletePtr := nextEvent;
         AssignValues(WPEvents, nextEvent);
         DisposPtr(Ptr(deletePtr));
      end;
   TOC^^.IPMoved := TRUE;
   TOC^^.EventSince := TRUE
end;

{ Procedure HRHandleMoveIP handles the first event = MoveIP.  In this case, the user could
simply be moving the insertion point, or the user could be moving the insertion point to
delete.}

procedure HRHandleMoveIP;
   var
      nextEvent, deletePtr: WPERPtr;
begin
   nextEvent := WPEvents^^.next;
   if (nextEvent <> nil) then
      begin
         if nextEvent^.theEvent = DeleteType then
            begin
               HR^^.HRnumDelete := HR^^.HRnumDelete + 1;
               HR^^.HRDelete1 := HR^^.HRDelete1 + 1;
               if nextEvent^.next <> nil then
                  begin
                     deletePtr := nextEvent^.next;
                     AssignValues(WPEvents, nextEvent^.next);
                     DisposPtr(Ptr(deletePtr))
                  end
               else
                  begin
                     DisposHandle(Handle(WPEvents));
                     WPEvents := nil
                  end;
               DisposPtr(Ptr(nextEvent));
            end
         else  {the event following the MoveIP event is something other than DeleteType}
            begin
               deletePtr := nextEvent;
               AssignValues(WPEvents, nextEvent);
               DisposPtr(Ptr(deletePtr))
            end;
      end;
   TOC^^.IPMoved := TRUE;
   TOC^^.EventSince := TRUE
end;

{ Procedure EventToHelp is the help procedure which dispatches the event queue to the
appropriate handling procedure for potential incorporation into the user help profile.}

procedure EventToHelp;
begin
   case WPEvents^^.theEvent of
      SizeChange, StyleChange, FontsChange:
         begin
            RemoveToHR;
            TutorPaste := FALSE;
            TutorFontSet := FALSE;
         end;
      ClickDnArrow, ClickUpArrow, ClickDnArrowRepeat, ClickUpArrowRepeat, PressDnArrow, PressUpArrow, ScreenP, ScreenN, LineP, LineN,
DragThumb:
         RemoveToHR;
      Select:
         HRHandleSelect(FALSE);
      SizeSet:
         HRHandleSizeSet;
      FontSet:
         HRHandleFontSet;
      StyleSet:
         HRHandleStyleSet;
      PressOK, PressCancel:
         HRDeleteEvent(WPEvents^, nil);
      Cut:
         if WPEvents^^.textSelected then
            HRHandleSelect(TRUE)
         else
            HRHandleCut;
      Copy:
         if WPEvents^^.textSelected then
            HRHandleSelect(TRUE)
         else
            HRHandleCopy;
      Delete:
         if WPEvents^^.textSelected then
            HRHandleSelect(TRUE)
         else
            HRHandleDelete;
      Move:
         if WPEvents^^.textSelected then
            HRHandleSelect(TRUE)
         else
            HRHandleMove;
      Paste:
         HRHandlePaste;
```

```
        Replace:
          if WPEvents^^.textSelected then
            HRHandleSelect(TRUE)
          else
            HRHandleReplace;
        Duplicate:
          if WPEvents^^.textSelected then
            HRHandleSelect(TRUE)
          else
            HRHandleDuplicate;
        DeleteType:
          begin
            if WPEvents^^.textSelected then
              HRHandleSelect(TRUE)
            else
              HRHandleDeleteType;
          end;
        MoveIP:
          HRHandleMoveIP;
        UserType:
          if WPEvents^^.textSelected then
            HRHandleSelect(TRUE)
          else
            HRHandleUserType;
        otherwise
      end;
    end;

{ Procedure doEvtsHelp is the help procedure which cleans up the latest event, then sends it
  off to be written to the journal file and incorporated into the user help profile }

procedure doEvtsHelp;

  var
    temp: WPERPtr;
    str, theText: str255;
    disposFlag, dummy: boolean;

begin
  disposFlag := FALSE;

  if WPEvents = nil then
    temp := nil
  else
    temp := WPEvents^;

  if temp <> nil then
    while temp^.next <> nil do
      temp := temp^.next;

  if temp <> nil then
    begin
      if (WPER^^.theEvent = ClickUpArrow) and ((temp^.theEvents = ClickUpArrow) or (temp^.theEvent = ClickUpArrowRepeat)) then
        begin
          temp^.theEvent := ClickUpArrowRepeat;
          DisposHandle(Handle(WPER));
          WPER := nil
        end
      else if (WPER^^.theEvent = ClickDnArrow) and ((temp^.theEvent = ClickDnArrow) or (temp^.theEvent = ClickDnArrowRepeat)) then
        begin
          temp^.theEvent := ClickDnArrowRepeat;
          DisposHandle(Handle(WPER));
          WPER := nil
        end
      else if (WPER^^.theEvent = UserType) and (temp^.theEvent = UserType) then
        begin
          temp^.numChars := temp^.numChars + 1;
          DisposHandle(Handle(WPER));
          WPER := nil
        end
      else if (WPER^^.theEvent = DeleteType) and (temp^.theEvent = DeleteType) then
        begin
          temp^.numChars := temp^.numChars + 1
          DisposHandle(Handle(WPER));
          WPER := nil
        end
    end;

  if WPER <> nil then
    begin
      if (WPER^^.theEvent = UserType) or (WPER^^.theEvent = DeleteType) then
        WPER^^.numChars := 1;

      if (WPEvents = nil) then
        begin
          WPEvents := WPER;
          temp := WPER^
        end
      else
        begin
          temp^.next := WPERPtr(NewPtr(SizeOf(WPEventRecord)));
          AssignValuesBack(temp^.next, WPER);
          disposFlag := TRUE;
        end;
    end;

  if (temp <> nil) and (WPER <> nil) then
    if ((temp^.theEvent = userType) and (WPER^^.theEvent <> userType)) or ((temp^.theEvent = deleteType) and (WPER^^.theEvent <> deleteType)) then
      begin
        str := '';
        NumToString(temp^.numChars, str);
        theText := 'NumChars = ';
        TEInsert(@theText[1], 11, WPEventJournal);
        TEInsert(@str[1], 6, WPEventJournal);
```

```
            TEKey(CR, WPEventJournal);
            TEKey(CR, WPEventJournal)
        end;

    if WPER <> nil then
        begin
            EventToJournal;
            EventToHelp;
        end;

    if disposeFlag then
        begin
            DisposeHandle(Handle(WPER));
            WPER := nil
        end;

Reset booleans:}
    H_MenuKey := FALSE;
    H_Palette := FALSE;

    UnLoadSeg(@InitHelp);                    {Segment 2}
    UnLoadSeg(@HandleUpdate);                {Segment 3}
  end;

end.
```

```
{
Margaret Stone
Sc.M. Project, Brown University

This unit contains the low levels routines necessary for updating the scroll bars for a
given window and scrolling the text as needed. All of the routines take a handle to the
particular window information to operate on. This is done so that some other application
which uses windows other than the Edit windows supported by the editor may easily
update their windows, even if they are not the front window. This would be the case for a
language processor of some kind, for example.
}

unit ShowEdit;

{
Interface Section
}

interface

  uses
    EditorGlobals, QuickDraw, ToolMgr;

  procedure AdjVScrollMax (wInfo: WindowHandle);
  procedure AdjText (wInfo: WindowHandle);
  procedure ShowSelect (wInfo: WindowHandle);
  function LinesInText (wInfo: WindowHandle): integer;

{
Implementation Section
}

implementation

{
LinesInText is a kluge necessitated by a minor bug in TE. The bug is that if the last
character in the TE text buffer is a Carriage Return, the field "nLines" may be off by one,
and cause slightly incorrect updating of the window. This routine always returns the
correct number of lines in the buffer.
}

  function LinesInText;      {(wInfo: WindowHandle): integer;}
    var
      lines: integer;
      txt: CharsHandle;
      traverse: TextInfoPtr;
  begin
    lines := 0;
    traverse := EditWinInfo^^.theText;
    while traverse <> nil do
      begin
        lines := lines + traverse^.teh^^.nLines;
        txt := CharsHandle(traverse^.teh^^.hText);
        if traverse^.teh^^.teLength > 0 then
          if txt^^[traverse^.teh^^.teLength - 1] = CR then
            lines := lines + 1;
        traverse := traverse^.next[T];
      end;
    LinesInText := lines
  end;

{
AdjVScrollMax adjusts the "Max" control setting for a given vertical scroll bar. This is
simply the number of lines of text minus the number of lines that can be displayed in the
window.
}

  procedure AdjVScrollMax; {(wInfo : WindowHandle)}
    var
      ctlMax: integer;
  begin
    ctlMax := (TEGetHeight(LinesInText(EditWinInfo), 0, EditWinInfo^^.theText^.teHt) div EditWinInfo^^.height) * EditWinInfo^^.scrollUnits;
    if ctlMax < 1 then
      ctlMax := 0;
    SetCtlMax(EditWinInfo^^.vScrollBar, ctlMax);
  end;

{
AdjText scrolls the text vertically to reflect the current "value" of the scroll bar.
The scroll bar values are set elsewhere.
}

  procedure AdjText; {(wInfo : WindowHandle)}
    var
      oldScroll, newScroll, delta, avgHeight: integer;
  begin
    avgHeight := EditWinInfo^^.height div EditWinInfo^^.scrollUnits;
    oldScroll := EditWinInfo^^.theText^.teh^^.viewRect.top - EditWinInfo^^.theText^.teh^^.destRect.top;
    newScroll := GetCtlValue(EditWinInfo^^.vScrollBar) * avgHeight;
    delta := oldScroll - newScroll;
    if delta <> 0 then
      TEScroll(0, delta, EditWinInfo^^.theText^.teh);
  end;

{
DoShow adjusts the scroll bar "value" so that the start of the selection, or current
insertion point becomes the top one in the window, or at least appears in the window.
}

  procedure DoShow (wInfo: WindowHandle; pos: integer);
  begin
    SetCtlValue(EditWinInfo^^.vScrollBar, pos - EditWinInfo^^.scrollUnits div 2);
    AdjText(EditWinInfo);
  end;

{
ShowSelect adjusts the vertical scroll bar, and forces the insertion point, or top of the
```

```
{ current text selection to be displayed in the window. This routine is typically called after }
{ inserting a character into the text, or after an edit.                                       }

  procedure ShowSelect;  { (wHnd : WindowHandle) }
    var
      top, bottom, unitSize, pos, oldPos, selLine: integer;
      selPt: point;
  begin
    Adj(VScrollMax(EditwinHo));
    Adj(Text(EditwinHo));
    selLine := 0;

    while EditwinHo^^.theText^.len^^.selStart >= EditwinHo^^.theText^.len^^.lineStarts[selLine] do
      selLine := selLine + 1;
    oldPos := TEGetHeight(selLine, 0, EditwinHo^^.theText^.len);
    unitSize := EditwinHo^^.height div EditwinHo^^.scrollUnits;
    pos := oldPos div unitSize;
    top := GetCtlValue(EditwinHo^^.vScrollBar);
    bottom := top + EditwinHo^^.scrollUnits;
    if (pos - 1 < top) then
      DoShow(EditwinHo, pos)
    else if (pos + 1 >= bottom) then
      DoShow(EditwinHo, pos);
  end;
end.
```

```
{
Margaret Stone
Sc.M. Project, Brown University

This unit contains all code to manipulate edit records, plus some scrolling code.
}

unit EditRecords;

interface

    uses
        EditorGlobals, ShowEdit, Quickdraw, ToolIntf;


    function AutoScroll: boolean;
    procedure HandleScroll (scrollBar: ControlHandle; part: integer);

    procedure NewER (theWind: WindowHandle; dest: Rect; window: WindowPtr);
    procedure AdderEnd (theWind: WindowHandle);
    function CurrentER (theWind: WindowHandle): TextInfoPtr;

implementation

{
HandleScroll handles mouse-downs in the up-line, down-line, up-page, or down-page
regions of the horizontal or vertical scroll bar. We must first determine which scroll bar
the mouse was clicked in so that the proper page size is used. The value of the associated
control is then updated and the text is scrolled if necessary. This procedure is the same as
the actionProc "MyAction" which is described in the Control Manager section of "Inside
Macintosh".
}

    procedure HandleScroll;      { ( scrollBar : ControlHandle;part : integer)}
        var
            delta, pageSize: integer;
    begin
        if part <> 0 then
            pageSize := EditWinPtr^.scrollUnits;
        case part of
            inUpButton:
                begin
                    delta := -1;
                    WPER^^.theEvent := ClickUpArrow
                end;
            inDownButton:
                begin
                    delta := +1;
                    WPER^^.theEvent := ClickDnArrow
                end;
            inPageUp:
                begin
                    delta := -pageSize;
                    WPER^^.theEvent := ScreenP;
                    WPER^^.from := mouse
                end;
            inPageDown:
                begin
                    delta := +pageSize;
                    WPER^^.theEvent := ScreenN;
                    WPER^^.from := mouse
                end;
            otherwise
        end;
        HLock(Handle(scrollBar));
        SetCtlValue(scrollBar, GetCtlValue(scrollBar) + delta);
        AdjText(TheWinfo);
        HUnLock(Handle(scrollBar))
    end;

{
AutoScroll is the "CtlLoop" routine which is described in the TextEdit Programmer's
Guide in Inside Macintosh. It is installed in all Edit windows opened by the editor. It is
called repeatedly from the ToolBox, for as long as the user holds down the mouse button,
when the mouse was first clicked in the text of some window. The text in the window is
scrolled up or down depending upon whether the mouse is above or below the text.
}

    function AutoScroll; { : boolean }
        var
            mouse: Point;
            oldClip: RgnHandle;
    begin
        AutoScroll := TRUE;
        oldClip := NewRgn;
        GetClip(oldClip);
        ClipRect(TheWindow^.portRect);
        GetMouse(mouse);
        HLock(Handle(theWinfo));
        if mouse.v < theWinfo^^.theText^.ten^^.viewRect.top then
            begin
                HandleScroll(theWinfo^^.vScrollBar, inUpButton);
                WPER^^.theEvent := nullEvt
            end
        else if mouse.v > theWinfo^^.theText^.ten^^.viewRect.bottom then
            begin
                HandleScroll(theWinfo^^.vScrollBar, inDownButton);
                WPER^^.theEvent := nullEvt;
            end;
        SetClip(oldClip);
        DisposeRgn(oldClip);
        HUnLock(Handle(theWinfo));
    end;


{
NewER creates a new edit record in a new window, with the font, size, style (and eventually
```

```
{ spacing) set as described by the user before opening the window.  This is the first edit
  record of the window.  (It doesn't create the window - it is called when a new window is
  created.                                                                              }

procedure NewER;          {(theWind:WindowHandle;  dest: Rect; window:WindowPtr)}
   var
      intoF: FontInfo;
      height: Integer;
      text:  TextInfoPtr;
      tStyle:  TextStyle;
      tempTEH: TEHandle;

begin
   new(TheWind^^.theText);
   text := TheWind^^.theText;

   tempTEH := TEStyleNew(dest, dest);
   theWind^^.theText^.teh := tempTEH;
   tStyle.tsFont := FontNum;
   tStyle.tsFace := FontStyle;
   tStyle.tsSize := FontSize;
   TESetSelect(0, 0, theWind^^.theText^.teh);
   TESetStyle(doAll, tStyle, TRUE, theWind^^.theText^.teh);

   height := TEGetHeight(1, 3, TheWind^^.theText^.teh);
   if not ShowClip then
      begin
         theWind^^.height := shortHeight;
         theWind^^.scrollUnits := shortScrollUnits
      end
   else
      begin
         theWind^^.height := longHeight;
         theWind^^.scrollUnits := longScrollUnits;
      end;
   theWind^^.textLines := (dest.bottom - dest.top) div theWind^^.height;
   SetCtlMLoop(@AutoScroll, TheWind^^.theText^.teh);
   TheWind^^.theText^.IP := TRUE;
   TheWind^^.theText^.nextTI := nil;
end;

{ AddatEnd is a procedure to add a new edit record at the end of a list of edit records, for
  example when a user is typing along in Geneva, then changes to Chicago before typing any
  more.  It makes the new edit record the IP, or active, edit record                    }

procedure AddatEnd;       {(theWind:WindowHandle)}
   var
      oldText, newText:  TextInfoPtr;
      oldBounds, newBounds: Rect;
      intoF: FontInfo;
      height, numLines, newBottom, lines: Integer;
begin
   oldText := theWind^^.theText;
   if (oldText <> nil) then
      begin
         while (oldText^.nextTI <> nil) do
            oldText := oldText^.nextTI;
         oldText^.IP := FALSE;
         TEDeactivate(oldText^.teh);
         oldBounds := oldText^.teh^^.destRect;
         numLines := oldText^.teh^^.nLines;
         height := oldText^.lineHeight;
         newBottom := oldBounds.top + (numLines * height);
         SetRect(oldText^.teh^^.destRect, oldBounds.left, oldBounds.top, oldBounds.right, newBottom);
         SetRect(oldText^.teh^^.viewRect, oldBounds.left, oldBounds.top, oldBounds.right, newBottom);
         new(oldText^.nextTI);
         newText := oldText^.nextTI;
         TextFont(FontNum);
         TextFace(FontStyle);
         TextSize(FontSize);
         GetFontInfo(intoF);
         newBounds := oldBounds;
         newBounds.top := newBottom;
         with intoF do
            height := ascent + descent + leading;
         newText^.lineHeight := height;
         lines := (newBounds.bottom - newBounds.top) div newText^.lineHeight;
         newBounds.bottom := newBounds.top + lines * newText^.lineHeight;
         newText^.teh := TEStyleNew(newBounds, newBounds);
         SetCtlMLoop(@AutoScroll, newText^.teh);
         newText^.IP := TRUE;
         newText^.nextTI := nil;
         TESetSelect(0, 0, newText^.teh);
         TEActivate(newText^.teh)
      end
   else
      sysbeep(5);
end;

{ CurrentER returns the current edit record (the one with the IP in it).              }

function CurrentER;              (theWind : WindowHandle) : TextInfoPtr)
   var
      text:  TextInfoPtr;
begin
   text := theWind^^.theText;
   if (text <> nil) then
      while ((text^.IP <> TRUE) and (text^.nextTI <> nil)) do
         text := text^.nextTI;
   CurrentER := text;
end;
end.
```

```
{ Margaret Stone
  Sc.M. Project, Brown University

  This unit contains the code to change the font type and size used by all Editor windows.

unit ChangeFont;

{ Interface part

interface

  uses
    EditorGlobals, EditorHelp, EditRecords, QuickDraw, ToolIntf;

  procedure DoText (Item: integer);
  procedure FixFontMenu (new: boolean);
  function CheckToStyle (CheckStyle: CheckStyles): Style;
  function CheckToFont (CheckFont: integer): integer;
  function CheckToSize (CheckSize: integer): integer;

{ Implementation part

implementation

  CheckToStyle,CheckToFont, and CheckToSize all convert a menu item number into a font,size
  or style number.  These functions are used to convert the Check font,size, and style variables
  to the Font num, size, and style variables.

  function CheckToStyle  {(CheckStyle : CheckStyles)  Style}
    var
      theStyle: Style;
  begin
    theStyle := [];
    if not (PlainItem in CheckStyle) then
      begin
        if BoldItem in CheckStyle then
          theStyle := theStyle + [bold];
        if ItalicItem in CheckStyle then
          theStyle := theStyle + [italic];
        if UnderlineItem in CheckStyle then
          theStyle := theStyle + [underline];
        if OutlineItem in CheckStyle then
          theStyle := theStyle + [outline];
        if ShadowItem in CheckStyle then
          theStyle := theStyle + [shadow];
      end;
    CheckToStyle := theStyle
  end;

  function CheckToFont;  { ( CheckFont : integer )  integer }
    var
      theFont: integer;
  begin
    case CheckFont of
      ChicagoItem:
        theFont := systemFont;
      GenevaItem:
        theFont := geneva;
      MonacoItem:
        theFont := monaco;
      NewYorkItem:
        theFont := newYork;
      otherwise
    end;
    CheckToFont := theFont
  end;

  function CheckToSize;  {(CheckSize : integer)  integer}
    var
      theSize: integer;
  begin
    case CheckSize of
      NineItem:
        theSize := 9;
      TenItem:
        theSize := 10;
      TwelveItem:
        theSize := 12;
      FourteenItem:
        theSize := 14;
      EighteenItem:
        theSize := 18;
      TwentyfourItem
        theSize := 24;
      otherwise
    end;
    CheckToSize := theSize
  end;

{ DoFontChoice changes the global FontNum and CheckFont variables, and changes the text
  menu to reflect the choice.  It then changes the font within the text edit window

  procedure doFontChoice (item: integer);
    var
      tStyle, oldStyle: TextStyle;
      oldPort: windowPtr;
```

```
  begin
{get the old style:}
      oldStyle.tsFont := FontNum;
      oldStyle.tsFace := FontStyle;
      oldStyle.tsSize := FontSize;
      WPER^^.oldFont := FontNum;

{set the new font:}
      CheckItem(TextMenu, CheckFont, FALSE);
      CheckFont := Item;
      CheckItem(TextMenu, CheckFont, TRUE);
      FontNum := CheckToFont(CheckFont);
      GetPort(oldPort);
      SetPort(ComPtr);
      InvertRect(TheFontRect);
      if CheckFont = Geneva4em then
         begin
            InvertRect(GenevaRect);
            TheFontRect := GenevaRect
         end
      else if CheckFont = Chicago4em then
         begin
            InvertRect(ChicagoRect);
            TheFontRect := ChicagoRect
         end
      else if CheckFont = Monaco4em then
         begin
            InvertRect(MonacoRect);
            TheFontRect := MonacoRect
         end
      else if CheckFont = NewYork4em then
         begin
            InvertRect(NewYorkRect);
            TheFontRect := NewYorkRect
         end;
      SetPort(oldPort);

{Should get current text record here, but for now...}
{here's where you handle any edit record stuff:}
      tStyle.tsFont := FontNum;
      tStyle.tsFace := FontStyle;
      tStyle.tsSize := FontSize;
      WPER^^.newFont := FontNum;
      HLock(Handle(EditWindow));
      if EditWindow <> nil then
         TESetStyle(doFont, tStyle, TRUE, EditWindow^^.theText^^ left);
      HUnLock(Handle(EditWindow));
   end;

{ DoStyleChoice changes the global FontStyle and CheckStyle variables, and changes the text
  menu to reflect the choice.  It then changes the font face within the text edit window.
  Style is slightly more complicated than font and size, as it is a set.  A choice of Plain turns
  off the other choices, choosing something other than plain, when not chosen, turns it on;
  when chosen, it will turn it off.  If it was the only thing chosen, it then turns Plain on. }


   procedure doStyleChoice (item: integer);
      var
         current: integer;
         tStyle, oldStyle: TextStyle;
         oldPort: windowPtr;
   begin
{set the new style:}
      GetPort(oldPort);
      SetPort(ComPtr);
      WPER^^.oldStyle := FontStyle;
      if (item = PlainItem) then
         begin
            for current := BoldItem to ShadowItem do
               if current in CheckStyle then
                  begin
                     CheckItem(TextMenu, current, FALSE);
                     if (current = BoldItem) then
                        InvertRect(BoldRect)
                     else if (current = ItalicItem) then
                        InvertRect(ItalicRect)
                     else if (current = UnderlineItem) then
                        InvertRect(UnderlineRect)
                     else if (current = OutlineItem) then
                        InvertRect(OutlineRect)
                     else if (current = ShadowItem) then
                        InvertRect(ShadowRect)
                  end;
            CheckStyle := [PlainItem];
            CheckItem(TextMenu, PlainItem, TRUE);
            InvertRect(PlainRect);
         end
      else
         begin
            if (PlainItem in CheckStyle) then
               begin
                  CheckItem(TextMenu, PlainItem, FALSE);
                  InvertRect(PlainRect);
                  CheckStyle := []
               end;
            if (item in CheckStyle) then
               begin
                  CheckItem(TextMenu, item, FALSE);
                  CheckStyle := CheckStyle - [item];
                  if (item = BoldItem) then
                     InvertRect(BoldRect)
                  else if (item = ItalicItem) then
                     InvertRect(ItalicRect)
                  else if (item = UnderlineItem) then
                     InvertRect(UnderlineRect)
                  else if (item = OutlineItem) then
```

```
                InvertRect(OutlineRect)
            else if (item = ShadowItem) then
                InvertRect(ShadowRect);
            if CheckStyle = [] then
                begin
                    CheckStyle := [PlainItem];
                    CheckItem(TextMenu, PlainItem, TRUE);
                    InvertRect(PlainRect);
                end
        end
    else
        begin
            CheckItem(TextMenu, item, TRUE);
            CheckStyle := CheckStyle + [item];
            if (item = BoldItem) then
                InvertRect(BoldRect)
            else if (item = ItalicItem) then
                InvertRect(ItalicRect)
            else if (item = UnderlineItem) then
                InvertRect(UnderlineRect)
            else if (item = OutlineItem) then
                InvertRect(OutlineRect)
            else if (item = ShadowItem) then
                InvertRect(ShadowRect)
        end
    end;
FontStyle := CheckToStyle(CheckStyle);

{Should get current text record here, but for now ..}
{here's where you handle any edit record stuff!}
    tStyle.tsFont := FontNum;
    tStyle.tsFace := FontStyle;
    tStyle.tsSize := FontSize;
    WPER^^.NewStyle := FontStyle;
    HLock(Handle(EditWindow));
    if EditWindow <> nil then
        TESetStyle(doFace, tStyle, TRUE, EditWindow^^.theText^.len);
    HUnLock(Handle(EditWindow));
    SetPort(oldPort);
end;

{ doSizeChoice changes the global FontSize and CheckSize variables, and changes the text menu
  to reflect the choice.  It then changes the font size within the text edit window }

procedure doSizeChoice (item: integer);
    var
        tStyle, oldStyle: TextStyle;
        oldPort: WindowPtr;
begin
{get the old style!}
    oldStyle.tsFont := FontNum;
    oldStyle.tsFace := FontStyle;
    oldStyle.tsSize := FontSize;
    WPER^^.oldSize := FontSize;

{set the new size!}
    CheckItem(TextMenu, CheckSize, FALSE);
    CheckSize := item;
    CheckItem(TextMenu, CheckSize, TRUE);
    FontSize := CheckToSize(CheckSize);

    GetPort(oldPort);
    SetPort(ComPtr);
    InvertRect(TheSizeRect);
    if CheckSize = NineItem then
        begin
            InvertRect(NineRect);
            TheSizeRect := NineRect
        end
    else if CheckSize = TenItem then
        begin
            InvertRect(TenRect);
            TheSizeRect := TenRect
        end
    else if CheckSize = TwelveItem then
        begin
            InvertRect(TwelveRect);
            TheSizeRect := TwelveRect
        end
    else if CheckSize = FourteenItem then
        begin
            InvertRect(FourteenRect);
            TheSizeRect := FourteenRect
        end
    else if CheckSize = EighteenItem then
        begin
            InvertRect(EighteenRect);
            TheSizeRect := EighteenRect
        end
    else if CheckSize = TwentyFourItem then
        begin
            InvertRect(TwentyFourRect);
            TheSizeRect := TwentyFourRect
        end;
    SetPort(oldPort);

{Should get current text record here, but for now ..}
{here's where you handle any edit record stuff!}
    tStyle.tsFont := FontNum;
    tStyle.tsFace := FontStyle;
    tStyle.tsSize := FontSize;
    WPER^^.newSize := FontSize;
    HLock(Handle(EditWindow));
    if EditWindow <> nil then
        TESetStyle(doSize, tStyle, TRUE, EditWindow^^.theText^.len);
    HUnLock(Handle(EditWindow));
```

```
end;

{-------------------------------------------------------------------------------
DoText handles a choice from the Text menu.  In the case of font, size, or style, it passes the
choice to the appropriate procedure.
-------------------------------------------------------------------------------}

procedure DoText  (item : integer);
begin
   case item of
      Chicagoitem, Genevaitem, Monaceitem, NewYorkItem:
         begin
            doFontChoice(item);
            if (EditWinfo^^.theText^.teh^^.selStart <> EditWinfo^^.theText^.teh^^.selEnd) then
               WPER^^.theEvent := FontChange
            else
               WPER^^.theEvent := FontSet;
            if H_Palette then
               WPER^^.from := palette
            else
               WPER^^.from := menu;
         end;
      Plainitem, Bolditem, Italicsitem, Underlineitem, Outlineitem, Shadowitem:
         begin
            doStyleChoice(item);
            if (EditWinfo^^.theText^.teh^^.selStart <> EditWinfo^^.theText^.teh^^.selEnd) then
               WPER^^.theEvent := StyleChange
            else
               WPER^^.theEvent := StyleSet;
            if H_Palette then
               WPER^^.from := palette
            else
               WPER^^.from := menu
         end;
      Nineitem, Tenitem, Twelveitem, Fourteenitem, Eighteenitem, Twentyfouritem:
         begin
            doSizeChoice(item);
            if (EditWinfo^^.theText^.teh^^.selStart <> EditWinfo^^.theText^.teh^^.selEnd) then
               WPER^^.theEvent := SizeChange
            else
               WPER^^.theEvent := SizeSet;
            if H_Palette then
               WPER^^.from := palette
            else
               WPER^^.from := menu
         end;
      otherwise
   end;
   EditWinfo^^.textDirty := TRUE;
   doEvtoHelp
end;

{-------------------------------------------------------------------------------
The three functions StyleToCheck,FontToCheck, and SizeToCheck are used by FixFontsMenu.
They convert font,size, and style to menu items.
FixFontsMenu fixes the font menu to reflect the position of the insertion point.
-------------------------------------------------------------------------------}

function StyleToCheck (aStyle: Style): CheckStyles;
   var
      theStyle: CheckStyles;
begin
   theStyle := [];
   if aStyle = [] then
      theStyle := [Plainitem]
   else
      begin
         if bold in aStyle then
            theStyle := theStyle + [BoldItem];
         if italic in aStyle then
            theStyle := theStyle + [ItalicItem];
         if underline in aStyle then
            theStyle := theStyle + [UnderlineItem];
         if outline in aStyle then
            theStyle := theStyle + [OutlineItem];
         if shadow in aStyle then
            theStyle := theStyle + [ShadowItem];
      end;
   StyleToCheck := theStyle
end;

function FontToCheck (Font: integer): integer;
   var
      theFont: integer;
begin
   case Font of
      systemFont:
         theFont := Chicagoitem;
      geneva:
         theFont := Genevaitem;
      monaco:
         theFont := Monaceitem;
      newYork:
         theFont := NewYorkItem;
      otherwise
   end;
   FontToCheck := theFont
end;

function SizeToCheck (Size: integer): integer;
   var
      theSize: integer;
begin
   case Size of
      9:
         theSize := Nineitem;
      10:
         theSize := Tenitem;
```

```
        12:
          theSize := TwelveItem;
        14:
          theSize := FourteenItem;
        18:
          theSize := EighteenItem;
        24:
          theSize := TwentyFourItem;
        otherwise
      end;
      SizeToCheck := theSize
  end;

{-------------------------------------------------------------------------------
FixFontMenu only checks the font at the beginning of the selection.  If this is an insertion
point, then it's accurate.  However, if multiple styles span a selection, the style at the
beginning of the selection is displayed.
-------------------------------------------------------------------------------}

  procedure FixFontMenu;          {(new:boolean)}
    var
      offset, height, ascent, i: integer;
      style: TextStyle;
      oldPort: windowPtr;
      oldselStart, oldselEnd: integer;
  begin
{Get the style at the mouse click}
    HLock(Handle(EditWinInfo))
    if new then
      begin
        style.tsFont := Geneva;
        style.tsSize := 12;
        style.tsFace := [];
        oldselStart := EditWinInfo^^.theText^.teh^^.selStart;
        oldselEnd := EditWinInfo^^.theText^.teh^^.selEnd;
        TESetSelect(0, EditWinInfo^^.theText^.teh^^.teLength, EditWinInfo^^.theText^.teh);
        TESetStyle(doAll, style, TRUE, EditWinInfo^^.theText^.teh);
        TESetSelect(oldselStart, oldselEnd, EditWinInfo^^.theText^.teh);
      end
    else if (EditWinInfo^^.theText^.teh^^.selStart >= EditWinInfo^^.theText^.teh^^.teLength) then
      TEGetStyle(EditWinInfo^^.theText^.teh^^.teLength - 1, style, height, ascent, EditWinInfo^^.theText^.teh)
    else
      TEGetStyle(EditWinInfo^^.theText^.teh^^.selStart, style, height, ascent, EditWinInfo^^.theText^.teh);
    HUnLock(Handle(EditWinInfo));

{Change actual font, size, style}
    FontNum := style.tsFont;
    FontSize := style.tsSize;
    FontStyle := style.tsFace;

{Change menu and palette items}
    GetPort(oldPort);
    SetPort(CenPtr);
    CheckItem(TextMenu, CheckFont, FALSE);
    if CheckFont = Geneva then
      InvertRect(GenevaRect)
    else if CheckFont = Chicago then
      InvertRect(ChicagoRect)
    else if CheckFont = Monaco then
      InvertRect(MonacoRect)
    else if CheckFont = NewYork then
      InvertRect(NewYorkRect);
    CheckFont := FontToCheck(style.tsFont);
    CheckItem(TextMenu, CheckFont, TRUE);
    if CheckFont = Geneva then
      begin
        InvertRect(GenevaRect);
        TheFontRect := GenevaRect
      end
    else if CheckFont = Chicago then
      begin
        InvertRect(ChicagoRect);
        TheFontRect := ChicagoRect
      end
    else if CheckFont = Monaco then
      begin
        InvertRect(MonacoRect);
        TheFontRect := MonacoRect
      end
    else if CheckFont = NewYork then
      begin
        InvertRect(NewYorkRect);
        TheFontRect := NewYorkRect;
      end;


    CheckItem(TextMenu, CheckSize, FALSE);
    if CheckSize = Nine then
      InvertRect(NineRect)
    else if CheckSize = Ten then
      InvertRect(TenRect)
    else if CheckSize = Twelve then
      InvertRect(TwelveRect)
    else if CheckSize = Fourteen then
      InvertRect(FourteenRect)
    else if CheckSize = Eighteen then
      InvertRect(EighteenRect)
    else if CheckSize = TwentyFour then
      InvertRect(TwentyFourRect);

    CheckSize := SizeToCheck(style.tsSize);
    CheckItem(TextMenu, CheckSize, TRUE);

    if CheckSize = Nine then
      begin
        InvertRect(NineRect);
        TheSizeRect := NineRect
      end
```

```
    else if CheckSize = TenItem then
      begin
        InvertRect(TenRect);
        TheSizeRect := TenRect
      end
    else if CheckSize = TwelveItem then
      begin
        InvertRect(TwelveRect);
        TheSizeRect := TwelveRect
      end
    else if CheckSize = FourteenItem then
      begin
        InvertRect(FourteenRect);
        TheSizeRect := FourteenRect
      end
    else if CheckSize = EighteenItem then
      begin
        InvertRect(EighteenRect);
        TheSizeRect := EighteenRect
      end
    else if CheckSize = TwentyFourItem then
      begin
        InvertRect(TwentyFourRect);
        TheSizeRect := TwentyFourRect
      end;


    for i := PlainItem to ShadowItem do
      if (i in CheckStyle) then
        begin
          CheckItem(TextMenu, i, FALSE);
          if (i = PlainItem) then
            InvertRect(PlainRect)
          else if (i = BoldItem) then
            InvertRect(BoldRect)
          else if (i = ItalicItem) then
            InvertRect(ItalicRect)
          else if (i = UnderlineItem) then
            InvertRect(UnderlineRect)
          else if (i = OutlineItem) then
            InvertRect(OutlineRect)
          else if (i = ShadowItem) then
            InvertRect(ShadowRect)
        end;
    CheckStyle := StyleToCheck(style.tsFace);
    for i := PlainItem to ShadowItem do
      if (i in CheckStyle) then
        begin
          CheckItem(TextMenu, i, TRUE);
          if (i = PlainItem) then
            InvertRect(PlainRect)
          else if (i = BoldItem) then
            InvertRect(BoldRect)
          else if (i = ItalicItem) then
            InvertRect(ItalicRect)
          else if (i = UnderlineItem) then
            InvertRect(UnderlineRect)
          else if (i = OutlineItem) then
            InvertRect(OutlineRect)
          else if (i = ShadowItem) then
            InvertRect(ShadowRect)
        end;

    SetPort(oldPort);
  end;
end.
```

```
{ Margaret Stone
  Sc.M. Project, Brown University

  This unit contains all of the one time start up initialization code for the editor.}

unit EditorInit;

{ Interface Section }

interface

  uses
    EditorGlobals, EditorHelp, ChangeFont;

  procedure InitEditor;

{ Implementation Section }

implementation

{ Menu is a routine to add a menu and the associated menu items to the menu bar.}

  procedure doMenu (id: integer; var hndl: MenuHandle; title, items: string);
  begin
    hndl := NewMenu(id, title);
    AppendMenu(hndl, items);
    InsertMenu(hndl, 0)
  end;

{ InitMenuBar adds all of the items to the menu bar, then draws the menu bar.}

  procedure InitMenuBar;
    var
      items: string;
  begin
    doMenu(AppleMenuId, AppleMenu, Chr(20), 'About Text Editor...');
    AddResMenu(AppleMenu, 'DRVR');
    items := 'Changing my Document Appearance ...;Editing my Document ...;Selecting Text ...;Seeing What is Above or Below the Screen ...';
    doMenu(HelpMenuId, HelpMenu, 'Help', items);
    items := 'New;Open...;(-;Save;(-;Close Unsaved Text...;Quit';
    doMenu(FileMenuId, FileMenu, 'File', items);
    items := 'Cut/X;Copy/C;Paste/V;(-;Move;Replace;Delete;Duplicate;(-;Show Clipboard';
    doMenu(EditMenuId, EditMenu, 'Edit', items);
    items := 'Select All;Select a Word;Select a Sentence;Select a Paragraph;Select Some Text';
    doMenu(SelectMenuId, SelectMenu, 'Select', items);
    items := 'Next Line/L;Previous Line/K;(-;Next Screen/S;Previous Screen/A';
    doMenu(ViewMenuId, ViewMenu, 'View', items);
    items := 'Chicago;Geneva;Monaco;New York;(-;Plain Text;Bold-B;Italic-I;Underline-U;Outline-O;Shadow-S;(-;9 point;10 point;12 point;14 point;18
point;24 point';
    doMenu(TextMenuId, TextMenu, 'Set Type Style', items);
    DrawMenuBar;
  end;

{ InitBitMaps initializes the bitmaps used in the control panel palette}

  procedure InitBitMaps;
  begin
    StuffHex(@DeleteImage, CONCAT('00200020', '00200020', '07BC71E0', '08A28A20', '7FFFFFFC', '09A28220', '06BC79E0'));
    with DeleteMap do
      begin
        baseAddr := @DeleteImage;
        rowBytes := 4;
        SetRect(bounds, 0, 0, 32, 7);
      end;
    StuffHex(@ReplaceImage, CONCAT('003FF800', '01C00700', '060000C0', '09145F20', '10A44210', '10444410', '10A44810', '0913DF20',
'60040C0', '01C38700', '003FF800', '00000000', '00010000', '00010000', '00010000', '00010000', '00010000', '0007C000', '00038000'
'00010000', '00200020', '00200020', '07BC71E0', '08A28A20', '7FFFFFFC', '09A28220', '06BC79E0'));
    with ReplaceMap do
      begin
        baseAddr := @ReplaceImage;
        rowBytes := 4;
        SetRect(bounds, 0, 0, 32, 27);
      end;
    StuffHex(@CutImage, CONCAT('00000001FC00', '380000030600', '0C70000030600', '40E0000070600', '381C000FFC00', '0703801FF800'
'0E0703C0000', '001C0E780000', '000381F80000', '000071180000', '00000ED00000', '000018DCFF00', '0000E03FFFC0', '000701FF81E0',
'0183E07C1E0', '00FFC000FFC0', '07E000007F00', '3C0000000000'));
    with CutMap do
      begin
        baseAddr := @CutImage;
        rowBytes := 8;
        SetRect(bounds, 0, 0, 44, 18);
      end;
    StuffHex(@MoveImageTop, CONCAT('001FFFFC', '00100004', '00100004', '00100004', '00100004', '00100004', '00100004', '00100004',
'00100004', '001FFFFC', '00040000', '00080000', '0FFF0000', '08040000', '08040000', '081FFFFC', '08100004', '08100004', '08100004', '08100004',
'08100004', '08100004', '081FFFFC', '08000000', '08000000', '081FFFFC', '08100004', '08100004', '08100004', '0FD00004', '00100004'));
    with MoveMapTop do
      begin
        baseAddr := @MoveImageTop;
        rowBytes := 4;
        SetRect(bounds, 0, 0, 32, 30);
      end;
    StuffHex(@MoveImageBottom, CONCAT('00100004', '00100004', '00100004', '001FFFFC'));
    with MoveMapBottom do
      begin
```

```
        baseAddr := @MoveImageBottom;
        rowBytes := 4;
        SetRect(bounds, 0, 0, 32, 4);
      end;
  end;

{.....}
  SetPaletteRects sets the rectangles for all of the QuickDraw and bitmap - drawn "controls" in the }
  control palette.                                                                                    }
{.....}

  procedure SetPaletteRects;
  begin
    SetRect(MoveRect, 0, 86, 43, 135);
    SetRect(ReplaceRect, 44, 86, 91, 135);
    SetRect(DeleteRect, 0, 136, 43, 169);
    SetRect(CutRect, 44, 136, 91, 169);
    SetRect(CopyToTextRect, 0, 170, 91, 187);
    SetRect(TextToClipRect, 0, 188, 91, 205);
    SetRect(ChangeToRect, 0, 206, 91, 225);
    SetRect(ChicagoRect, 0, 226, 91, 245);
    SetRect(GenevaRect, 0, 246, 91, 261);
    SetRect(NewYorkRect, 0, 262, 91, 279);
    SetRect(MonacoRect, 0, 280, 91, 297);
    SetRect(PlainRect, 0, 299, 91, 316);
    SetRect(BoldRect, 0, 317, 91, 331);
    SetRect(ItalicRect, 0, 332, 91, 344);
    SetRect(UnderlineRect, 0, 345, 91, 362);
    SetRect(OutlineRect, 0, 363, 91, 378);
    SetRect(ShadowRect, 0, 379, 91, 397);
    SetRect(NineRect, 0, 399, 43, 418);
    SetRect(TenRect, 44, 399, 91, 416);
    SetRect(TwelveRect, 0, 416, 43, 433);
    SetRect(FourteenRect, 44, 416, 91, 433);
    SetRect(EighteenRect, 0, 434, 43, 450);
    SetRect(TwentyFourRect, 44, 434, 91, 450);
  end;

{.....}
  initEditor is the main editor initialization code. The menu bar is drawn, rectangles, global booleans, }
  the default fonts, and other variables are initialized.                                                  }
{.....}

  procedure initEditor;

    var
      oldPort: GrafPtr;
      JournalDest, JournalView: Rect;
      JournalHeader: Str255;

  begin
    initMenuBar;
    initBitMaps;

{initialize fonts: }
    CheckFont := GenevaItem;
    CheckSize := TwelveItem;
    CheckStyle := (PlainItem);
    FontNum := CheckToFont(CheckFont);
    FontSize := CheckToSize(CheckSize);
    FontStyle := CheckToStyle(CheckStyle);
    CheckItem(TextMenu, GenevaItem, TRUE);
    CheckItem(TextMenu, TwelveItem, TRUE);
    CheckItem(TextMenu, PlainItem, TRUE);
    TheFontRect := GenevaRect;
    TheSizeRect := TwelveRect;

{initialize window and palette rectangles: }
    SetRect(utRect, 3, 40, 540, 474);
    SetRect(utClipRect, 3, 40, 540, 404);
    SetRect(InstructionsRect, 3, 40, 540, 80);
    SetRect(utInstructionsRect, 3, 101, 540, 474);
    SetRect(utInstClipRect, 3, 101, 540, 404);
    SetRect(ClipBdRect, 3, 424, 540, 474);
    SetRect(ContRect, 544, 24, 635, 474);
    SetRect(OkRect, 8, 6, 86, 26);
    SetRect(CancelRect, 6, 31, 86, 51);
    SetRect(HelpRect, 6, 56, 86, 76);
    SetPaletteRects;

    ClipBdWindow := @ClipBdStorage;

{Global booleans: }
    ShowClip := TRUE;
    ShowInstructions := FALSE;
    Done := FALSE;
    MessageDone := FALSE;
    OkPress := FALSE;
    MessageBar := FALSE;
    isOldMessage := FALSE;
    isSelecting := FALSE;
    isEditing := FALSE;
    isReplacing := FALSE;
    isClicking := FALSE;
    SelectInstruct := FALSE;
    SelectChoice := FALSE;
    H_MenuKey := FALSE;
    H_Palette := FALSE;
    SelectMethod := Normal;

{Global integers: }
    OldScrapCount := 0;
    UntitledNum := 1;
    FirstClick := 0;
    FileCount := 0;
    KeyStrokes := 0;
    OldMessageLength := 0;
```

```
{Initialize variables for characters (printable and otherwise): }
      BS := Chr(8);
      Tab := Chr(9);
      CR := Chr(13);
      Space := Chr(32);
      ExclPt := Chr(33);
      Quote := Chr(34);
      SQuote := Chr(39);
      OpenParen := Chr(40);
      CloseParen := Chr(41);
      Comma := Chr(44);
      Period := Chr(46);
      Colon := Chr(58);
      Semi := Chr(59);
      QMark := Chr(63);
      OpenBrack := Chr(91);
      CloseBrack := Chr(93);
      OpenQuote := Chr(210);
      CloseQuote := Chr(211);
      OpenSQuote := Chr(212);
      CloseSQuote := Chr(213);

      SafetyHandle := NewHandle(1000);
      IBeam := GetCursor(IBeamCURS);
      Watch := GetCursor(WatchCURS);
      TheWindo := nil;
      TheWindow := nil;
      EditWindow := nil;
      WPEvents := nil;

{Set up the journalling edit records: }
      SetRect(JournalDest, 650, 40, 800, 475);
      SetRect(JournalView, 650, 40, 800, 475);
      JournalHeader := 'User Journal of Events';
      WPEventJournal := TENew(JournalDest, JournalView);
      HLock(Handle(WPEventJournal));
      TEInsert(@JournalHeader[1], 22, WPEventJournal);
      TEKey(CR, WPEventJournal);
      TEKey(CR, WPEventJournal);
      HUnLock(Handle(WPEventJournal));
      JnLoadSeg(@DoText);                    Segment 15
      JnLoadSeg(@doEvtoHelp);                Segment 6
   end;


end
```

```
{
  Margaret Stone
  Sc.M. Project, Brown University

This unit contains the main Event Loop code and the top level of the File I/O code, and the
instructional messaging code.
}

unit EditorTopLevel;

{
  Interface part.
}

interface

  uses
    EditorGlobals, EditorHelpInit, HelpFiling, EditorHelp, EditHelp, ShowEdit, ChangeFont, EditorUtilities, EditRecords, HelpMessages, QuickDraw, ToolIntf;

  procedure Editor;

{
  Implementation part.
}

implementation

  procedure DoQuit (saveOLDO: Integer);
  Forward;

  procedure DoLoop (ControlsShowing: boolean; leaveWindow: boolean);
  Forward;

  procedure HandleKey (leaveWindow: boolean);
  Forward;

  procedure HandleGrow (Wrect: Rect; TheWind: WindowPtr; TheWindWInfo: WriteHandle);
  Forward;

{
  INSTRUCTIONAL MESSAGING
}

{
  Procedure message draws the message and the OK and Cancel buttons.
}

  procedure message (text: Str255; length: integer; leaveWindow: boolean; RestoreInstructions: boolean);
    var
      dest, view: Rect;
      oldPort: windowPtr;
  begin
    if not Done then
      begin
{get the old message:}
        if not ShowInstructions then
          begin
            OldMessage := text;
            OldMessageLength := length;
            isOldMessage := FALSE
          end
        else
          isOldMessage := TRUE;

{add the message:}
        dest := InstructionsWInfo^^.theText^.teh^^.destRect;
        view := InstructionsWInfo^^.theText^.teh^^.viewRect;
        GetPort(oldPort);
        SetPort(InstructionsPtr);
        EraseRect(view);
        DisposHandle(InstructionsWInfo^^.theText^.teh^^.hText);
        TEDispose(InstructionsWInfo^^.theText^.teh);
        HLock(Handle(InstructionsWInfo));
        InstructionsWInfo^^.theText^.teh := TEStyleNew(dest, view);
        TESetSelect(0, 0, InstructionsWInfo^^.theText^.teh);
        TEInsert(@text[1], length, InstructionsWInfo^^.theText^.teh);
        HUnLock(Handle(InstructionsWInfo));
        SetPort(oldPort);

        ShowInstructions := TRUE;
        ShowWindow(InstructionsPtr);
        if EditWindow <> nil then
          if ShowClip then
            HandleGrow(ulInstructionsRect, EditWindow, EditWInfo)
          else
            HandleGrow(ulInstrClipRect, EditWindow, EditWInfo);
        SelectWindow(InstructionsPtr);

        FlashMenuBar(0);
        FlashMenuBar(0);
        SysBeep(5);

{add the Ok and Cancel buttons:}
        ShowControl(OkButton);
        ShowControl(CancelButton);

{wait for a response:}
        messageDone := FALSE;
        OkPress := FALSE;
        DoLoop(TRUE, leaveWindow);
        messageDone := FALSE;

{restore old message or hide controls:}
        if (isOldMessage and RestoreInstructions) and (not (SelectInstruct and SelectChoice and OkPress)) then
          begin
            dest := InstructionsWInfo^^.theText^.teh^^.destRect;
            view := InstructionsWInfo^^.theText^.teh^^.viewRect;
```

```
                GetPort(oldPort);
                SetPort(InstructionsPtr);
                DisposeHandle(InstructionsWindo^^.theText^.teh^^.hText);
                TEDispose(InstructionsWindo^^.theText^.teh);
                HLock(Handle(InstructionsWindo));
                InstructionsWindo^^.theText^.teh := TEStyleNew(dest, view);
                TESetSelect(0, 0, InstructionsWindo^^.theText^.teh);
                TEInsert(@OldMessage[1], OldMessage.Length, InstructionsWindo^^.theText^.teh);
                HUnLock(Handle(InstructionsWindo));
                SetPort(oldPort);
                isOldMessage := FALSE
              end
            else if (not LeaveWindow) or (not OkPress) then
              begin
                HideWindow(InstructionsPtr);
                if (EditWindow <> nil) and ShowClip then
                  HandleDraw(utRect, EditWindow, EditWinfo)
                else
                  HandleDraw(utClipRect, EditWindow, EditWinfo);
                ShowInstructions := FALSE;
                HideControl(OkButton);
                HideControl(CancelButton)
              end
        end
    end;

  procedure fixControls;
  begin
    ShowControl(OkButton);
    ShowControl(CancelButton);
  end;

{ Procedure doTutorFontSet is a procedure that is called if the user has not
  demonstrated the ability to choose an object before operation, when the user chooses a font,
  size, or style.

  procedure doTutorFontSet (item: integer);
    var
      tempSelect: longint;
  begin
    MustTutorFont := TRUE;
    isEditing := TRUE;
    if (EditWinfo <> nil) then
      if (EditWindo^^ theText <> nil) then
        begin
          SelectInstruct := TRUE;
          case item of
            Chicagoitem, Genevaitem, Monacoitem, NewYorkitem:
              begin
                WPER^^ theEvent := FontSet;
                if H_Palette then
                  WPER^^ from := palette
                else
                  WPER^^ from := menu;
                if EditWindo^^ theText^ teh^^ selStart = EditWinfo^^ theText^.teh^^.selEnd then
                  WPER^^ textSelected := FALSE
                else
                  WPER^^ textSelected := TRUE;
                doEvtoHelp;
                message('If you want to change text that is already typed, SELECT the text you want to change.  Otherwise, click where you want to
begin typing in the new typeface.  Click in the Okay button when you are done, or click in the Cancel button to Cancel.', 241, FALSE, TRUE);
              end;
            Plainitem, Boolditem, Italicitem, Underlineitem, Outlineitem, Shadowitem:
              begin
                WPER^^ theEvent := StyleSet;
                if H_Palette then
                  WPER^^.from := palette
                else
                  WPER^^ from := menu;
                if EditWindo^^ theText^ teh^^ selStart = EditWinfo^^ theText^.teh^^.selEnd then
                  WPER^^ textSelected := FALSE
                else
                  WPER^^ textSelected := TRUE;
                doEvtoHelp;
                message('If you want to change text that is already typed, SELECT the text you want to change.  Otherwise, click where you want to
begin typing in the new style.  Click in the Okay button when you are done, or click in the Cancel button to Cancel.', 238, FALSE, TRUE);
              end;
            Nineitem, Tenitem, Twelveitem, Fourteenitem, Eighteenitem, Twentyfouritem:
              begin
                WPER^^ theEvent := SizeSet;
                if H_Palette then
                  WPER^^ from := palette
                else
                  WPER^^ from := menu;
                if EditWindo^^ theText^.teh^^ selStart = EditWinfo^^ theText^ teh^^ selEnd then
                  WPER^^ textSelected := FALSE
                else
                  WPER^^ textSelected := TRUE;
                doEvtoHelp;
                message('If you want to change text that is already typed, SELECT the text you want to change.  Otherwise, click where you want to
begin typing in the new size.  Click in the Okay button when you are done, or click in the Cancel button to Cancel.', 237, FALSE, TRUE);
              end;
          end;
          SelectInstruct := FALSE;
          SelectChoice := FALSE;

          if not done then
            begin
              if OkPress then
                begin
                  DispatchText(item)
                end;
            end;
          tempSelect := EditWinfo^^ theText^.teh^^ selEnd;
          TESetSelect(tempSelect, tempSelect, EditWinfo^^ theText^.teh);
        end;
```

```
    isEditing := FALSE;
end;


{ ------------------------------------------------------------ }
{ RESIZING AND ERRORS                                          }
{ ------------------------------------------------------------ }

{ ------------------------------------------------------------ }
{ MyGrowZone is called by the Macintosh Memory Manager when no more free space is left }
{ in the User's zone. This constitutes a fatal error. The User is prompted to save any dirty }
{ files, and we exit the program. Note that the User may not cancel the file save operation. }
{ This procedure is installed as the GrowZone procedure in the "Editor" top level procedure. }
{ ------------------------------------------------------------ }

function MyGrowZone (cbNeeded: Size): Size;
begin
    DisposHandle(SafetyHandle);
    MyErrorAlert('Out of Memory. Click "Ok" to save files and quit.');
    DoQuit(SaveDlogLOG);
    ExitToShell
end;

{ ------------------------------------------------------------ }
{ HandleGrow grows the passed window when necessary.  The two parameters }
{ "hSize" and "vSize" are the new height and width of the window with respect to the }
{ top-left corner of the window. The window, scroll bars, and text viewRect of the window }
{ are first resized. We then validate any remaining uncovered portion of the window's text }
{ if the text was not scrolled. This makes for a much cleaner update. }
{ ------------------------------------------------------------ }

procedure HandleGrow;          {(WrectRect: TheWind: WindowPtr: TheWindWinfo: WinfoHandle);}
    var
        r: Rect;
        oldPort: GrafPtr;
        wp: WindowPeek;
        oldCtlVal, totalHeight, numLines, avgHeight, hSize, vSize: integer;
begin
    hSize := WrectRight - WrectLeft;
    vSize := WrectBottom - WrectTop;
    GetPort(oldPort);
    SetPort(TheWind);
    MoveWindow(TheWind, WrectLeft, Wrect.top, TRUE);
    SizeWindow(TheWind, hSize, vSize, TRUE);
    InvalRect(TheWind^.portRect);
    r := TheWindWinfo^^.theText^.tehn^^ viewRect;
    oldCtlVal := GetCtlValue(TheWindWinfo^^ vScrollBar);
    TheWindWinfo^^.theText^.tehn^^.viewRect.bottom := TheWind^.portRect.bottom - 2;
    TheWindWinfo^^.theText^.tehn^^.viewRect.right := TheWind^.portRect.right - SBarWidth - 4;
    numLines := LinesInText(TheWindWinfo);
    totalHeight := TEGetHeight(numLines, 0, TheWindWinfo^^.theText^.tehn);
    if not ShowClip then
        begin
            TheWindWinfo^^.height := shortHeight;
            TheWindWinfo^^.scrollUnits := shortScrollUnits;
        end
    else if not ShowClip and ShowInstructions then
        begin
            TheWindWinfo^^.height := shortestHeight;
            TheWindWinfo^^.scrollUnits := shortestScrollUnits;
        end
    else
        begin
            TheWindWinfo^^.height := longHeight;
            TheWindWinfo^^.scrollUnits := longScrollUnits;
        end;
    avgHeight := TheWindWinfo^^.scrollUnits;
    TheWindWinfo^^.textLines := (TheWindWinfo^^.theText^.tehn^^.viewRect.bottom - TheWindWinfo^^.theText^.tehn^^.viewRect.top) div avgHeight;
    HidePen;
    SizeControl(TheWindWinfo^^.vScrollBar, SBarWidth, (TheWind^.portRect.bottom + 1) - (TheWind^.portRect.top - 1));
    AdjVScrollBar(TheWindWinfo);
    AdjText(TheWindWinfo);
    ShowPen;
    if GetCtlValue(TheWindWinfo^^.vScrollBar) < oldCtlVal then
        begin
            if TheWindWinfo^^.theText^.tehn^^.viewRect.right < r.right then
                r.right := TheWindWinfo^^.theText^.tehn^^.viewRect.right;
            if TheWindWinfo^^.theText^.tehn^^.viewRect.bottom < r.bottom then
                r.bottom := TheWindWinfo^^.theText^.tehn^^.viewRect.bottom;
            ValidRect(r)
        end;
    SetPort(oldPort);
end;

{ ------------------------------------------------------------ }
{ FILE AND WINDOW OPENING                                      }
{ ------------------------------------------------------------ }


{ ------------------------------------------------------------ }
{ OpenWindow is a general routine used by the editor to open its windows. A window is }
{ opened using the WindowPtr provided. A WinfoHandle is created and initialized. The }
{ handle is stored in the window's WRefCon field. A horizontal and vertical scroll bar, and }
{ a TEHandle for the text of the window, are allocated and stored in the WinfoHandle. A }
{ ClikLoop routine is installed for the TEHandle. The WindType parameter is used to }
{ mark, and do various type specific initializations to the window. }
{ ------------------------------------------------------------ }

procedure OpenWindow (title, fName: string; vRef: integer; window: WindowPtr; bounds: Rect; wTyp: WindType);
    var
        r: Rect;
        winfo: WinfoHandle;
        finfo: FontInfo;
        tStyle: TextStyle;
begin
    SetCursor(Watch^^);
    window := NewWindow(Ptr(nil), bounds, title, FALSE, 0, WindowPtr(-1), FALSE, 0);
    winfo := WinfoHandle(NewHandle(SizeOf(Windowinfo)));
```

```
  SetWRefCon(window, Ord(winIo));
  SetPort(window);
  HLock(Handle(winIo));
  with window^.portRect, winIo^^ do
    begin
      wType := wTyp;
      if wType = wEdit then
        begin
          SetRect(r, right - SBarWidth + 1, -1, right + 1, bottom + 2);
          vScrollBar := NewControl(window, r, Null, TRUE, 0, 0, 0, ScrollBarProc, 0);
          SetRect(r, 4, 4, right - SBarWidth - 4, bottom - 2);
          hndlER(winIo, r, window);
        end
      else
        begin
          SetRect(r, 4, 4, right - 4, bottom - 2);
          vScrollBar := nil;
          hndlER(winIo, r, window)
        end;
      textDirty := FALSE;
      fileName := fName;
      vRefNum := vRef;
      if wTyp = wClipBd then                    {Clipboard window}
        begin
          ClipWinIo := winIo;
          ClipBdWindow := window;
        end
      else if wTyp = wInstructions then         {Instructions window}
        begin
          InstructionsWinIo := winIo;
          InstructionsPtr := window;
          TextFont(geneva);
          TextSize(9);
        end
      else if wTyp = wEdit then                 {Edit window}
        begin
          EditWindow := window;
          EditWinIo := winIo;
          if fName <> Null then
            begin
              TESetText(FilePtr, FileLength, theText^ ish);
              FixFontMenu(TRUE);
              Ad1VScrollMax(winIo);
              if fName <> title then            {Duplicate file, behaves like UntitledXX}
                fileName := Null;               {fileName is null to force SaveAs...}
            end;
          ShowWindow(window)
        end
    end;
  HUnlock(Handle(winIo));
  InWindow := FALSE;
  InitCursor
end;
```

```
{ OpenFile is the code for the "Open..." and "New" commands on the "File" menu. The "New"
command opens an empty window with an "Unsaved Text" name. The "Open" command
prompts the user to enter the name of a file to edit with the SFGetFile dialog. If the name
is the same as a file that is already open, the user is prompted to open the file with an
"UntitledXX" name. If ReadFile is successful, the text for the window is in the global
variable "FilePtr". Note that any windows that may be open are updated before
opening the new window.                                                              }

procedure OpenFile (newFile: boolean);
  var
    vRef: integer;
    title, fName: Str255;
begin
  if newFile then
    begin
      title := NewName;
      fName := Null;
      vRef := 0;
      windowOpen := TRUE;
    end
  else  {Open a file}
    begin
      title := Null;
      fName := StandardFile(StandardGet, Null, 'TEXT', vRef);
      if fName <> Null then
        begin
          windowOpen := TRUE;
          if not DuplicateFile(fName) then
            title := fName
          else
            title := DuplicateName(fName);
          if title <> Null then
            if not ReadFile(fName, vRef) then
              title := Null
            else
              while GetNextEvent(updateMask, Event) do
                HandleUpdate
        end;
    end;
  if title <> Null then
    if ShowClip and not ShowInstructions then
      OpenWindow(title, fName, vRef, @utStorage, utRect, wEdit)
    else if ShowClip and ShowInstructions then
      OpenWindow(title, fName, vRef, @utStorage, utInstClipRect, wEdit)
    else
      OpenWindow(title, fName, vRef, @utStorage, utClipRect, wEdit)
end;
```

```
{ SAVING                                                                             }
```

```
{SaveText is the code for the "Save" and "Save As..." commands on the "File" menu. The
"Save" command simply writes the file to the disk and resets various flags. The "Save As"
command puts up the SFPutFile dialog and prompts the user to enter a name to save the
file as. For "Save As", the window title and associated "Windows" menu item are changed.
SaveText returns TRUE as result if the file is actully saved, and FALSE otherwise.}

function SaveText (window: WindowPtr; winHo: WindoHandle; saveAs: boolean): boolean;
    var
        vRef: integer;
        fName: Str255;
begin
    SaveText := FALSE;
    vRef := winHo^^.vRefNum;
    fName := winHo^^.fileName;
    if saveAs then
        begin
            if fName = Null then
                GetWTitle(window, fName);
            while GetNextEvent(updateMask, Event) do
                HandleUpdate;
            fName := StandardFile(StandardPut, fName, 'TEXT', vRef);
        end;
    if fName <> Null then
        if WriteFile(fName, vRef, winHo^^.theText^.teHt) then
            begin
                SaveText := TRUE;
                winHo^^.textDirty := FALSE;
                winHo^^.fileName := fName;
                winHo^^.vRefNum := vRef;
                if saveAs then
                    ChangeFile(window, fName)
            end
end;

{SaveFile is called when the user attempts to QUit, or Close a file. The WindowInfo record
for a given file is passed to SaveFile, and the "textDirty" field is tested. If the text is
not dirty, the function returns with TRUE as value. Otherwise, the user is prompted with
a dialog to Save, Discard, or Cancel the save operation. In the case of quitting after
running out of memory, the only options are Save and Discard.}

function SaveFile (window: WindowPtr; winHo: WindoHandle; saveOp: string; saveOLOG: integer): boolean;
    const
        Ok = 1;
        Discard = 2;
        Cancel = 3;
    var
        title: Str255;
        item: integer;
        oldPort: GrafPtr;
        saveDialog: DialogPtr;
        saveStorage: DialogRecord;
begin
    SaveFile := TRUE;
    if winHo <> nil then
        if winHo^^.textDirty then
            begin
                InitCursor;
                while GetNextEvent(updateMask, Event) do
                    HandleUpdate;
                GetPort(oldPort);
                GetWTitle(window, title);
                ParamText(title, saveOp, Null, Null);
                saveDialog := GetNewDialog(saveOLOG, @saveStorage, WindowPtr(-1));
                SetPort(saveDialog);
                FrameDItem(saveDialog, Ok);
                ModalDialog(Nil, item);
                DisposDialog(saveDialog);
                SetPort(oldPort);
                inWindow := FALSE;
                case item of
                    Ok:
                        SaveFile := SaveText(window, winHo, winHo^^.fileName = Null);
                    Cancel:
                        SaveFile := FALSE;
                    otherwise
                end
            end
end;

{ QUITTING, CLOSING WINDOWS AND FILES }

{CloseMyWindow is used to close an Edit window. The window is closed, and the associated
data structures are Disposed of.}

procedure CloseMyWindow (window: WindowPtr);
    var
        winHo: WindoHandle;
begin
    winHo := WindoHandle(GetWRefCon(window));
    KillControls(window);
    CloseWindow(window);
    TEDispose(winHo^^.theText^.teHt);
    Dispose(winHo^^.theText);
    DisposHandle(Handle(winHo))
end;

{CloseFront is called when the "Close" command on the "File" menu is issued.
There are four possibilities. (1) The window is a
Desk Accessory, in which case it is closed. (2) The window is a U.I. window, in which case
```

```
It is hidden. (3) The window is the Clipboard, in which case it is hidden.
(4) The window is an Edit window, in which case an attempt is made to save
the associated file, and if successful, the window is closed.

procedure CloseFront;
   var
      window: WindowPeek;

begin
   if FrontWindow <> TheWindow then
      begin
         window := WindowPeek(FrontWindow);
         with window^ do
            if windowKind < 0 then
               CloseDeskAcc(windowKind)                    {Desk Accessory}
            else if TheWrite^^.wType <> wPalette then
               HideWindow(WindowPtr(window))               {U.I. window}
      end
   else if TheWrite^^.wType = wClipBd then                 {Clipboard}
      begin
         HideWindow(TheWindow);
         ShowClip := TRUE;
         if EditWindow <> nil then
            HandleGrowth(theRect, EditWindow, EditWrite);
         SetItem(EditMenu, ClipBdItem, 'Show Clipboard');
      end
   else if SaveFile(TheWindow, TheWrite, CloseOp, SaveDiscCancDLOG) then
      begin
         DeallocFile(TheWindow);
         CloseMyWindow(TheWindow);
         TheWindow := nil;
         TheWrite := nil;
         EditWindow := nil;
         WindowOpen := FALSE;
      end
end;

DoQuit is called from the "Quit" command on the "File" menu, and from the GrowZone
procedure if the program runs out of memory. An attempt is made to save all open files,
and if successful, the global variable "Done" is set to TRUE.

procedure DoQuit; { (saveDLOG : integer) }
   var
      closing, dummy: boolean;
      write: WriteHandle;
      window: WindowPeek;
begin
   closing := TRUE;
   dummy := WriteFile('Journal', 0, WPEventJournal);
   window := WindowPeek(FrontWindow);
   DumpProfile;
   while closing and (window <> nil) do
      begin
         if window^.windowKind = userKind then
            begin
               write := WriteHandle(GetWRefCon(WindowPtr(window)));
               closing := SaveFile(WindowPtr(window), write, QuitOp, saveDLOG)
            end;
         window := window^.nextWindow
      end;
   Done := closing
end;

MENU HANDLING

DoSelect is the code for the Select menu.

procedure DoSelect (item: integer; leaveWindow: boolean);
   var
      OldStart, OldEnd: integer;
begin
   if EditWindow <> nil then
      begin
         OldStart := EditWrite^^.theText^.len^^.selStart;
         OldEnd := EditWrite^^.theText^.len^^.selEnd;
         isSelecting := TRUE;
         SelectChoice := TRUE;
         case item of
            SelAll:item:
               begin
                  TESetSelect(0, EditWrite^^.theText^.len^^.teLength, EditWrite^^.theText^.len);
                  WPER^^.theEvent := Select;
                  WPER^^.from := menu;
                  WPER^^.textSelected := TRUE;
                  doEvtHelp;
               end;
            SelSentence:item
               begin
                  SelectMethod := sentence;
                  message('Click in the sentence to be selected, then click in the OK button (or click in the Cancel button to cancel).', 108, leaveWindow,
TRUE);
                  if not Done then
                     if not OkPress then
                        TESetSelect(OldStart, OldEnd, EditWrite^^.theText^.len)
                     else
                        begin
                           WPER^^.theEvent := Select;
                           WPER^^.from := menu;
                           WPER^^.textSelected := TRUE;
                           doEvtHelp;
                        end
```

```
            end;
         SetWordItem:
            begin
               SelectMethod := word;
               message('Click in the word to be selected, then click in the OK button (or click in the Cancel button to cancel).', 104, leaveWindow,
TRUE);
               if not Done then
                  if not OkPress then
                     TESetSelect(OldStart, OldEnd, EditWinfo^^.theText^.teh)
                  else
                     begin
                        WPER^^.theEvent := Select;
                        WPER^^.from := menu;
                        WPER^^.textSelected := TRUE;
                        doEvttoHelp;
                     end;
            end;
         SetParaItem:
            begin
               SelectMethod := Paragraph;
               message('Click in the paragraph to be selected, then click in the OK button (or click in the Cancel button to cancel).', 109, leaveWindow,
TRUE);
               if not Done then
                  if not OkPress then
                     TESetSelect(OldStart, OldEnd, EditWinfo^^.theText^.teh)
                  else
                     begin
                        WPER^^.theEvent := Select;
                        WPER^^.from := menu;
                        WPER^^.textSelected := TRUE;
                        doEvttoHelp;
                     end;
            end;
         SetOldItem:
            begin
               SelectMethod := UnusualStart;
               message('Click at the start of the text to be selected, then click in the OK button (or click in the Cancel button to cancel).', 117, TRUE,
TRUE);
               if not Done then
                  if OkPress then
                     begin
                        OldSelectStart := EditWinfo^^.theText^.teh^^.selStart;
                        SelectMethod := Unusual;
                        message('Click at the end of the text to be selected, then click in the OK button (or click in the Cancel button to cancel).', 115,
leaveWindow, FALSE);
                        if not Done then
                           if not OkPress then
                              TESetSelect(OldStart, OldEnd, EditWinfo^^.theText^.teh)
                           else
                              begin
                                 WPER^^.theEvent := Select;
                                 WPER^^.from := menu;
                                 WPER^^.textSelected := TRUE;
                                 doEvttoHelp;
                              end
                     end
                  else
                     TESetSelect(OldStart, OldEnd, EditWinfo^^.theText^.teh)
            end;
         otherwise

      end;
      SelectMethod := Normal;
      IsSelecting := FALSE;
   end;
end;
```

DoClip is the code for the Show/Hide Clipboard item of the Edit menu. It shows or hides the Clipboard, changes the Edit menu item, and resizes the Edit window, if any.

```
procedure DoClip;
begin
   if ShowClip then
      begin
         ShowClip := FALSE;
         ShowWindow(ClipBdWindow);
         if (EditWindow <> nil) and not ShowInstructions then
            HandleGrow(r,ClipRect, EditWindow, EditWinfo)
         else
            HandleGrow(r,InstructionsClipRect, EditWindow, EditWinfo);
         SelectWindow(ClipBdWindow);
         SetItem(EditMenu, ClipBdItem, 'Hide Clipboard');
      end
   else
      begin
         ShowClip := TRUE;
         SetItem(EditMenu, ClipBdItem, 'Show Clipboard');
         HideWindow(ClipBdWindow);
         if (EditWindow <> nil) and not ShowInstructions then
            HandleGrow(r,Rect, EditWindow, EditWinfo)
         else
            HandleGrow(r,InstructionsRect, EditWindow, EditWinfo);
      end;
   UpdateClipBd(TRUE)
end;
```

DoCut is the code for the Cut item of the Edit menu. It cuts the item (placing it on the desk scrap, with style information in place), then TEStylePastes the item to the application clipboard window.

```
procedure DoCut (current: TextInfoPtr);
   var
      desk, view: Rect;
```

```
        oldPort: windowPtr;
        noSelection: boolean;
        tempSelect: longint;
    begin
        if EditWinIo <> nil then
            if (EditWinIo^^.theText <> nil) then
                begin
                    noSelection := EditWinIo^^.theText^.teh^^.selStart = EditWinIo^^.theText^.teh^^.selEnd;
                    if noSelection then
                        begin
                            SelectInstruct := TRUE;
                            message('SELECT the text you want to Cut, and click in the Okay button (or click in the Cancel button to cancel).', 104, FALSE, TRUE);
                            while (EditWinIo^^.theText^.teh^^.selStart = EditWinIo^^.theText^.teh^^.selEnd) and OkPress do
                                begin
                                    sysbeep(5);
                                    message('You did not select any text. Use the SELECT MENU if you do not know how. Select the text you want to Cut, and click in
the Okay button (or click in the Cancel button to cancel).', 177, FALSE, TRUE);
                                end;
                            SelectInstruct := FALSE;
                            SelectChoice := FALSE;
                        end;

                    if not done then
                        begin
                            if (not noSelection) or OkPress then
                                begin
                                    OkPress := FALSE;
                                    dest := ClipWinIo^^.theText^.teh^^.destRect;
                                    view := ClipWinIo^^.theText^.teh^^.viewRect;
                                    EditWinIo^^.textDirty := TRUE;
                                    TECut(EditWinIo^^.theText^.teh);
                                    GetPort(oldPort);
                                    SetPort(ClipBdWindow);
                                    HLock(Handle(ClipWinIo));
                                    TEDispose(ClipWinIo^^.theText^.teh);
                                    ClipWinIo^^.theText^.teh := TEStyleNew(dest, view);
                                    TESetSelect(0, 0, ClipWinIo^^.theText^.teh);
                                    TEStylPaste(ClipWinIo^^.theText^.teh);
                                    HUnLock(Handle(ClipWinIo));
                                    SetPort(oldPort);
                                    ShowSelect(EditWinIo);
                                    SetWTitle(ClipBdWindow, 'Clipboard - Cut Text');
                                    SetItem(EditMenu, PasteItem, 'Paste Cut Text');
                                    UpdateClipBd(TRUE);

                                    if ShowClip then
                                        DoClip;
                                end
                            else
                                begin
                                    tempSelect := EditWinIo^^.theText^.teh^^.selEnd;
                                    TESetSelect(tempSelect, tempSelect, EditWinIo^^.theText^.teh);
                                end;
                        end;
                end;
    end;

{-------------------------------------------------------------------------------}
{ DoCopy is the code for the Copy item of the Edit menu.  It copies the item (placing it on the }
{ desk scrap, with style information in place), then TEStylePastes the item to the application }
{ clipboard window.                                                             }
{-------------------------------------------------------------------------------}

    procedure DoCopy (current: TextWinePtr);
        var
            dest, view: Rect;
            oldPort: windowPtr;
            noSelection: boolean;
            tempSelect: longint;
        begin
            if EditWinIo <> nil then
                if (EditWinIo^^.theText <> nil) then
                    begin
                        noSelection := EditWinIo^^.theText^.teh^^.selStart = EditWinIo^^.theText^.teh^^.selEnd;
                        if noSelection then
                            begin
                                SelectInstruct := TRUE;
                                message('SELECT the text you want to Copy, and click in the Okay button (or click in the Cancel button to cancel).', 105, FALSE, TRUE);
                                while (EditWinIo^^.theText^.teh^^.selStart = EditWinIo^^.theText^.teh^^.selEnd) and OkPress do
                                    begin
                                        sysbeep(5);
                                        message('You did not select any text. Use the SELECT MENU if you do not know how. Select the text you want to Copy, and click in
the Okay button (or click in the Cancel button to cancel).', 178, FALSE, TRUE);
                                    end;
                                SelectInstruct := FALSE;
                                SelectChoice := FALSE;
                            end;

                        if not done then
                            begin
                                if (not noSelection) or OkPress then
                                    begin
                                        OkPress := FALSE;
                                        dest := ClipWinIo^^.theText^.teh^^.destRect;
                                        view := ClipWinIo^^.theText^.teh^^.viewRect;
                                        TECopy(EditWinIo^^.theText^.teh);
                                        GetPort(oldPort);
                                        SetPort(ClipBdWindow);
                                        HLock(Handle(ClipWinIo));
                                        TEDispose(ClipWinIo^^.theText^.teh);
                                        ClipWinIo^^.theText^.teh := TEStyleNew(dest, view);
                                        TESetSelect(0, 0, ClipWinIo^^.theText^.teh);
                                        TEStylPaste(ClipWinIo^^.theText^.teh);
                                        HUnLock(Handle(ClipWinIo));
                                        SetPort(oldPort);
                                        ShowSelect(EditWinIo);
                                        SetWTitle(ClipBdWindow, 'Clipboard - Copied Text');
                                        SetItem(EditMenu, PasteItem, 'Paste Copied Text');
```

```
                    UpdateClipBd(TRUE);

                    if ShowClip then
                      DoClip;
                  end;
                tempSelect := EditWinfo^^.theText^.teh^^.selEnd;
                TESetSelect(tempSelect, tempSelect, EditWinfo^^.theText^.teh);
              end;
          end;
end;

{-------------------------------------------------------------------------------
Procedure doTutorPaste is a procedure that is called if the user has not
demonstrated the ability to choose an object before operation.
-------------------------------------------------------------------------------}

procedure doTutorPaste (current: TextInfoPtr);
  var
    tempSelect: longint;
begin
  MustTutorPaste := TRUE;
  if EditWinfo <> nil then
    if (EditWinfo^^.theText <> nil) then
      begin
        SelectInstruct := TRUE;
        message('Click where you want to paste, and click in the OK button (or click in the Cancel button to cancel).', 100, FALSE, TRUE);
        SelectInstruct := FALSE;
        SelectChoice := FALSE;

        if not done then
          begin
            if OkPress then
              begin
                OkPress := FALSE;
                EditWinfo^^.textDirty := TRUE;
                TEStylPaste(EditWinfo^^.theText^.teh);
                ShowSelect(EditWinfo);
              end;
            tempSelect := EditWinfo^^.theText^.teh^^.selEnd;
            TESetSelect(tempSelect, tempSelect, EditWinfo^^.theText^.teh);
          end;
      end;
end;

{-------------------------------------------------------------------------------
DoPaste is the code for the Paste item of the Edit menu.  It pastes the desk scrap contents to
the application window, at the insertion point.
-------------------------------------------------------------------------------}

procedure DoPaste (current: TextInfoPtr);
begin
  if EditWinfo <> nil then
    if (EditWinfo^^.theText <> nil) then
      begin
        EditWinfo^^.textDirty := TRUE;
        TEStylPaste(EditWinfo^^.theText^.teh);
        ShowSelect(EditWinfo)
      end;
end;

{-------------------------------------------------------------------------------
DoMove is the code for the Move item of the Edit menu.  It cuts the selection (placing it on the
clipboard), then pastes it in where the user clicks.
-------------------------------------------------------------------------------}

procedure DoMove (current: TextInfoPtr);
  var
    dest, view: Rect;
    oldPort: windowPtr;
    noSelection: boolean;
    tempSelect: longint;
begin
  if EditWinfo <> nil then
    if (EditWinfo^^.theText <> nil) then
      begin
        noSelection := EditWinfo^^.theText^.teh^^.selStart = EditWinfo^^.theText^.teh^^.selEnd;
        if noSelection then
          begin
            SelectInstruct := TRUE;
            message('SELECT the text you want to Move, then click in the Okay button (or click in the Cancel button to cancel).', 106, TRUE, TRUE);
            while (EditWinfo^^.theText^.teh^^.selStart = EditWinfo^^.theText^.teh^^.selEnd) and OkPress do
              begin
                sysbeep(5);
                message('You did not select any text.  Use the SELECT MENU if you do not know how.  Select the text you want to Move, and click
the Okay button (or click in the Cancel button to cancel).', 178, FALSE, TRUE);
              end;
            SelectInstruct := FALSE;
            SelectChoice := FALSE;
          end;

        if not done then
          begin
            if (not noSelection or OkPress then
              begin
                OkPress := FALSE;
                dest := ClipWinfo^^.theText^.teh^^.destRect;
                view := ClipWinfo^^.theText^.teh^^.viewRect;
                EditWinfo^^.textDirty := TRUE;
                TECut(EditWinfo^^.theText^.teh);
                GetPort(oldPort);
                SetPort(ClipEdWindow);
                HLock(Handle(ClipWinfo));
                TEDispose(ClipWinfo^^.theText^.teh);
                ClipWinfo^^.theText^.teh := TEStyNew(dest, view);
                TESetSelect(0, 0, ClipWinfo^^.theText^.teh);
                TEStylPaste(ClipWinfo^^.theText^.teh);
                HUnLock(Handle(ClipWinfo));
                SetPort(oldPort);
```

```
                    ShowSelect(EditWinHdl);
                    SetWTitle(ClipBdWindow, 'Clipboard -- Text to be Moved');
                    SetItem(EditMenu, PasteItem, 'Paste Text From Move');
                    UpdateClipBd(TRUE);

                    if ShowClip then
                       DoClip;

                    isEditing := FALSE;
                    isClicking := TRUE;
                    message('Click where you want the text to be, then click in the OK button (or click in the Cancel button to cancel).', 107, FALSE,
FALSE);
                    if not done then
                       if OkPress then
                          DoPaste(EditWinHdl^^.theText);
                    isClicking := FALSE;
                 end;
              tempSelect := EditWinHdl^^.theText^.hn^^.selEnd;
              TESetSelect(tempSelect, tempSelect, EditWinHdl^^.theText^.hn);
           end;
        end;
  end;

{ ------------------------------------------------------------------------------ }
{ DoReplace is the code for the Replace item of the Edit menu.  It replaces the selection with the }
{ desired text, using ordinary Macintosh text-editing techniques. }
{ ------------------------------------------------------------------------------ }

  procedure DoReplace (myEvent: TextEditPtr);
     var
        noSelection: boolean;
        oldText: CharsHandle;
        start, finish, i: integer;
        select: ptr;
        selection: packed array[0..32000] of char;
        tempSelect: longint;
  begin
     if EditWinHdl <> nil then
        if (EditWinHdl^^.theText <> nil) then
           begin
              noSelection := EditWinHdl^^.theText^.hn^^.selStart = EditWinHdl^^.theText^.hn^^.selEnd;
              if noSelection then
                 begin
                    SelectInstruct := TRUE;
                    message('SELECT the text you want to Replace, and click in the Okay button (or click in the Cancel button to cancel).', 108, TRUE,
TRUE);
                    while (EditWinHdl^^.theText^.hn^^.selStart = EditWinHdl^^.theText^.hn^^.selEnd) and OkPress do
                       begin
                          sysbeep(6);
                          message('You did not select any text.  Use the SELECT MENU if you do not know how.  Select the text you want to Move, and click
n the Okay button (or click in the Cancel button to cancel).', 141, FALSE, TRUE);
                       end;
                    SelectInstruct := FALSE;
                    SelectChoice := FALSE;
                 end;

              if not done then
                 if (not noSelection) or OkPress then
                    begin
                       OkPress := FALSE;
                       start := EditWinHdl^^.theText^.hn^^.selStart;
                       finish := EditWinHdl^^.theText^.hn^^.selEnd;
                       if (finish > EditWinHdl^^.theText^.hn^^.teLength) then
                          finish := EditWinHdl^^.theText^.hn^^.teLength;
                       oldText := TEGetText(EditWinHdl^^.theText^.hn);
                       for i := start to finish do
                          selection[i - start] := oldText^^[i];
                       KeyStrokes := 0;
                       isEditing := FALSE;
                       isReplacing := TRUE;
                       message('Type the new text, then click in the OK button (or click in the Cancel button to cancel).', 99, FALSE, FALSE);
                       isReplacing := FALSE;
                       if not done then
                          if not OkPress and (KeyStrokes > 0) then
                             begin
                                TESetSelect(start, start + KeyStrokes, EditWinHdl^^.theText^.hn);
                                TEKey(BS, EditWinHdl^^.theText^.hn);
                                TESetSelect(start, start, EditWinHdl^^.theText^.hn);
                                TEInsert(@selection, finish - start, EditWinHdl^^.theText^.hn);
                                EditWinHdl^^.textDirty := TRUE;
                             end
                    end;
              tempSelect := EditWinHdl^^.theText^.hn^^.selEnd;
              TESetSelect(tempSelect, tempSelect, EditWinHdl^^.theText^.hn);
           end;
  end;

{ ------------------------------------------------------------------------------ }
{ DoDelete is the code for the Delete item of the Edit menu.  It deletes the selection from the }
{ edit window, leaving the contents of the clipboard and desk scrap intact. }
{ ------------------------------------------------------------------------------ }

  procedure DoDelete (myEvent: TextEditPtr);
     var
        noSelection: boolean;
        tempSelect: longint;
  begin
     if EditWinHdl <> nil then
        if (EditWinHdl^^.theText <> nil) then
           begin
              noSelection := EditWinHdl^^.theText^.hn^^.selStart = EditWinHdl^^.theText^.hn^^.selEnd;
              if noSelection then
                 begin
                    SelectInstruct := TRUE;
                    message('SELECT the text you want to Delete, and click in the Okay button (or click in the Cancel button to cancel).', 107, FALSE,
TRUE);
                    while (EditWinHdl^^.theText^.hn^^.selStart = EditWinHdl^^.theText^.hn^^.selEnd) and OkPress do
                       begin
```

```
                        sysbeep(5);
                        message('You did not select any text. Use the SELECT MENU if you do not know how. Select the text you want to Delete, and click
in the Okay button (or click in the Cancel button to cancel).', 180, FALSE, TRUE);
                     end;
                  SelectInstruct := FALSE;
                  SelectChoice := FALSE;
               end;

            if not done then
               if (not noSelection) or OkPress then
                  begin
                     TEDelete(EditWinInfo^^.theText^.teh);
                     EditWinInfo^^.textDirty := TRUE;
                  end;
               tempSelect := EditWinInfo^^.theText^.teh^^.selEnd;
               TESetSelect(tempSelect, tempSelect, EditWinInfo^^.theText^.teh);
               OkPress := FALSE;
            end;
      end;


{ DoDuplicate is the code for the Duplicate item of the Edit menu.  It copies the item (placing it
  on the desk scrap, with style information in place), then TEStylePastes the item to the
  application clipboard window.  It then asks the user where to place the duplicate in the text,
  and pastes the duplicate in.                                                                    }

   procedure DoDuplicate (current: TextInfoPtr);
      var
         dest, view: Rect;
         oldPort: windowPtr;
         noSelection: boolean;
         tempSelect: longint;
   begin
      if EditWinIo <> nil then
         if (EditWinIo^^.theText <> nil) then
            begin
               noSelection := EditWinIo^^.theText^.teh^^.selStart = EditWinIo^^.theText^.teh^^.selEnd;
               if noSelection then
                  begin
                     SelectInstruct := TRUE;
                     message('SELECT the text you want to Duplicate, and click in the Okay button (or click in the Cancel button to cancel).', 110, TRUE,
TRUE);
                     while (EditWinIo^^.theText^.teh^^.selStart = EditWinIo^^.theText^.teh^^.selEnd) and OkPress do
                        begin
                           sysbeep(5);
                           message('You did not select any text. Use the SELECT MENU if you do not know how. Select the text you want to Duplicate, and
click in the Okay button (or click in the Cancel button to cancel).', 183, FALSE, TRUE);
                        end;
                     SelectInstruct := FALSE;
                     SelectChoice := FALSE;
                  end;

               if not done then
                  begin
                     if (not noSelection) or OkPress then
                        begin
                           OkPress := FALSE;
                           dest := ClipWinIo^^.theText^.teh^^.destRect;
                           view := ClipWinIo^^.theText^.teh^^.viewRect;
                           TECopy(EditWinIo^^.theText^.teh);
                           GetPort(oldPort);
                           SetPort(ClipBdWindow);
                           HLock(Handle(ClipWinIo));
                           TEDispose(ClipWinIo^^.theText^.teh);
                           ClipWinIo^^.theText^.teh := TEStyleNew(dest, view);
                           TESetSelect(0, 0, ClipWinIo^^.theText^.teh);
                           TEStylePaste(ClipWinIo^^.theText^.teh);
                           HUnlock(Handle(ClipWinIo));
                           SetPort(oldPort);
                           ShowSelect(EditWinIo);
                           SetWTitle(ClipBdWindow, 'Clipboard -- Duplicated Text');
                           SetItem(EditMenu, PasteItem, 'Paste Duplicated Text');
                           UpdateClipBd(TRUE);

                           if ShowClip then
                              DoClip;

                           gEditing := FALSE;
                           gClicking := TRUE;
                           message('Click where you want the duplicate to be, then click in the OK button (or click in the Cancel button to cancel).', 112,
FALSE, FALSE);
                           if not done then
                              if OkPress then
                                 DoPaste(EditWinIo^^.theText);
                           gClicking := FALSE;
                        end;
                     tempSelect := EditWinIo^^.theText^.teh^^.selEnd;
                     TESetSelect(tempSelect, tempSelect, EditWinIo^^.theText^.teh);
                  end;
            end;
      end;


{ DoEdit is the high-level dispatching routine for the Edit menu.                                 }

   procedure DoEdit (item: integer);
      var
         current: TextInfoPtr;
   begin
      current := CurrentER(EditWinIo);
      if not SystemEditItem(item) then
         begin
            gEditing := TRUE;
            if EditWinIo^^.theText^.teh^^.selStart <> EditWinIo^^.theText^.teh^^.selEnd then
               WPER^^.TextSelected := TRUE
```

```
        else
          WPER^^.TextSelected := FALSE;
        case item of
          CutItem:
            begin
              WPER^^.theEvent := cut;
              if H_MenuKey then
                WPER^^.from := key
              else if H_Palette then
                WPER^^.from := palette
              else
                WPER^^.from := menu;
              doEvtoHelp;
              DoCut(EditWInfo^^.theText);
            end;
          CopyItem:
            begin
              WPER^^.theEvent := copy;
              if H_MenuKey then
                WPER^^.from := key
              else if H_Palette then
                WPER^^.from := palette
              else
                WPER^^.from := menu;
              doEvtoHelp;
              DoCopy(EditWInfo^^.theText);
            end;
          PasteItem:
            begin
              WPER^^.theEvent := paste;
              if H_MenuKey then
                WPER^^.from := key
              else if H_Palette then
                WPER^^.from := palette
              else
                WPER^^.from := menu;
              doEvtoHelp;
              if TutorPaste or MustTutorPaste then
                DoTutorPaste(EditWInfo^^.theText)
              else
                DoPaste(EditWInfo^^.theText);
            end;
          ClipBdItem:
            DoClip;
          MoveItem:
            begin
              WPER^^.theEvent := move;
              if H_Palette then
                WPER^^.from := palette
              else
                WPER^^.from := menu;
              doEvtoHelp;
              DoMove(EditWInfo^^.theText);
            end;
          ReplaceItem:
            begin
              WPER^^.theEvent := replace;
              if H_Palette then
                WPER^^.from := palette
              else
                WPER^^.from := menu;
              doEvtoHelp;
              DoReplace(EditWInfo^^.theText);
            end;
          DeleteItem:
            begin
              WPER^^.theEvent := delete;
              if H_Palette then
                WPER^^.from := palette
              else
                WPER^^.from := menu;
              doEvtoHelp;
              DoDelete(EditWInfo^^.theText);
            end;
          DuplicateItem:
            begin
              WPER^^.theEvent := duplicate;
              WPER^^.from := menu;
              doEvtoHelp;
              DoDuplicate(EditWInfo^^.theText);
            end;
          otherwise

        end;
        tEditing := FALSE;
      end;
end;


{-----------------------------------------------------------------------}
{DoFile is the top level dispatch routine for the "File" menu.}
{-----------------------------------------------------------------------}

procedure DoFile (item: integer);
begin
  case item of
    NewItem:
      if not WindowOpen then
        begin
          OpenFile(TRUE);
          ForFontMenu(TRUE);
        end;
    OpenItem:
      if not WindowOpen then
        OpenFile(FALSE);
    CloseItem:
      CloseFront;
```

```
      SaveItem:
         if SaveText(TheWindow, EditWinHe, FALSE) then

      QuitItem:
         DoQuit(SaveDlogCancelLOGI);
      otherwise
   end
end;

{-----------------------------------------------------------------------------}
DoView is the routine to handle a choice from the "View" menu.
{-----------------------------------------------------------------------------}

procedure DoView (item: integer);
   var
      delta: integer;
begin
   if GetCtlMax(EditWinHe^^.vScrollBar) <> 0 then
      begin
         case item of
            NextLItem:
               begin
                  delta := +1;
                  WPER^^.theEvent := LineN;
                  if H_MenuKey then
                     WPER^^.item := key
                  else
                     WPER^^.item := menu;
                  doEvtleHelp;
               end;
            PrevLItem:
               begin
                  delta := -1;
                  WPER^^.theEvent := LineP;
                  if H_MenuKey then
                     WPER^^.item := key
                  else
                     WPER^^.item := menu;
                  doEvtleHelp;
               end;
            NextSItem:
               begin
                  delta := +EditWinHe^^.scrollUnits;
                  WPER^^.theEvent := ScreenN;
                  if H_MenuKey then
                     WPER^^.item := key
                  else
                     WPER^^.item := menu;
                  doEvtleHelp;
               end;
            PrevSItem:
               begin
                  delta := -EditWinHe^^.scrollUnits;
                  WPER^^.theEvent := ScreenP;
                  if H_MenuKey then
                     WPER^^.item := key
                  else
                     WPER^^.item := menu;
                  doEvtleHelp;
               end;
            otherwise
         end;
         SetCtlValue(EditWinHe^^ vScrollBar, GetCtlValue(EditWinHe^^ vScrollBar) + delta);
         AdjText(EditWinHe);
      end;
end;

{-----------------------------------------------------------------------------}
HandleMenu is the top level dispatch routine for menu commands. The item selected is
passed to the appropriate menu handler.
{-----------------------------------------------------------------------------}

procedure HandleMenu (doesWindow: boolean);
   var
      menuCode: longint;
begin
   TestFront;
   if EventWhat = MouseDown then
      menuCode := MenuSelect(EventWhere)
   else
      menuCode := MenuKey(Chr(EventMessage));
   case HiWord(menuCode) of
      AppleMenuId:
         DoApple(LoWord(menuCode));
      HelpMenuId:
         DoHelp(TRUE, LoWord(menuCode));
      FileMenuId:
         if (not isEditing) and (not isSelecting) then
            DoFile(LoWord(menuCode))
         else
            YouCantNotDoThat(TRUE, FALSE, FileMenuId);
      EditMenuId:
         if (not isEditing) and (not isSelecting) then
            DoEdit(LoWord(menuCode))
         else
            YouCantNotDoThat(TRUE, FALSE, EditMenuId);
      SelectMenuId:
         if not isSelecting then
            DoSelect(LoWord(menuCode), leaveWindow)
         else
            YouCantNotDoThat(TRUE, FALSE, SelectMenuId);
      ViewMenuId:
         DoView(LoWord(menuCode));
      TextMenuId:
         if (not isEditing) and (not isSelecting) then
            begin
               if (TutorFontSet and (EditWinHe^^.theText^.lsh^^.selStart = EditWinHe^^.theText^.lsh^^.selEnd)) or MustTutorFont then
                  doTutorFontSet(LoWord(menuCode))
```

```
            else
                DoText(LoWord(menuCode))
            end
        else
            YouCanNotDoThat(TRUE, FALSE, TextMenuId);
        otherwise
    end;
    HiliteMenu(0)
end;

{ ------------------------------------------------------------ }
{ WINDOW CONTENT                                               }
{ ------------------------------------------------------------ }

{ ------------------------------------------------------------ }
{ ScrollContent is the top level code for handling mouse-downs in a scroll bar. The code }
{ tests if the mouse-down was in the "thumb" or not. All parts of the control other than the }
{ thumb are handled in a uniform manner.                       }
{ ------------------------------------------------------------ }

procedure ScrollContent (scrollBar: ControlHandle; part: integer; where: Point);
    var
        theValue1, theValue2: integer;
begin
    if part <> inThumb then
        begin
            theValue1 := GetCtlValue(scrollBar);
            part := TrackControl(scrollBar, where, @HandleScroll);
            theValue2 := GetCtlValue(scrollBar);
            if ((theValue2 < theValue1 - 1) and (WPER^^.theEvent = ClickUpArrow)) then
                WPER^^.theEvent := PressUpArrow
            else if ((theValue2 > theValue1 + 1) and (WPER^^.theEvent = ClickDnArrow)) then
                WPER^^.theEvent := PressDnArrow;
        end
    else
        begin
            part := TrackControl(scrollBar, where, nil);
            Adj(Text(EditWindo));
            WPER^^.theEvent := DragThumb
        end;
    doEvtHelp;
end;

{ ------------------------------------------------------------ }
{ SelectMenuHandle performs selection as per the Select menu   }
{ ------------------------------------------------------------ }

procedure SelectMenuHandle (localWhere: point);
    var
        start, finish: integer;
        found: boolean;
begin
    found := FALSE;
    HLock(Handle(EditWindo));
    TEClick(localWhere, BitAnd(EventModifiers, ShiftKey) = ShiftKey, EditWindo^^.theText^.teh);
    case SelectMenuId of
{Sentence Selection}
        Sentence:
            begin
{Here we should check to see if user has selected, using ordinary Macintosh means, a sentence}
{And then not execute the if not found block of code}
                if not found then
                    begin
                        start := FindSentenceStart;
                        finish := FindSentenceFinish;
                        TESetSelect(start, finish, EditWindo^^.theText^.teh);
                    end
            end;

{Word Selection}
        Word:
            begin
{Here we should check to see if user has selected, using ordinary Macintosh means, a word}
{And then not execute the if not found block of code}
                if not found then
                    begin
                        start := FindWordStart;
                        finish := FindWordFinish;
                        TESetSelect(start, finish, EditWindo^^.theText^.teh);
                    end
            end;

{Paragraph Selection}
        Paragraph:
            begin
{Here we should check to see if user has selected, using ordinary Macintosh means, a paragraph}
{And then not execute the if not found block of code}
                if not found then
                    begin
                        start := FindParaStart;
                        finish := FindParaFinish;
                        TESetSelect(start, finish, EditWindo^^.theText^.teh);
                    end
            end;

{Odd Selection}
        Unusual:
            begin
                if EditWindo^^.theText^.teh^^.selStart > OddSelectStart then
                    TESetSelect(OddSelectStart, EditWindo^^.theText^.teh^^.selStart, EditWindo^^.theText^.teh)
                else
                    TESetSelect(EditWindo^^.theText^.teh^^.selStart, OddSelectStart, EditWindo^^.theText^.teh);
            end;
        otherwise

    end;
```

```
    HUnlock(Handle(EditWinHdl));
end;

{-------------------------------------------------------------------------------}
{ DoControlWContent is the procedure for handling a mouse-down in the control window. }
{-------------------------------------------------------------------------------}

procedure DoControlWContent (control: ControlHandle);
    var
        tempOld: handle;
        part: integer;
        current: TextEditePtr;
begin
    H_Palette := TRUE;
    if (control <> nil) then
        begin
            part := TrackControl(control, Event.where, nil);
            if part <> 0 then
                if control = OkButton then
                    begin
                        OkPress := TRUE;
                        messageDone := TRUE;
                        if not isSelecting then
                            begin
                                WPER^^.theEvent := PressOK;
                                if EditWinHdl^^.theText^.teh^^.selStart <> EditWinHdl^^.theText^.teh^^.selEnd then
                                    WPER^^.TextSelected := TRUE
                                else
                                    WPER^^.TextSelected := FALSE;
                                doEvtteHelp;
                            end
                    end
                else if control = CancelButton then
                    begin
                        OkPress := FALSE;
                        messageDone := TRUE;
                        if not isSelecting then
                            begin
                                WPER^^.theEvent := PressCancel;
                                if EditWinHdl^^.theText^.teh^^.selStart <> EditWinHdl^^.theText^.teh^^.selEnd then
                                    WPER^^.TextSelected := TRUE
                                else
                                    WPER^^.TextSelected := FALSE;
                                doEvtteHelp;
                            end;
                    end
                else if control = HelpButton then
                    DoHelpButton
        end
    else if (not isSelecting) and (not isEditing) and (not isClicking) and (not isReplacing) then
        begin
            current := CurrentEPtr(EditWinHdl);
            if PtInRect(Event.where, MoveRect) then
                begin
                    isEditing := TRUE;
                    InvertRect(MoveRect);
                    if EditWinHdl <> nil then
                        DoEdit(MoveItem);
                    InvertRect(MoveRect);
                end
            else if PtInRect(Event.where, ReplaceRect) then
                begin
                    isEditing := TRUE;
                    InvertRect(ReplaceRect);
                    if EditWinHdl <> nil then
                        DoEdit(ReplaceItem);
                    InvertRect(ReplaceRect);
                end
            else if PtInRect(Event.where, DeleteRect) then
                begin
                    isEditing := TRUE;
                    InvertRect(DeleteRect);
                    if EditWinHdl <> nil then
                        DoEdit(DeleteItem);
                    InvertRect(DeleteRect);
                end
            else if PtInRect(Event.where, CutRect) then
                begin
                    isEditing := TRUE;
                    InvertRect(CutRect);
                    if EditWinHdl <> nil then
                        DoEdit(CutItem);
                    InvertRect(CutRect);
                end
            else if PtInRect(Event.where, ClipToTextRect) then
                begin
                    isEditing := TRUE;
                    InvertRect(ClipToTextRect);
                    if EditWinHdl <> nil then
                        DoEdit(PasteItem);
                    InvertRect(ClipToTextRect);
                end
            else if PtInRect(Event.where, TextToClipRect) then
                begin
                    isEditing := TRUE;
                    InvertRect(TextToClipRect);
                    if EditWinHdl <> nil then
                        DoEdit(CopyItem);
                    InvertRect(TextToClipRect);
                end
            else if PtInRect(Event.where, ChangeToRect) then
                doChangeTo
            else if PtInRect(Event.where, GenevaRect) then
                if (TutorFontSet and (EditWinHdl^^.theText^.teh^^.selStart = EditWinHdl^^.theText^.teh^^.selEnd)) or MustTutorFont then
                    doTutorFontSet(GenevaItem)
                else
                    doText(GenevaItem)
            else if PtInRect(Event.where, ChicagoRect) then
```

```
            if (TutorFontSet and (EditWnds^^.theText^.teh^^.selStart = EditWnds^^.theText^.teh^^.selEnd)) or MustTutorFont then
               doTutorFontSet(Chicagoitem)
            else
               doText(Chicagoitem)
         else if PtInRect(Event.where, MonacoRect) then
            if (TutorFontSet and (EditWnds^^.theText^.teh^^.selStart = EditWnds^^.theText^.teh^^.selEnd)) or MustTutorFont then
               doTutorFontSet(Monacoitem)
            else
               doText(Monacoitem)
         else if PtInRect(Event.where, NewYorkRect) then
            if (TutorFontSet and (EditWnds^^.theText^.teh^^.selStart = EditWnds^^.theText^.teh^^.selEnd)) or MustTutorFont then
               doTutorFontSet(NewYorkItem)
            else
               doText(NewYorkItem)
         else if PtInRect(Event.where, NineRect) then
            if (TutorFontSet and (EditWnds^^.theText^.teh^^.selStart = EditWnds^^.theText^.teh^^.selEnd)) or MustTutorFont then
               doTutorFontSet(Nineitem)
            else
               doText(Nineitem)
         else if PtInRect(Event.where, TenRect) then
            if (TutorFontSet and (EditWnds^^.theText^.teh^^.selStart = EditWnds^^.theText^.teh^^.selEnd)) or MustTutorFont then
               doTutorFontSet(Tenitem)
            else
               doText(Tenitem)
         else if PtInRect(Event.where, TwelveRect) then
            if (TutorFontSet and (EditWnds^^.theText^.teh^^.selStart = EditWnds^^.theText^.teh^^.selEnd)) or MustTutorFont then
               doTutorFontSet(Twelveitem)
            else
               doText(Twelveitem)
         else if PtInRect(Event.where, FourteenRect) then
            if (TutorFontSet and (EditWnds^^.theText^.teh^^.selStart = EditWnds^^.theText^.teh^^.selEnd)) or MustTutorFont then
               doTutorFontSet(FourteenItem)
            else
               doText(Fourteenitem)
         else if PtInRect(Event.where, EighteenRect) then
            if (TutorFontSet and (EditWnds^^.theText^.teh^^.selStart = EditWnds^^.theText^.teh^^.selEnd)) or MustTutorFont then
               doTutorFontSet(Eighteenitem)
            else
               doText(Eighteenitem)
         else if PtInRect(Event.where, TwentyFourRect) then
            if (TutorFontSet and (EditWnds^^.theText^.teh^^.selStart = EditWnds^^.theText^.teh^^.selEnd)) or MustTutorFont then
               doTutorFontSet(TwentyFouritem)
            else
               doText(TwentyFouritem)
         else if PtInRect(Event.where, PlainRect) then
            if (TutorFontSet and (EditWnds^^.theText^.teh^^.selStart = EditWnds^^.theText^.teh^^.selEnd)) or MustTutorFont then
               doTutorFontSet(Plainitem)
            else
               doText(Plainitem)
         else if PtInRect(Event.where, BoldRect) then
            if (TutorFontSet and (EditWnds^^.theText^.teh^^.selStart = EditWnds^^.theText^.teh^^.selEnd)) or MustTutorFont then
               doTutorFontSet(Boldltem)
            else
               doText(Boldltem)
         else if PtInRect(Event.where, ItalicRect) then
            if (TutorFontSet and (EditWnds^^.theText^.teh^^.selStart = EditWnds^^.theText^.teh^^.selEnd)) or MustTutorFont then
               doTutorFontSet(Italicitem)
            else
               doText(Italicitem)
         else if PtInRect(Event.where, UnderlineRect) then
            if (TutorFontSet and (EditWnds^^.theText^.teh^^.selStart = EditWnds^^.theText^.teh^^.selEnd)) or MustTutorFont then
               doTutorFontSet(Underlineitem)
            else
               doText(Underlineitem)
         else if PtInRect(Event.where, OutlineRect) then
            if (TutorFontSet and (EditWnds^^.theText^.teh^^.selStart = EditWnds^^.theText^.teh^^.selEnd)) or MustTutorFont then
               doTutorFontSet(Outlineitem)
            else
               doText(Outlineitem)
         else if PtInRect(Event.where, ShadowRect) then
            if (TutorFontSet and (EditWnds^^.theText^.teh^^.selStart = EditWnds^^.theText^.teh^^.selEnd)) or MustTutorFont then
               doTutorFontSet(Shadowitem)
            else
               doText(Shadowitem);
         IsEditing := FALSE;
      end
   else
      YouCantNotDoThat(FALSE, FALSE, 0);
end;

{-----------------------------------------------------------------------------
HandleContent is the top level dispatching routine for mouse-downs in the text or scroll
bars of the current front window.  Note that mouse-downs in the text of non-Edit
windows are ignored.
-----------------------------------------------------------------------------}

procedure HandleContent (window: WindowPtr);
   var
      part: integer;
      control: ControlHandle;
      current: TextEditPtr;

begin
   GlobalToLocal(Event.where);
   part := FindControl(Event.where, window, control);
   if (window = EditWindow) and (not IsReplacing) then
      begin
         current := CurrentEP(EditWindow);
         if part <> 0 then
            ScrollContent(control, part, Event.where)
         else if EditText then
            with EditWnds^^ EditWnds^^.theText^ do
               if PtInRect(Event.where, teh^^.viewRect) then
                  begin
                     if (SelectMethod = Normal) then
                        with Event do
                           begin
                              TECBlink(where, BitAnd(modifiers, ShiftKey) = ShiftKey, teh);
```

```
                        if (BitAnd(theStore, ShiftKey) = ShiftKey) then
                            WPER^^.from := key
                        else
                            WPER^^.from := mouse;
                        if (EditWinds^^.theText^.teh^^.selStart = EditWinds^^.theText^.teh^^.selEnd) then
                            WPER^^.theEvent := MoveIP
                        else
                            WPER^^.theEvent := Select;
                        if not isSelecting then
                            doEvtsHelp;
                    end
                else
                    SelectMenuHandler(Event.where);
                PtrForMenu(FALSE)
            end
        end
    else if (window = ContPtr) then
        DoControlWContent(control);
end;

{ ********************************************************************** }
{ EVENT HANDLING                                                         }
{ ********************************************************************** }

{ ---------------------------------------------------------------------- }
{ HandleMouse is the top level routine for handling mouse-down events. If the mouse was }
{ clicked in the content or contents of an Editor created window that is not the current }
{ front window, the window is first activated as per Macintosh User Interface guide lines. }
{ ---------------------------------------------------------------------- }

procedure HandleMouse (leaveWindow: boolean);
    var
        part: integer;
        newSize: longint;
        window: WindowPtr;
begin
    part := FindWindow(Event.Where, window);
    case part of
        inMenuBar:
            if not isPressing and not isClicking then
                HandleMenu(leaveWindow);
        inSysWindow:
            SystemClick(Event, window);
        inDesk:
            ;
        otherwise
            if ((window <> FrontWindow) and (window <> nil)) then
                SelectWindow(window)
            else if (window <> nil) then
                HandleContent(window);
    end
end;

{ ---------------------------------------------------------------------- }
{ HandleKey  is the top level routine for handling Key events. We first test if the key is a }
{ menu key equivalent. If so, we dispatch to the menu handler. Otherwise, we must test if }
{ an Edit window is the current front window. If so, we test if  (A) the key is a printing }
{ character, a <Cr>, or a <BS>, and (B) there is enough room in the text buffer to insert the }
{ character. If so, the character is inserted and various flags are updated.              }
{ ---------------------------------------------------------------------- }

procedure HandleKey;        {(leaveWindow: boolean)}
    var
        ch: char;

begin
    Event.Message := BitAnd(Event.Message, $FF);
    if BitAnd(Event.Modifiers, CmdKey) = CmdKey then
        begin
            H_MenuKey := TRUE;
            HandleMenu(leaveWindow)
        end
    else if EditWinds = nil then
        SysBeep(5)
    else
        begin
            if EditWinds^^.theText <> nil then
                begin
                    ch := Chr(Event.Message);
                    if EditWinds^^.wType = wClipBd then
                        SysBeep(5)
                    else if (Ord(ch) < Ord(SP)) and (ch <> CR) and (ch <> BS) then
                        SysBeep(5)
                    else if (EditWinds^^.theText^.teh^^.teLength = MaxInt) and (ch <> BS) then
                        SysBeep(5)
                    else
                        begin
                            EditWinds^^.textDirty := TRUE;
                            if EditWinds^^.theText^.teh^^.selStart <> EditWinds^^.theText^.teh^^.selEnd then
                                WPER^^.TextSelected := TRUE
                            else
                                WPER^^.TextSelected := FALSE;
                            TEKey(ch, EditWinds^^.theText^.teh);
                            ShowSelect(EditWinds);
                            if (ch <> BS) then
                                begin
                                    KeyStrokes := KeyStrokes + 1;
                                    WPER^^.theEvent := UserType
                                end
                            else
                                begin
                                    KeyStrokes := KeyStrokes - 1;
                                    WPER^^.theEvent := DeleteType
                                end;
                            doEvtsHelp;
                        end
                end;
        end
```

```
end;

{-----------------------------------------------------------------------}
{ HandleActivate is the top level routine for handling Activate/Deactivate events. Various }
{ flags and menu items are set as appropriate. Notably, the global variable "EditText" is set }
{ to TRUE if the window being activated is an Edit window.              }
{-----------------------------------------------------------------------}

procedure HandleActivate;
   var
      wrnd: WindowHandle;
      window: WindowPtr;
      current: TextHndPtr;
begin
   InitCursor;
   inWindow := FALSE;
   window := WindowPtr(EventMessage);
   wrnd := WindowHandle(GetWRefCon(window));
   if wrnd^^.wType <> wPalette then
      current := CurrentER(wrnd);
   SetPort(window);
   if wrnd^^.wType <> wPalette then
      begin
         if BitAnd(EventModifiers, ActiveFlag) <> 0 then  {activate the window}
            begin
               TheWrnd := wrnd;
               TheWindow := window;
               EditText := wrnd^^.wType = wEdit;
               TEActivate(EditWrnd^^.theText^.teh);
               if wrnd^^.wType = wEdit then
                  HiLiteControl(wrnd^^.vScrollBar, Active);
            end

         else        {deactivate the window}
            begin
               TheWrnd := nil;
               TheWindow := nil;
               EditText := FALSE;
               if wrnd^^.wType = wEdit then
                  HiLiteControl(wrnd^^.vScrollBar, Inactive);
            end;
         if EditText then
            SetItem(FileMenu, CloseItem, 'Close Unsaved Text')
         else
            SetItem(FileMenu, CloseItem, 'Close Clipboard')
      end;
end;

{-----------------------------------------------------------------------}
{ HandleCursor changes the cursor to an I-Beam if it is over the text of an active Edit }
{ window. The insertion point in the text is also flashed on and off here. Otherwise, the }
{ cursor is changed to the Arrow. The global variable "inWindow" is used to keep track of }
{ the previous status of the cursor.                                    }
{-----------------------------------------------------------------------}

procedure HandleCursor;
   var
      mouse, Gmouse: Point;
      current: TextHndPtr;
      part: integer;
      wrndWind: WindowPtr;
begin
   GetMouse(mouse);
   Gmouse := mouse;
   LocalToGlobal(Gmouse);
   part := FindWindow(Gmouse, wrndWind);
   if wrndWind <> nil then
      SelectWindow(wrndWind)
   else
      InitCursor;
   if ((wrndWind = EditWindow) and (EditWindow <> nil)) then
      begin
         TEIdle(EditWrnd^^.theText^.teh);
         if PtInRect(mouse, EditWrnd^^.theText^.teh^^.viewRect) then
            begin
               if not inWindow then
                  begin
                     SetCursor(IBeam^^);
                     inWindow := TRUE
                  end
            end
         else if inWindow then
            begin
               InitCursor;
               inWindow := FALSE
            end
      end
end;

{-----------------------------------------------------------------------}
{ HIGH LEVEL EVENT HANDLING                                             }
{-----------------------------------------------------------------------}


procedure doLoop;  {( ControlsShowing:boolean; leaveWindow:boolean)
   var
      wrndOld: handle;
begin
   repeat
      if ControlsShowing then
         fixControls;
      SystemTask;
      HandleCursor;

      if (not WEselecting) or (WPER = nil) then
         begin
            WPER := WPERHandle(NewHandle(SIZEOF(WPEventRecord)));
            WPER^^.theEvent := nullEvt;
```

```
            WPER^^.front := nuSFront;
            WPER^^.textSelected := FALSE;
            WPER^^.next := nil
          end;

        if GetNextEvent(EveryEvent, Event) then
          case Event.what of
            MouseDown:
              HandleMouseInSaveWindow;
            KeyDown, AutoKey:
              if (not isSelecting) and (not isEditing) and (not isClicking) then
                HandleKey(SaveWindow)
              else
                YouCantHaveDoThat(FALSE, TRUE, 0);
            ActivateEvt:
              HandleActivate;
            UpdateEvt:
              HandleUpdate;
            otherwise
          end;
        if (SelectIsHuge and SelectChoice and OkPress) then
          messageDone := TRUE;
        if ((Event.what <> MouseDown) and (Event.what <> KeyDown) and (Event.what <> AutoKey) and (WPER <> nil)) then
          begin
            DisposeHandle(Handle(WPER));
            WPER := nil
          end;
      until (Done or messageDone);

      UnLoadSeg(@DoApple);              {Segment 3}
      UnLoadSeg(@doEvtSetup);           {Segment 4}
      UnLoadSeg(@doHelp);               {Segment 5}
    end;
```

```
{ Editor is the Main Event Loop for the editor program.
  First unload as many segments as possible.  Then set up the grow zone procedure, and open the
  clipboard, instructions, and "untitled" windows.  The "Paste" operation is for testing
  purposes.  Then execute the main event loop.
```

```
  procedure Editor;
  begin
    UnLoadSeg(@DoApple);              {Segment 3}
    UnLoadSeg(@doEvtSetup);           {Segment 4}
    UnLoadSeg(@doHelp);               {Segment 5}

    SetGrowZone(@MyGrowZone);
    OpenWindow('Instructions', Nil, 0, InstructionsPtr, InstructionsRect, wInstructions);
    OpenWindow('Clipboard', Nil, 0, ClipBdWindow, ClipBdRect, wClipBd);
    OpenFile(TRUE);

{Paste in some text for testing views}
    if EdWrite <> nil then
      begin
        TEPaste(EdWrite^^.theText^.teHd);
        ShowSelect(EdWrite)
      end;

    DoLoop(FALSE, FALSE);
  end;
end.
```

```
{
 Margaret Stone
 Sc.M. Project, Brown University

 This file is the "Main program" for the editor. The editor is initialized, and the top level
 of the editor is executed.
}

program Editor;

   uses
      EditorInit, EditorHelpInit, EditorTopLevel, OpenPalette;

begin
   MoreMasters;
   MoreMasters;
   InitEditor;
   InitHelp;
   CreatePalette;
   UnLoadSeg(@InitEditor);          {Segment 2}

   Editor
end.
```