KEDIT for Windows User's Guide
Version 1.6

Mansfield Software Group, Inc.
P.O. Box 532
Storrs, CT 06268
http://www.kedit.com

December 2007

This PDF file contains the full text of the KEDIT for Windows 1.6 User's Guide. The entire document is in black and white, aside from the colored KEDIT logo on the cover page.

The text of the KEDIT for Windows 1.6 Reference Manual is available in a separate PDF file.

The contents of both the Reference Manual and the User's Guide are also available in HTML Help format in the KEDIT for Windows Help file, KEDITW.CHM.

Contents

Chapter 1. Introduction

KEDIT for Windows is a text editor for Microsoft Windows. It provides many powerful and useful facilities for working with text files. KEDIT is typically used to edit computer programs, notes and memos, e-mail, lists of information, and other textual data files

KEDIT is aimed at users who have some familiarity with personal computers and with the Windows user interface. Programming experience is useful for taking full advantage of KEDIT's macro facilities and extensive data manipulation capabilities, but is not required for normal use of the editor.

As a text editor, KEDIT works with files that consist of lines of text, where each line ends with a carriage return and/or a linefeed character. So, there are some types of files that KEDIT is not intended for, such as binary files and graphics files. KEDIT is also not a full word processor, and therefore does not automatically number pages or generate a table of contents, and does not have sophisticated printer support or support for multiple fonts within a document. Further, KEDIT generally assumes that the files you are editing are encoded using the ANSI or OEM character sets; most text files on Windows have historically used the ANSI character set. KEDIT does not support Unicode, a newer format that is beginning to come into wider use

While KEDIT can be useful to almost every Windows user who needs to work with text files, special features are included in KEDIT for users of IBM's mainframe editor XEDIT. Many of KEDIT's commands are compatible with corresponding XEDIT commands, and KEDIT supports a prefix area like XEDIT's. XEDIT users should note, however, that not all XEDIT features are available in KEDIT, and there are a number of differences between KEDIT and XEDIT. Most of the differences come about because KEDIT tries to take full advantage of the more flexible keyboard and display interface provided by the PC and by Windows.

1.1 Overview of Documentation

The KEDIT for Windows documentation is divided into two separate books: the *KEDIT for Windows User's Guide* and the *KEDIT for Windows Reference Manual*.

The *KEDIT for Windows User's Guide* is the book you're reading now. This guide provides an introduction to KEDIT, background information on most of KEDIT's features, and information on KEDIT's menu structure, dialog boxes, and keyboard interface.

The second book, the *KEDIT for Windows Reference Manual*, contains details of all of KEDIT's commands, options, and macro facilities.

The entire contents of both the User's Guide and the Reference Manual can also be accessed interactively via the KEDIT for Windows Help system, accessed via the Help menu from within KEDIT.

The User's Guide has 12 chapters and two appendices:

You are now reading Chapter 1, "Introduction".

Chapter 2, "Getting Started", is an informal introduction to KEDIT.

Chapter 3, "Using KEDIT for Windows", discusses a number of aspects of KEDIT, such as the command line, fonts and character sets, DIR.DIR directory listings, and the undo facility.

Chapter 4, "Keyboard and Mouse", discusses the key combinations and mouse actions that you can use to interact with KEDIT.

Chapter 5, "Menus and Toolbars", discusses KEDIT's menus and toolbars.

Chapter 6, "Targets", covers KEDIT's target facilities. Targets provide a flexible way to describe and locate any line within your file. You must have a good understanding of targets to make effective use of KEDIT. Regular expressions provide even more options for specifying string targets.

Chapter 7, "The Prefix Area", describes the use of the prefix area, an optional feature of KEDIT that is compatible with the prefix area provided by XEDIT.

Chapter 8, "Selective Line Editing and Highlighting", describes the ALL command and related commands that provide for selective viewing and editing of subsets of your file, and describes KEDIT's highlighting facility.

Chapter 9, "Tailoring KEDIT", discusses ways in which you can tailor many aspects of KEDIT's behavior to suit your preferences.

Chapter 10, "Using Macros", describes KEDIT's macro facilities, which allow you to reconfigure KEDIT and automate your editing tasks.

Chapter 11, “Sample Macros”, helps you learn more about KEDIT’s macro facilities by giving detailed explanations of a number of sample macros.

Chapter 12, “File Processing”, discusses KEDIT’s file locking facility, the file formats that KEDIT can read and write, and KEDIT’s handling of long filenames.

Many of KEDIT’s features are compatible with the features of XEDIT, the text editor used with IBM’s CMS system. Appendix A, “XEDIT Compatibility”, has some notes for XEDIT users on the differences between the two editors.

The other Appendix is Appendix B, “Glossary”.

The Reference Manual has nine chapters:

Reference Manual Chapter 1, “Introduction”

Reference Manual Chapter 2, “Invoking KEDIT”.

Reference Manual Chapter 3, “KEDIT Commands”.

Reference Manual Chapter 4, “The SET Command”.

Reference Manual Chapter 5, “QUERY and EXTRACT”.

Reference Manual Chapter 6, “Macro Reference”.

Reference Manual Chapter 7, “Built-in Macro Handling”.

Reference Manual Chapter 8, “KEDIT Language Definition Files”.

Reference Manual Chapter 9, “Error Messages and Return Codes”.

Chapter 2. Getting Started

2.1 Installing KEDIT

KEDIT for Windows 1.6 is a 32-bit program for Windows 2000/XP/Vista. To install KEDIT for Windows, simply run the install program that you received, either on CD or by download from the Internet, when you purchased your KEDIT license.

2.2 Your First KEDIT Session

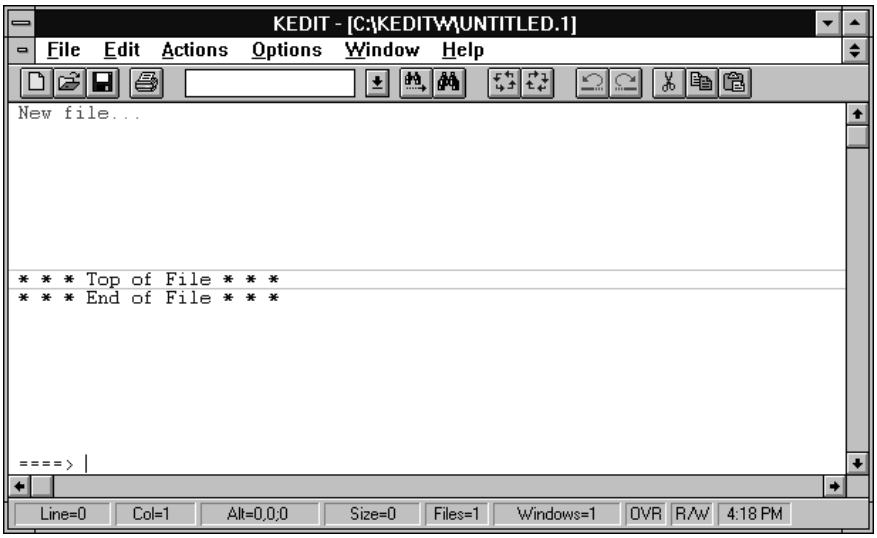
The best way to learn about KEDIT for Windows is simply to start using it. You need to be familiar with the basics of working with Windows and with Windows applications. If you are used to working with other Windows applications, then you already know a lot about working with KEDIT for Windows. This chapter therefore focuses on unique aspects of KEDIT for Windows that first-time users of the product need to be aware of.

If you have just installed KEDIT for Windows on your system, it will work as described in this chapter. However, KEDIT for Windows is highly configurable, so if someone else has installed KEDIT for Windows for you and has made changes to the default settings, KEDIT's appearance and behavior may vary from what is described here.

To start KEDIT, double-click on the KEDIT for Windows icon, which installed on your desktop and in the Windows Start Menu by KEDIT's install program.

2.3 The KEDIT Screen

When you first start KEDIT, the screen looks something like this:



Screen 2.1: A first look at the KEDIT screen

Some important elements of the screen include:

the frame window

the document window

and, within the document window, the file area and the command line

Frame window

The main KEDIT window is called the *frame window.* Elements of the frame window include the title bar, menu bar, toolbar and status line.

Document window

Within the frame window are one or more document windows. Note that, as in the above illustration, the document window is often maximized so that it occupies the entire frame window. But you can edit multiple documents and display them in separate document windows that appear within the frame window.

On the outside of the document window are a title bar, displaying the fileid, and horizontal and vertical scrollbars. Within the document window are the message line, top-of-file line, end-of-file line, current line, command line, and file area.

When KEDIT starts up, it normally presents a new empty file to which you can add text. This file is given a temporary name of UNTITLED.1, but when you save the file you would give it a permanent name. Most often, however, you would not work with UNTITLED.1 but would instead use the File Open dialog box to open an existing file that you want to edit. This existing file would replace UNTITLED.1 on the screen.

Nonetheless, we will look again at UNTITLED.1 for the moment. Near the top of the document window you may see the message "New file...". Some of KEDIT's messages, like this one, are presented to you in a message line at the top of the window.

Next, you will notice lines that say "Top of File" and "End of File". KEDIT displays the text of your file between these lines. In the empty file we are looking at now, there is nothing between these lines because there is no text in the file yet, but the lines help to distinguish between a file that has no data and a file that has a lot of blank lines.

Command line

The command line is one special feature of KEDIT that is not found in most windows applications, since most windows applications rely solely on menus and dialog boxes for input. KEDIT primarily uses menus and dialog boxes, but also uses the command line, where you can type some of the more specialized or less frequently used commands.

See Section 3.4, "The KEDIT Command Line", for a discussion of KEDIT's command line.

Current line

When the cursor is on the command line, as it is when KEDIT starts up, a line near the middle of the window, known as the *current line*, has a box drawn around it. In Screen 2.1 the box is drawn around the top-of-file line.

The current line is special because it is used in connection with many commands issued from the command line. For example, if you issue from the command line the command to go to line 30 of a file, line 30 will be positioned as the current line, with a box drawn around it.

Since the current line is important only in connection with the command line, the box is drawn only when the cursor is on the command line.

2.4 Working in the File Area

Most of the document window is occupied by the file area. This is the area where your file is displayed.

Moving to the file area

Most of your work is done with the cursor in the file area, which is the main body of the document window. To move the cursor from the command line, simply press the cursor up key. Now the cursor is in the file area.

Since this is an empty file, the only thing we can usefully do is to add some text to it. You can't type on the top-of-file or end-of-file lines, so you'll need to add a blank line to the file.

Adding a line

Press Enter to add a blank line to the file. Then, type some text, pressing Enter whenever you want to start a new line.

Changing text

You can edit the text, using the mouse pointer or the cursor arrow keys to move through the text, and using keys like Backspace and Delete.

Overtype Mode and Insert Mode

KEDIT normally starts out in Overtype Mode, and characters that you type will replace existing characters at the cursor position. When you are in Insert Mode, text at the cursor position is pushed to the right when you type in new characters. You can press the Insert key to toggle between Overtype Mode and Insert Mode.

Note that your current Insert Mode or Overtype Mode status is displayed on the status line at the bottom of the frame window, where an indicator shows either "INS" or "OVR". As an additional reminder of which mode you are in, KEDIT makes the cursor thicker when you are in Insert Mode and thinner when you are in Overtype Mode.

2.5 Working With Menus

Now let's look at some of the things you can do using the KEDIT menus. For a full description see Chapter 5, "Menus and Toolbars".

2.5.1 The File Menu

The File menu is used to begin editing files, finish editing files, and to do some related tasks like printing a file.

Editing multiple files

At the moment you are editing the initial UNTITLED.1 file. We will now begin working with a second file. (For the purposes of this discussion, be sure that you have added at least one line of text to UNTITLED.1. This is necessary because we want to show how KEDIT can work with multiple files, and UNTITLED.1 is a special file that KEDIT automatically closes if it is unchanged and you begin to edit another file.)

You can use File Open to begin editing an additional file. A sample file that you can use to practice with, DICKENS.TXT, is installed by default in the main KEDIT program directory, which is usually the C:\Program Files\KEDITW directory. Using the File Open dialog box, navigate to that directory and open the file DICKENS.TXT. You will then be working with two files: UNTITLED.1 and DICKENS.TXT. Each file is in its own window. Since your document window is maximized, occupying the entire frame, you'll initially see only one window, but there are two there.

2.5.2 The Window Menu

To work with your document windows, you can use the Window menu. If you want to see all of your windows laid out on the screen you can select Window Tile Horizontally, which arranges the windows from top to bottom. You can also try Window Tile Vertically, which arranges your document windows from left to right. Window Cascade arranges the document windows so that all their title bars are visible.

Maximizing a window

If you used one of the Window menu items just discussed, both of your document windows (the document window with DICKENS.TXT and the document window with UNTITLED.1) will be visible within the frame window. You may want to work with document windows this way, or you may more often want to see only one window at a time, maximized so that it occupies the entire frame window. To maximize a document

window you can either double-click on its title bar or you can click on the maximize button at the right of the document window's title bar. When windows are maximized, one way to switch between document windows is to select the name of the document window that you want to work with from the window list at the bottom of the Window menu.

2.5.3 The Edit Menu

Find and Replace

You can use the Edit menu for Find and Replace operations. For example, assume that you're using the file DICKENS.TXT and you want to find the string "Dickens". Select Find from the Edit menu to get to the Find dialog box. Then type in the string ''Dickens'' and click on the Forward button to search forward in the file or on the Backward button to search backward in the file. When you are finished with the Search dialog box, click on the Close button. The Edit Replace dialog box is similar to the Edit Find dialog box, except that it not only finds text strings within a file but can also replace them with new strings.

Using the clipboard

You can also use the Edit menu to work with the clipboard. First drag with mouse button 1 to mark some text that you want to cut or copy to the clipboard. You can use Edit Cut on the Edit menu to put the text in the clipboard and delete it from your file, or you can use Edit Copy to put the text in the clipboard and leave it in your file. Once you have copied text to the clipboard you can move the cursor to another location in the file, or you can switch to another Windows application, and use Edit Paste to take the text from the clipboard and place it into the file at the new location.

Undo and Redo

The Edit menu also has Undo and Redo entries. You can undo a change to your file with Edit Undo. Immediately after you've undone a change to the file, you can use Edit Redo to redo that change.

Selective Editing

Another useful feature of KEDIT is its selective editing facility. You can use Edit Selective Editing to access this facility. Selective editing lets you focus only on the lines of your file that contain a particular string of interest, for example the name of a variable in a computer program.

To try out the selective editing facility with DICKENS.TXT, use the Edit Selective Editing menu item to get to the Selective Editing dialog box. Then type in the word ''Dickens'' and press the Matching Lines button to view only the lines containing the string "Dickens". You will see the lines in the file containing "Dickens", and you will also see what are called "shadow lines", indicating how many lines are omitted because they do not contain the word "Dickens". In a large file in which a word appears only a few times, Selective Editing lets you zero in on exactly the lines you are interested in. To go back to the normal mode of working with the entire file, again use Edit Selective Editing and press the All Lines button.

2.5.4 The Actions Menu

Several editing operations are controlled through the Actions menu. You can set bookmarks at different locations in your file, so that you can easily return to these locations

later in an editing session. You can sort the contents of your file, fill in portions of your file with a specified character, uppercase text in your file, or lowercase text in your file.

2.5.5 The Options Menu

Screen font

The Options Screen Font dialog box lets you control the font that KEDIT uses within your document windows. Because KEDIT's main focus is on the content of your file, as opposed to its appearance, KEDIT's facilities for working with fonts are limited. KEDIT uses only fixed pitch fonts (that is, non-proportional fonts, in which each character has the same width). A single font is used for all of the text within your files; you cannot display part of a file in one font and another part of a file in a different font. See Section 3.6, "Fonts", for more information about KEDIT's font handling. Note that the font that KEDIT uses when printing is controlled separately, via the File Print dialog box.

CUA and Classic interfaces

The Options Interface dialog box is primarily intended for users of earlier text mode versions of KEDIT who are moving to KEDIT for Windows.

By default, keyboard and mouse usage within KEDIT is very much like keyboard and mouse usage in other Windows applications. But some KEDIT users would prefer to stay with the keyboard and mouse conventions they are familiar with from text mode versions of KEDIT. The Options Interface dialog box lets you choose between these two ways of using KEDIT. The default interface is the Windows-style interface that we will refer to as the *CUA* (Common User Access) interface. The text mode compatible interface is referred to as the *Classic* interface.

The Options Interface dialog box also lets users of the CUA interface fine-tune some details of the keyboard and mouse behavior, making things work a bit more like they did in text mode KEDIT.

For more on the CUA and Classic interfaces, see Section 3.2, "CUA and Classic Interfaces".

Settings

KEDIT has a large number of settings that you can use to control, for example, whether KEDIT displays scroll bars, what colors KEDIT uses to display your files, and whether KEDIT's wordwrap facility is enabled. These settings are also referred to as SET command options, because you can control them by using the dialog boxes discussed here or by issuing the SET command from the KEDIT command line.

Three dialog boxes related to your settings are accessed through the Options menu:

Options SET Command lets you work with individual settings. You can select a setting from a list of all available settings or from groups of settings organized by category. You can then examine the value of the setting, and can make changes to that value.

Options Status displays a list of the current values of most KEDIT settings.

Options Save Settings lets you save the current values of most settings for use in future editing sessions.

2.6 Working with the Toolbar

The fastest way to get to many frequently-used KEDIT features is via the toolbar. For example, the toolbar has items to undo a change, redo a change, save a file to disk, to print a file, etc. The toolbar appears at the top of the window. When you position the mouse over a toolbar item and click, the function of that toolbar button will be carried out. If you linger over a button with the mouse, a small pop-up help box will appear, indicating the function of the button. Additional information about the button appears in the status line at the bottom of the frame window. An optional bottom toolbar provides additional toolbar buttons; you can use the TOOLBAR option of the Options SET Command dialog box to enable the bottom toolbar. See Chapter 5, "Menus and Toolbars", for a full discussion of KEDIT's toolbars.

Quick Find

One item on the toolbar is not a button, but is a special combo box known as the Quick Find toolbar item. Quick Find displays the string you most recently searched for, and provides a shortcut way of accessing most functions of the Edit Find dialog box. You can press the Find Next button, located to the right of Quick Find, to search again for this string. You can also type in a new string, or choose from a dropdown list of all the strings that you have recently searched for.

2.7 Getting Help

To get into the help system from within KEDIT press F1.

Press F1 from within any dialog box to get help related to that dialog box.

Press F1 while pulling down a menu to get help with the current menu item.

Press F1 at any other time to get to the help system's main Contents screen.

You can also access the help system from the Help menu.

2.8 Ending a Session

Saving Files

In this practice session we haven't made any useful changes to DICKENS.TXT or UNTITLED.1, so we don't need to save them to disk. In future editing sessions, however, you will probably want to save your files before leaving KEDIT.

You can save a file to disk at any time during a KEDIT editing session by using the File Save menu item or by using the Save File toolbar button.

Closing Files

When you are completely finished with a file you can select File Close from the menu. That will close the file, removing it from memory and removing its document windows from the screen. If you have made changes to the file that have not yet been saved, KEDIT will ask if you want to save the file to disk before closing it.

Note that if you try to save a temporary file like UNTITLED.1 to disk, KEDIT will prompt you for the name you want to save it under.

Leaving KEDIT

Now that we've had a very brief look at some of the facilities of KEDIT, let's end the KEDIT session.

The easiest way to leave KEDIT completely is to select File Exit. This will close any files that are active within KEDIT, again prompting you, if necessary, to save any unsaved changes, and will then take you out of KEDIT.

Chapter 3. Using KEDIT for Windows

This chapter takes brief looks at a number of topics that you will need to be familiar with if you want to take full advantage of KEDIT for Windows.

3.1 Frame Window and Document Windows

Many Windows applications use a set of conventions known as the Multiple Document Interface (MDI) to let you work with several documents at the same time. Since KEDIT lets you edit several files at a time, KEDIT also uses the Multiple Document Interface.

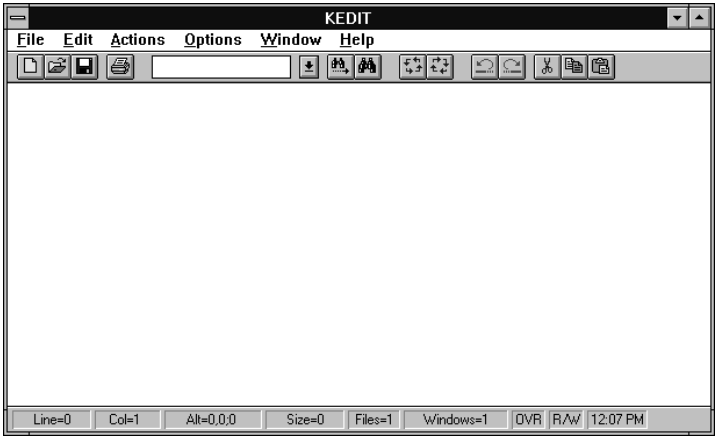
In an MDI application there is one main window, known as the *frame window*, which provides the work area in which one or more *document windows* are displayed.

The frame Window can be *maximized*, so that it occupies your entire Windows desktop, it can be in its *normal* state (sometimes referred to as the *restored* state), where it occupies only part of the Windows desktop and can be moved and resized, or it can be *minimized*, so that it appears as an icon on your Windows desktop.

Just as the frame window can be maximized, restored, and minimized on your Windows desktop, your document windows can be maximized, restored, and minimized within the frame window. A document window can be maximized so that it occupies the entire area of the frame window; it can be in its normal state, where it occupies only a part of the frame window, and can be moved and resized; or it can be minimized, so that it appears as an icon within the frame window.

Frame window layout

KEDIT's frame window is organized as follows:



Screen 3.1: KEDIT's frame window

At the top of the frame window is a *title bar*. The title bar displays the name of the application, KEDIT. If you are working with a maximized document window, the

frame window's title bar also displays the name of the file that you are working with. At the right of the title bar are icons that you can click on to maximize, minimize, or restore the frame window. At the left of the title bar is the frame window's *system menu* icon. If you click on the system menu icon, you will get a menu that allows you to do things like move, resize, or close the frame window. Closing the frame window, which you can also do by double-clicking on the frame window's system menu icon, closes all of your document windows (you are prompted to save any unchanged files) and ends your KEDIT editing session.

Underneath the title bar is the *menu bar*, from which you can access all of KEDIT's menu items.

Underneath the menu bar is the *toolbar*, which has a number of buttons that you can click on to perform tasks like saving the current file, printing the current file, etc. The toolbar also contains a special Quick Find item that displays the string that you most recently searched for and provides a fast way to access some of the functions of the Edit Find dialog box.

The main portion of the frame window is the workspace in which your document windows are displayed.

Near the bottom of the frame window an optional bottom toolbar, with an additional set of useful toolbar buttons, can be displayed. The bottom toolbar is not displayed by default, but can be enabled through the TOOLBAR setting of the Options SET Command dialog box.

At the very bottom of the frame window is the *status line*, which gives assorted information on the status of your KEDIT session.

Status line contents

The following information is displayed on the status line at the bottom of the frame window:

The line number of the cursor's location within the current file (or the line number of the current line, if the cursor is on the command line).

The column location of the cursor.

The alteration and undo counts. Three numbers are given. The first is the number of changes to your file since it was last saved (for example, via the File Save menu item or the SAVE command) or autosaved (via KEDIT's autosave facility). The second is the number of changes to your file since the last save; this number is not reset after an autosave. The third number is the number of changes to your file that can currently be reversed by using the Edit Undo menu item.

The size of the current file.

The number of files in the ring. The set of files being edited is known as the *ring*. You can use KEDIT to edit up to a maximum of 500 files at a time.

The number of document windows. This can be different than the number of files in the ring if, for example, you have used Window New to create additional document windows viewing the same file. If you have marked a block of text within

your file, KEDIT displays information about the type of block that is marked in place of the number of document windows.

Insert/Overtype status: ‘‘INS’’ if you are in Insert Mode, or ‘‘OVR’’ if you are in Overtype Mode.

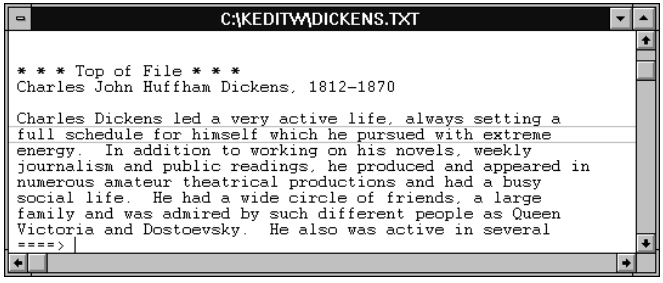
File locking status: ‘‘Lock’’ if you are using KEDIT’s file locking facility to prevent others from accessing a file that you are editing, ‘‘R/O’’ if you are editing a file with the read-only attribute bit set in its directory entry, or ‘‘R/W’’ for any other file.

The time-of-day. This field is optional and can be turned off through the CLOCK option of the Options SET Command dialog box.

The hexadecimal and decimal codes for the character at the cursor position. By default, this item is not displayed; it can be enabled via the HEXDISPLAY option of the Options SET Command dialog box.

Document window layout

Here is a description of the default document window layout, which is shown below. Note, however, that the document window layout can vary greatly, since there are a number of options that you can use to change its appearance:



Screen 3.2: KEDIT's document window

At the top of the document window is a title bar. The title bar gives the name of the file that is displayed within the document window. At the right of the title bar are icons that you can click on to maximize, minimize, or restore the document window. At the left of the title bar is the document window’s *system menu* icon. If you click on its system menu icon, you will get a menu that allows you to do things like move, resize, or close the document window. You can also close the document window by double-clicking on the system menu icon. If the document window is maximized, the file name involved is displayed in the frame window’s title bar and the document windows’s system menu icon is displayed at the left of the menu bar.

At the right of the document window is the vertical scroll bar, which you can use to scroll up or down within your file. At the bottom of the document window is the horizontal scroll bar, which you can use to scroll left and right within your file.

The main area of the document window is known as the *file area*. The file area displays a portion of your file; you can use the scroll bars to scroll different portions of your file into view. In the middle of the document window is the current line.

When the cursor is on the command line, KEDIT draws a box around the current line to make it stand out, because most KEDIT commands issued from the command line operate on the current line or on some portion of your file beginning with the current line.

Between the bottom of the file area and the horizontal scroll bar at the bottom of the document window is the command line. KEDIT displays an arrow (“====>”) at the beginning of the command line to make it easily distinguishable from the file area. You can give commands to KEDIT by typing them on the command line and then pressing the Enter key.

3.2 CUA and Classic Interfaces

KEDIT for Windows supports two sets of keyboard and mouse conventions. With the default conventions, keyboard and mouse usage in KEDIT is very much like keyboard and mouse usage in most other Windows applications. If you are comfortable using other Windows applications, you will probably find KEDIT’s default keyboard and mouse interface, referred to as the CUA interface, easy to get used to. Your other choice is KEDIT’s Classic interface, in which keyboard and mouse usage is very much like it was in earlier text mode versions of KEDIT, such as KEDIT 5.0 for DOS and for OS/2. You can use the Options Interface dialog box to switch between the CUA interface and the Classic interface.

The CUA interface got its name because the conventions involved were originally based on IBM’s Common User Access (CUA) guidelines, but the version adopted in KEDIT comes mainly from MicrosoftÆs user interface guidelines.

KEDIT’s Classic interface is intended for users of text mode versions of KEDIT who are converting to KEDIT for Windows and do not want to switch to a different set of keyboard and mouse conventions. For example, with the Classic interface, pressing Alt+W deletes the word at the cursor position, because this is what Alt+W does in text mode KEDIT. But with the CUA interface, Alt+W instead pulls down the Window menu, because this is how most other Windows applications work, and the word delete function has been moved to Shift+Ctrl+W.

Chapter 4, “Keyboard and Mouse”, gives the details of KEDIT’s keyboard and mouse usage, with full discussions of the CUA interface and of the Classic interface, and with a summary of the differences between the two.

Most KEDIT users will probably want to use the default CUA interface. This is certainly true if you are a new user of KEDIT who is not already accustomed to the conventions of text mode KEDIT. It is also true if you are a user of text mode KEDIT who is converting to KEDIT for Windows, because the CUA interface makes KEDIT work like the other Windows applications that you are likely to be using.

KEDIT text mode users switching to the CUA interface sometimes find that they can quickly adjust to most aspects of the new interface, but that the new behavior of a few keys is hard to get used to. For example, in text mode KEDIT the Enter key moves the cursor to the beginning of the next line and the Home key moves the cursor to the

command line, while with the CUA interface the Enter key inserts a new line into your file and the Home key moves the cursor to the beginning of the line it is on. To deal with this situation, you can use the Options Interface dialog box to make the Enter key, the Home key, and a few other keys work like they did in text mode KEDIT, even though most of KEDIT's behavior is based on the CUA interface.

The Options Interface dialog box is probably the most convenient way to choose between the CUA and Classic interfaces, but you can also use the command SET INTERFACE CUA to get the CUA interface and the command SET INTERFACE CLASSIC to get the Classic interface.

Notes

This documentation will frequently refer to whether INTERFACE CUA or INTERFACE CLASSIC is in effect. INTERFACE CUA is in effect when you are using the CUA interface, either because it is the default interface, or because you chose it via the Options Interface dialog box or the SET INTERFACE command. INTERFACE CLASSIC is in effect when you are using the Classic interface, selected via Options Interface or the SET INTERFACE command.

3.3 Blocks and Selections

This section introduces some of the concepts that you will need to understand to work effectively with KEDIT's blocks. Details of the keystrokes and mouse actions used to select and operate on blocks are not given here but are covered instead in Chapter 4, "Keyboard and Mouse".

A *block* is a portion of your file that you have selected, most often by dragging the mouse, so that you can operate on it as a unit. For example, you may want to copy the block to the Windows clipboard, or you may want to uppercase the text within the block, or sort it. KEDIT supports two types of blocks, *non-persistent blocks* (most often referred to as *selections*) and *persistent* blocks. Selections are only available if you are using the CUA interface, but persistent blocks are available under both the CUA and Classic interface.

A selection, or non-persistent block, works the same in KEDIT as in most other Windows applications: you can mark a selection by dragging the mouse pointer over the text involved, or by pressing Shift in combination with a cursor-movement key to move the cursor over the text. Once you have marked a selection, you must operate on it immediately, for example by using the Edit Copy menu item to copy the selection to the clipboard, or by pressing the Delete key to delete the selection. If you mark a selection and then type some text, the selection is deleted and the text that you type is entered in its place; this feature is referred to as ''typing-replaces-selection''. If you do not operate immediately on a selection that you have marked, but instead move the cursor elsewhere in your file, the selection is automatically unmarked; this is why selections are referred to as non-persistent.

Most Windows applications support only non-persistent blocks, but KEDIT also supports persistent blocks. A persistent block is a block that, once it has been marked, remains marked even if the cursor moves away from it. You can, for example, mark a

persistent block, move elsewhere in your file, copy the block to the new location, change some text within the block, and finally use the Edit Unmark menu item or press Alt+U to unmark the block. The ability to move the cursor away from a block, and to perform multiple operations on the block without unmarking it, can sometimes be quite useful.

Both persistent and non-persistent blocks are available in any of three different shapes:

A *stream block* is a stream of consecutive characters of your file, possibly spanning multiple lines of the file.

A *line block* is a group of consecutive lines of your file.

A *box block* is a rectangular area of text within your file. For example, the text in columns 10 through 20 of lines 50 through 80 of your file might make up a box block.

If you are using INTERFACE CUA, you have access to both persistent blocks and selections (that is, non-persistent blocks). The mouse marks selections, although you can use the Options Interface dialog box to specify that the mouse should instead mark persistent blocks. Shift+cursor key combinations also mark selections, as they do in other Windows applications. To mark persistent blocks with the keyboard, you can use the Alt+L, Alt+B, and Alt+Z keys, which mark, respectively, persistent line, box, and stream blocks. You can also mark a persistent block by first marking a selection, and then using the Edit Make Persistent menu item to convert the selection to a persistent block.

If you are using INTERFACE CLASSIC, only persistent blocks are available to you. This is because in text mode KEDIT all blocks are persistent blocks and selections are not supported.

When a block is marked, the "Windows=" item on the status line is replaced by an indication of the type of block involved (persistent or selection, and line, box, or stream). When the block is not in the current file (this is possible only for persistent blocks) the block type is followed by a greater than (">"). When the block has just been marked and you can press the Delete key to delete the block (this happens only with the CUA interface), the block type is followed by an asterisk ("*").

3.4 The KEDIT Command Line

3.4.1 Command Line Basics

The command line provides a very handy way to pass instructions to KEDIT.

Although most KEDIT facilities are accessible via menus, toolbar buttons, or key combinations, some features have options and operands that are only available through commands entered on KEDIT's command line.

And, once you are familiar with certain frequently-used KEDIT commands, you might find it faster to simply type them on the command line than to access them through the menu system.

Finally, some of KEDIT's less frequently used commands are available only from the command line.

The command line is normally displayed at the bottom of the document window and begins with an arrow ("====>") to help distinguish it from the file area. To move the cursor from the file area to the command line, press the F12 key. You can type a command on the command line, for example

```
DELETE 10
```

and then press the Enter key to tell KEDIT to execute the command. In this case you issued the DELETE command with the operand 10, telling KEDIT to delete ten lines from the current file.

Minimal truncations

In fact, most KEDIT commands can be abbreviated. For example, the DELETE command can be entered as DEL, DELE, DELET, or DELETE. The shortest legal abbreviation for a command is known as the *minimal truncation*. Since the minimal truncation for the DELETE command is DEL, you could also type

```
DEL 10
```

to delete ten lines from your file. To make each command's minimal truncation clear, the command documentation in KEDIT's Reference Manual and online help files gives the minimal truncation in uppercase, with the rest of the command name in lowercase. For example, the documentation for the DELETE command uses ''DELete'' to indicate that DEL is the minimal truncation for DELETE.

Current line

Commands issued from KEDIT's command line act relative to the current line, which is normally displayed in the middle of the document window and which normally has a box drawn around it whenever the cursor is on the command line. For example, with the command

```
DEL 10
```

it is the current line and the nine lines following it, for a total of ten lines, that will be deleted.

Command retrieval

You can use Ctrl+Cursor Up to redisplay the most-recently-issued command line. You can then make changes to this command line text, perhaps fixing a typing mistake, and then press Enter to reissue the command. You can press Ctrl+Cursor Up or Ctrl+Cursor Down repeatedly to cycle backward or forward through your recently-entered command lines. KEDIT keeps track of the last 40 command lines, and saves them from one editing session to the next.

If Ctrl+Cursor Up or Ctrl+Cursor Down are pressed when the command line is empty, they cycle through all of your recent command lines. But if you type some text on the command line and then press Ctrl+Cursor Up or Ctrl+Cursor Down, they will only display previous command lines that begin with that text. For example, you can type an ‘‘a’’ on the command line and then press Ctrl+Cursor Up to retrieve the most recent command line that began with an ‘‘a’’.

If you simply want to reissue the last command and don’t need to display it again, you can enter the = command on the command line or you can press F9.

3.4.2 Some Useful Commands

Here is an introduction to some commonly used commands. See Reference Manual Chapter 3, “KEDIT Commands”, for full documentation of all of KEDIT’s commands.

TOP and BOTTOM

You can use the TOP command to move to the beginning of your file, and you can use the BOTTOM command to move to the end of your file.

LOCATE

With the Edit Find dialog box you can search your file for the text that you specify. You can search forward or backward, and can match case, limit the search to whole words only, and use regular expression notation. Using the Edit Find dialog box is similar to using the LOCATE command from the KEDIT command line. For example, to find the string ‘‘payment’’, you can type ‘‘payment’’ into the Edit Find dialog box, or you can type the following on the command line:

```
LOCATE /payment/
```

In the above example, KEDIT would locate the next line that contained the string “payment” and make that line become the current line. Note that the string target is surrounded by delimiter characters which, as in this example, are usually slash characters (“/”). The LOCATE command is used so frequently than you can normally omit the word LOCATE and, as a shortcut, simply specify the target that you are looking for, as in this example:

```
/payment/
```

An advantage of the LOCATE command over Edit Find is that LOCATE can do more sophisticated string searches. You can, for example, use logical operators to combine targets. For example, to locate the next line containing both ‘‘payment’’ and ‘‘check’’, you could use the command

```
/payment/ & /check/
```

By specifying numeric operands for the LOCATE command, you can move up or down in your file, or to a specific line of your file; this use of the LOCATE command is equivalent to using the Edit Go To dialog box. So the command

```
LOCATE 25
```

or the equivalent command

```
25
```

would make the line 25 lines below the current line become the current line. The command

```
-25
```

would make the line 25 lines above the current line become the current line. Instead of specifying how many lines to move relative to the current line, you can precede the number with a colon (":") to go to a specific line number. The command

```
:25
```

would move to line 25 of your file.

Chapter 6, "Targets", has a full discussion of the different types of targets that you can use with LOCATE and other commands.

CHANGE

The CHANGE command is another example of a KEDIT command whose command line options allow more complex expressions than the dialog box equivalent. Much of the function of the CHANGE command is available through the Edit Replace dialog box, which allows you to specify a string to find and a replacement string. You can use the Edit Replace dialog box to find the next occurrence of a string and replace the string, for each individual occurrence, or for all occurrences. And, as with the Edit Find dialog box, you can match case, limit the search to whole words only, and use regular expression notation.

Using the CHANGE command on the KEDIT command line, however, provides even greater flexibility and control. You can, for example, type

```
CHANGE /warp/woof/ 10 *
```

to change all occurrences of the string "warp" to "woof" on the current line and the nine lines below it, for a total of ten lines. The first operand following the delimited strings specifies what portion of the file will be affected: the 10 means that ten lines will be affected. The next operand determines how many occurrences on each line will be affected: the asterisk ("*") means that all occurrences on each of the ten lines will be affected.

Another CHANGE command example:

```
C ;A/B;C/D; * 5
```

Here "C", the minimal truncation of the CHANGE command, is used. Note that the delimiter used in this example is the semicolon (";"), since if the strings that you are working with contain any slashes, then you must use some special character other than

a slash as the delimiter. The first five occurrences of ‘‘A/B’’ on all lines from the current line through the end of the file will be changed to ‘‘C/D’’. This is because the first operand following the delimited strings is an asterisk (‘‘*’’), indicating that all lines from the current line through the end of the file will be changed, and the next operand indicates that the up to five occurrences on each line will be affected.

ALL

Another frequently-used command is the ALL command, whose function is also available through the Edit Selective Editing dialog box. You can use KEDIT’s ALL command to view and work with a subset of your file. For example, you can tell KEDIT to display only those lines in your file that contain a particular string. The other lines of your file are temporarily excluded from display and from processing of most commands.

For example, to display only the lines of your file that contain the string ‘‘yesterday’’, you can type ‘‘yesterday’’ into the Edit Selective Editing dialog box and press the Matching Lines button, or you can issue the command

```
ALL /yesterday/
```

When you have finished working with the subset of your file, you can use the Edit Selective Editing dialog box and press the All Lines button, or you can issue the ALL command with no operands. KEDIT will again display all lines of your file.

When you enter the ALL command on the command line, more types of target specifications, and more complex specifications are possible than with the Edit Selective Editing dialog box. For example, you can enter

```
ALL /upper/ | /lower/
```

to select all the lines in your file containing either “upper” or “lower” to be displayed, or

```
ALL ALTERED
```

to select only lines that have been added or changed in the current editing session.

For a full discussion of the ALL command and of other commands used with KEDIT’s selective line editing facility, see Chapter 8, “Selective Line Editing and Highlighting”.

TAG

Similar to the ALL command is the TAG command, which tags, or highlights, specified lines. For example

```
TAG /yesterday/
```

would highlight lines containing “yesterday”, without excluding the other lines of the file from display. This can be handy if you want to see the targeted lines in the context of the whole file.

For more on KEDIT’s highlighting facility, see Chapter 8, “Selective Line Editing and Highlighting”.

KEDIT

There is a command called KEDIT (which has K as its minimal truncation) that you can issue from the command line, as an alternative to the File Open dialog box, when you want to begin editing additional files. For example, you could use the command

```
K FILESWAP.KEX
```

to begin editing the file FILESWAP.KEX.

You can use the KEDIT command with no operands to move to the next file in the ring of files being edited, and you can issue it repeatedly to cycle through all of the files that you are editing:

```
K
```

This use of the KEDIT command is equivalent to using the Next File toolbar button.

For more about editing multiple files with KEDIT, see Section 3.5, “Editing Multiple Files”.

SAVE, FILE, QUIT and QQUIT

Several KEDIT commands are concerned with saving your file to disk and/or removing your file from the ring of files being edited:

The SAVE command saves the current file to disk. Using SAVE is similar to using the File Save menu item.

The FILE command saves the current file to disk and then removes the file from the ring. Using the FILE command is similar to using File Close, except that FILE always writes the file to disk, regardless of whether it has been modified.

The QUIT command removes the current file from the ring if it has not been modified; attempting to QUIT from a modified file yields an error message. Using the QUIT command is similar to using File Close on an unmodified file.

The QQUIT command removes the current file from the ring; the file is not saved to disk, even if it has been modified. Using the QQUIT command is similar to using File Close and specifying ‘‘No’’ if you are asked whether to save a changed file to disk.

SET and QUERY

KEDIT has over a hundred SET options that you can use to modify different aspects of KEDIT’s behavior. Most of these SET options can be controlled from the Options SET Command dialog box, and all of them can be controlled by issuing the SET command from the KEDIT command line.

You might, for example, decide to change the margin settings (used in connection with wordwrap, paragraph reformatting, etc.) and enter the command

```
SET MARGINS 7 67
```

where 7 and 67 are your desired left and right margin columns. The same SET option could also be entered as

```
MAR 7 67
```

since “MAR” is the minimal truncation of the SET MARGINS option. Note that the word “SET” is optional for nearly all of the SET options; if KEDIT sees a command that it does not recognize, it checks to see if you are in fact using a SET command option.

To check the status of an individual SET option, you can use the QUERY command, whose minimal truncation is Q. For example,

```
Q MARGINS
```

will display the current margin settings in the document window’s message area.

You can use the STATUS command, which is equivalent to the Options Status menu item, to get a dialog box displaying the value of almost all of the SET options.

For more about using KEDIT’s SET options, see Chapter 9, “Tailoring KEDIT”, and Reference Manual Chapter 4, “The SET Command”.

HELP

There are a number of ways to access KEDIT’s help system. You can use the Help menu, you can press the F1 key at any time, or you can issue the HELP command from the KEDIT command line. An advantage of the HELP command is that you can specify the name of a KEDIT command, option, or function, or you can specify a KEDIT error message number, and if possible the help system will jump directly to the topic that you specify. For example,

```
HELP LOCATE
```

will display help for the LOCATE command, and

```
HELP ERROR 144
```

will display information about KEDIT error number 144.

3.5 Editing Multiple Files

3.5.1 The Ring

You can use KEDIT to edit multiple files at the same time, up to a maximum of 500 files. Since KEDIT reads the files that you are editing into memory, the number of files that you can edit may also be limited by available Windows virtual memory.

The set of files that you are editing is referred to as the *ring*. You can think of the files you are editing as organized in a circular list. For example, if you were editing three files, you could use the Next File toolbar button to move from the first file in the ring to the second file, and then to the third file, and then back to the first file. And you could use the Previous File toolbar button to move backwards in the ring, from the first file to the third file, and then to the second file, and then back to the first file. Issuing the KEDIT command with no operands:

```
KEDIT
```

is equivalent to using the Next File toolbar button—you move to the next file in the ring. Issuing the KEDIT command with a minus sign as its operand:

```
KEDIT -
```

is equivalent to using the Previous File toolbar button—you move to the previous file in the ring.

The file in which the cursor is located is referred to as the *current file*. When you begin editing an additional file, the new file is added to the ring after the current file, and the new file becomes the current file. When you remove a file from the ring, the file that precedes that file in the ring becomes the current file.

Note that the discussion in this section assumes that, as will normally be the case, you are in one-file-per-window mode. One-file-per-window mode is discussed below in Section 3.5.2, "One-File-Per-Window".

Adding files to the ring

When you add a file to the ring (that is, when you begin editing an additional file), KEDIT loads the file into memory, creates a new document window, and then displays the file in that document window. There are several ways to add files to the ring. Here are the most frequently-used methods:

You can use the File Open dialog box to begin editing an additional file. There are two versions of the File Open dialog box; SET FILEOPEN controls which version KEDIT uses. With the default of FILEOPEN SINGLE, the File Open dialog box can only add a single file at a time to the ring. But with FILEOPEN MULTIPLE, you can select multiple files (by using Ctrl+button 1 or using button 2) from within the File Open dialog box and add them all to the ring.

You can use the File New menu item to begin editing a new, untitled file. The first untitled file you edit will have the name UNTITLED.1, the second will have the name UNTITLED.2, etc. When saving one of these untitled files to disk, you will need to specify a more permanent name for the file, normally by using the File Save As dialog box.

You can issue the KEDIT command from the command line, specifying the fileid of the file that you want to begin editing as its operand. To add multiple files to the ring, you can specify multiple fileids or you can use asterisk ("*") and question mark ("?") as wildcards in the file name and extension. For example, to begin editing SAMPLE.C:

```
KEDIT SAMPLE.C
```

or to begin editing TEST1.C and all files in the current directory with an extension of .TXT:

```
KEDIT TEST1.C *.TXT
```

You can drag-and-drop files onto KEDIT from the Windows Explorer.

From a directory listing displayed in a DIR.DIR file created by KEDIT's DIR command, you can double-click on a file description, or can move the cursor to a file description and press Alt+X. KEDIT will add the file involved to the ring.

Two special cases to be aware of:

If you attempt to add a file to the ring that is already in the ring, KEDIT simply re-activates the copy of the file that you are already editing. For example, if you issue the command

```
KEDIT C:\DATA\FINDINGS.TXT
```

but C:\DATA\FINDINGS.TXT is already in the ring, KEDIT will make the existing in-memory copy of C:\DATA\FINDINGS.TXT become the current file, and will not reload C:\DATA\FINDINGS.TXT from disk.

If, as is most often the case, no fileid is specified at the start of a KEDIT session, KEDIT starts with a single untitled file in the ring, UNTITLED.1. If you then add some other file to the ring, without having made any changes to UNTITLED.1, KEDIT automatically removes UNTITLED.1 from the ring, on the assumption that you are really interested in editing the other file, and not UNTITLED.1.

Multiple views

KEDIT lets you view the same file in multiple document windows. For example, if you are editing a large file, you could display the top of the file in one document window and the bottom of the file in another document window. Use the Window New menu item to create a new document window that displays an additional view of the current file.

If you are using multiple document windows to view a file and want to remove one of the document windows, you can use the Close menu item on the document window's system menu, or, as a shortcut for this, you can double-click on the document window's system menu. As discussed below, if there are multiple document windows displaying a file, closing one of those document windows leaves the other windows open and leaves the file in the ring. If a file is displayed in only a single document window, closing that window also causes KEDIT to remove the file from the ring, after giving you a chance to save any changes that you have made to the file.

Removing files from the ring

When you have finished working with a file, you can remove it from the ring. Depending on how you remove the file from the ring, the version of the file on disk may or may not be updated to reflect any changes you have made to the file. The file will be removed from memory, and you will need to add it back to the ring if you want to view it again or make further editing changes to it.

Here are the most common methods of removing files from the ring:

You can use the File Close window item. If the file has been changed, KEDIT will ask whether it should first save the file to disk and, depending on your response, will do so. Then KEDIT will remove the file from the ring and will close all document windows displaying the file.

You can use the Close item on the document window's system menu or, as a shortcut for this, you can double-click on the document window's system menu. KEDIT will close the current document window. If, as is most often the case, the file is not displayed in any other document windows, KEDIT will also remove the file from the ring, after first asking whether any unsaved changes should be written to disk.

From the command line, you can issue the FILE command, which will write the file to disk and then remove it from the ring. You can also issue the QUIT command, which will remove the file from the ring without saving it to disk if the file has not been modified, but which will give you an error message if the file has been modified. You can also use the QQUIT command, which will remove the file from the ring without saving it to disk, regardless of whether it has been modified. With each of these commands, all document windows displaying the file will be closed when the file is removed from the ring.

You can close KEDIT. When you close KEDIT, KEDIT processes each file in the ring, asking whether unsaved changes should be written to disk, and then removing the file from the ring. There are several ways to close KEDIT: you can select the Close menu item from the frame window's system menu, you can double-click on the frame window's system menu, you can press Alt+F4, you can use the File Exit menu item, or you can close Windows itself, which will close KEDIT and all other active applications.

Note the distinction between closing a file and closing a document window. Closing a file, for example by using the File Close menu item or the FILE command, closes all document windows displaying the file and removes the file from the ring. Closing a document window, for example by using the Close menu item on the document window's system menu or by double-clicking on the document window's system menu, closes the current document window, but only removes the file from the ring if there are no other document windows viewing the file; if there are other document windows viewing the file, they remain open and the file remains in the ring.

3.5.2 One-File-Per-Window

By default, KEDIT manages the relationship between files and document windows in the same way as most other Windows MDI (Multiple Document Interface) applications do. This is referred to as ''one-file-per-window'' mode, because each document window is associated with a particular file.

One-file-per-window mode is controlled by SET OFPW. With OFPW ON, you are in one-file-per-window mode. This is the default, and is recommended for most KEDIT users, since it is consistent with the behavior of other Windows applications and seems to behave in the way that most users intuitively expect:

Whenever you begin editing a new file, a new document window is created to display the file, and no other file will ever be displayed in that document window.

Viewing another file involves switching to that other file's own document window. If you use the Next File or Previous File toolbar buttons to cycle through the files

in the ring, you switch to a different document window whenever you switch to a different file.

You can use the Window New menu item to create additional document windows viewing the same file.

If there are multiple document windows viewing a file and you close one of them by using, for example, the Close menu item in the document window's system menu, the other document windows remain open and the file remains in the ring. If there is only one document window for a file and you close it, the document window is closed and the file is removed from the ring.

If you remove a file from the ring by using, for example, the File Close menu item, all document windows displaying the file are closed.

The alternative to one-file-per-window mode is provided by OFPW OFF, which is available primarily for compatibility with text mode versions of KEDIT. In text mode KEDIT, where it is common to edit a large number of files but usually not practical to display more than one or two windows, the relationship between files and windows is different. A document window is not limited, as it is in one-file-per-window mode, to displaying a single file, but can display different files at different times:

When you begin editing a new file with OFPW OFF, a new document window is not created. Instead, the new file is displayed in the current document window. The old file remains in the ring, even if it is no longer being displayed in any document window. Note that files that are not displayed in any document window do not appear in the window list at the bottom of the Window menu.

Viewing a different file need not involve switching to a different document window. If you use the Next File and Previous File buttons on the toolbar to cycle through the files in the ring, the current document window will display each of the files in turn.

You can use the Window New menu item, or the SET SCREEN command, to create additional document windows. These document windows can be used to view any file in the ring, and are not limited to a single file.

If there are multiple document windows and you close one of them by, for example, double-clicking on the document window's system menu, the document window is closed but the file that it was displaying remains in the ring, and can still be accessed via the remaining document windows. However, if there is only a single document window, and you close it, KEDIT also removes all files from the ring (after giving you a chance to save any changed files to disk), since the files would otherwise be stranded, with no way to access them.

If there are multiple files in the ring and you remove a file from the ring by using, for example, the File Close menu item, all document windows remain open, and any document windows that were displaying the file removed from the ring will display the file that preceded it in the ring. However, if there is only a single file in the ring, and you remove it from the ring, all document windows are closed, since no files remain in the ring for them to display.

To summarize the behavior when OFPW OFF is in effect, there is no permanent association between files and windows. Files can be displayed in one or more document windows, and document windows can (at different times) display one or more files. Removing a file from the ring does not cause any document windows to be closed, and closing a document window does not cause any files to be removed from the ring, except that closing the last document window removes all remaining files from the ring, and removing the last file from the ring closes all remaining document windows.

3.6 Fonts

You can use the Options Screen Font dialog box to control the font that KEDIT uses within document windows. A separate Font dialog box, accessible from the File Print dialog box, controls the font used for printing.

As a text editor, KEDIT is more concerned with the content of your files than the details of their appearance. This emphasis is perhaps the primary distinction between a text editor like KEDIT and a word processor like Microsoft Word. The font that you select with Options Screen Font is used for all of the text within your files. That is, you cannot display part of a file in one font and another part of a file in a different font.

Fixed-pitch fonts

Because of KEDIT's emphasis on column-oriented data (with features like the scale line, box blocks, and column commands), KEDIT must display each column of data consistently on the screen. Column 10 of each line, for example, must be lined up horizontally with column 10 of every other line. KEDIT therefore uses only fixed-pitch fonts (in which each character has the same width) to display text in your document windows, as opposed to the proportional fonts (in which different characters can have different widths) used by many other Windows applications.

ANSI and OEM

Most Windows fonts use either the ANSI character set or the OEM character set. The ANSI character set is the character set used by most Windows applications and by Windows itself in most situations. The OEM character set, sometimes referred to as the DOS character set, is used by most DOS text mode applications and is therefore used by Windows for MS-DOS Prompt windows. The Terminal font and fonts with OEM in their name use the OEM character set; most other Windows fonts use the ANSI character set. As discussed below in Section 3.7, "Character Sets", which has more information about the ANSI and OEM character sets, KEDIT works best in most situations with ANSI fonts.

Choosing a font

When you try to decide what font to use with KEDIT, you will probably have a fairly limited selection to choose from. This is because most Windows fonts are proportional fonts, and KEDIT can only use fonts that are marked as fixed pitch. On a typical Windows, the following types of fixed pitch fonts are available:

`Courier New`
: The default font used by KEDIT for Windows is 10 point Courier New. Courier New fonts are fixed-pitch TrueType ANSI fonts. As TrueType fonts, they are scalable to any size, so they are available in a wide variety of point sizes and they work

well with Microsoft's ClearType, a feature of Windows XP and Vista which can improve the appearance of many fonts.

`Courier`
Courier fonts are fixed-pitch bitmapped ANSI fonts. 10 point Courier font was the default font in KEDIT for Windows 1.5 and earlier.

`Terminal`
Terminal fonts are fixed-pitch bitmapped OEM fonts, the same fonts used by Windows for MS-DOS Prompt windows. Note that they are OEM fonts, and ANSI fonts are recommended for most KEDIT users.

Options Screen Font only controls the font used for text within document windows. For all other text displayed by KEDIT, such as the text in dialog boxes, on the status line, and in menus, KEDIT uses standard Windows fonts, most often proportional ANSI fonts, that you cannot change.

Printer fonts

The font used when you print from within KEDIT for Windows is not the same as the font that KEDIT uses on the display. If PRINTER WINDOWS is in effect, the printer font is controlled by a separate Font dialog box accessible from the File Print dialog box. If PRINTER LPT1:, etc., is in effect, the font used depends on your printer's built-in default font.

3.7 Character Sets

3.7.1 Overview

Unicode Not Supported

KEDIT for Windows was designed to be used with the ANSI and OEM character sets discussed below.

KEDIT for Windows does not include support for Unicode, a newer format that provides better support for working with multiple languages and with languages like Japanese that use thousands of different characters.

Unicode is able to handle such large character sets by using multiple bytes to encode a single character. In contrast, the ANSI and OEM character sets use a single byte to encode each character, and can therefore only represent 256 different characters.

Windows 3.1 and Windows 95/98/Me used ANSI as their native character set and provided only very limited support for Unicode. Current versions of Windows still support the ANSI character set, but they use Unicode as their native character set, and an increasing number of Windows applications can work with text stored in Unicode format.

While it would be useful if KEDIT included support for text in Unicode format, this is not a feature that we currently plan to add to KEDIT.

Note that most Windows applications, even applications that include Unicode support, are also able to read and write text in ANSI format, and that programs like Windows Notepad are often able to convert text between ANSI and Unicode format.

ANSI and OEM

KEDIT can make use of two different character sets:

> As mentioned above, the ANSI character set was the native character set in Windows 3.1 and Windows 95/98/Me. The ANSI character set is still supported by current versions of Windows and is compatible with the most current Windows applications.
>
> The OEM character set was used by DOS and by most DOS applications, such as the DOS version of KEDIT. Its main use within Windows is for MS-DOS Command Prompt windows.

For most purposes, U.S. users of KEDIT need not be concerned about the differences between the ANSI and OEM character sets, since they are identical for characters with codes in the range 32-127, which includes all characters on the standard U.S. keyboard.

The ANSI and OEM character sets are very different, however, for characters in the range 128-255. All accented letters fall into this category. The ANSI character set includes a larger variety of accented characters than the OEM character set, but does not include the box drawing characters found in the OEM character set or the special graphic characters with codes below 32. Furthermore, even when the same characters

are present in both character sets, characters above 127 often have different character codes.

This means that text in the OEM character set will be incorrectly displayed if you are using an ANSI font. To display OEM text correctly when you are using the ANSI font, you must convert the text to ANSI. KEDIT has commands, discussed below, that can convert text from OEM to ANSI, and from ANSI to OEM.

Conversions between OEM and ANSI are not always perfect, because each character set has characters not found in the other. Box drawing characters in the OEM character set convert to plus signs, underscores, and the like. Accented characters in the ANSI character set that are not present in the OEM character set convert to their closest equivalents. For example, an accented uppercase character may lose its accent.

Control characters below 32 in the OEM character set correspond to special graphic characters such as smiley faces and musical notes. These characters are not defined in the ANSI character set, and with an ANSI font a dummy character, such as a black rectangle, is displayed whenever they occur. Most frequently-used control characters with codes below 32, such as the backspace, tab, carriage return, linefeed, and formfeed characters, are unaffected by conversions between ANSI and OEM.

ANSI preferred

KEDIT for Windows lets you use either an ANSI or OEM font within document windows, but uses ANSI fonts for all dialog boxes, menus, title bars, and on the status line. If you keep your text in the OEM character set and work with it using an OEM font, most editing operations done within the file area and from the KEDIT command line will work properly, but characters with codes above 127 will not be displayed or entered properly from dialog boxes such as the Edit Find dialog box and the Edit Replace dialog box. For this reason, you will probably get the best results from KEDIT for Windows if you are using an ANSI font and your files are in the ANSI character set.

English language

If, as is the case for most U.S. users of KEDIT, your files are in English and contain no special characters above 127, you can use either character set to work with your files: ANSI fonts, which are normally preferred, or OEM fonts. You need not worry about character set conversion issues.

If your files are in English, but depend on special characters above 127 (for example, the British pound symbol, which has OEM code 156 and ANSI code 163), you will probably want to work with the files in the ANSI character set, converting them if necessary from OEM, as discussed below in connection with other languages.

Files that depend on box drawing characters above 127 will need to be edited using an OEM font, and might be best handled by a DOS text editor. This is because the ANSI character set does not have useful equivalents of most box drawing characters. Similarly, if you depend on the graphic symbols associated in OEM fonts with control characters below 32, you will need to use an OEM font or a DOS text editor. Both the box drawing characters above 127 and the graphic symbols for characters below 32 were made part of the original IBM PC character set because the original PC's were typically text-mode only and had no other way to display graphic symbols or to draw boxes, but their usage is fading in graphical environments like Windows.

Other languages

If you use a language that has accented characters, you will probably want to use the ANSI character set and an ANSI font. Otherwise, accented OEM characters in your file will not be displayed or manipulated properly in KEDIT's dialog boxes, which all use an ANSI font.

If you use an ANSI font such as KEDIT's default 10 point Courier font for new files that you create, and use only KEDIT for Windows or other Windows programs that use the ANSI character set to work with these files, you will not need to worry about character set conversions, since the files will always be in the ANSI character set.

Conversion issues arise when you have an existing file in the OEM character set (such as a file created by the DOS version of KEDIT) and want to edit it with KEDIT for Windows. You will need to convert it from OEM to ANSI in order to display and edit it properly. When you have finished editing the file, you can save it to disk in the ANSI character set and maintain it from then on in the ANSI character set. If you instead want to maintain the file in the OEM character set, you can convert the file back from ANSI to OEM and save it to disk in the OEM character set.

Conversion issues also arise when you create a file in the ANSI character set using KEDIT for Windows, but need to use the file with a DOS program that requires the OEM character set. You can edit such a file with KEDIT for Windows, convert it from the ANSI to the OEM character set, and save it to disk in the OEM character set.

3.7.2 Converting between OEM and ANSI

OEM to ANSI

Conversion from the OEM character set to the ANSI character set is normally done when you are using an ANSI font and you are beginning to edit a file that is in the OEM character set. Again, conversion is only necessary for files that have characters with codes above 127 and is not necessary if your files consist entirely of characters found on the standard U.S. keyboard.

You can normally tell if this conversion is necessary by looking at the file while using an ANSI font within KEDIT for Windows: if most accented characters come out garbled, the file is probably in the OEM character set.

The usual way to convert a file from OEM to ANSI is to issue the command

```
OEMTOANSI ALL
```

from the KEDIT command line. The OEMTOANSI command can convert all or any portion of your file from ANSI to OEM. Since you normally will want to convert the entire file, you will normally specify OEMTOANSI ALL.

ANSI to OEM

You will sometimes be able to convert a file to ANSI and then work with it from then on in ANSI. In other cases, you will need to convert a file back to OEM before saving it to disk, so that the file can be used by DOS programs that require the OEM character set. The ANSITOOEM command handles this conversion. To convert the contents of an entire file from the ANSI character set to the OEM character set, you can use

```
ANSITOOEM ALL
```

Checking the results

Remember that some characters may not convert successfully, because some characters in the ANSI character set are not present in the OEM character set, and vice versa. KEDIT does not control the details of what each character is converted to. Instead, it calls routines within Windows, which do the conversion based on the Windows

language drivers installed on your system. So you should check the results of all OEM to ANSI and ANSI to OEM conversions until you gain confidence that the characters you depend on are handled satisfactorily.

Another reason to check the results of character set conversions is that if you inadvertently use the OEMTOANSI command on text that is already in the ANSI character set, or if you inadvertently use the ANSITOOEM command on text that is already in the OEM character set, you can end up with garbled data in your file. (If you run into this problem you can try to use Edit Undo immediately after the erroneous conversion to reverse its effect.)

To verify that the result of an OEM to ANSI conversion meets your needs, you can view the file in KEDIT for Windows with an ANSI font.

To verify that the result of an ANSI to OEM conversion meets your needs, you can view the file in KEDIT for Windows with an OEM font, or you can view the file with a DOS program, such as the DOS version of KEDIT, that uses the OEM font.

Non-reversible conversions

It is important to understand that converting a file from OEM to ANSI and back again, or from ANSI to OEM and back again, can change the contents of your file if your file uses characters that are not present in both character sets.

Consider, for example, a file in the OEM character set that contains box drawing characters with codes above 127. Since these box drawing characters are not present in the ANSI character set, converting this file from OEM to ANSI would cause these characters to be converted to their closest equivalents, which are underscores, plus signs, etc. Converting this file back from ANSI to OEM would leave these characters unchanged, since underscores and plus signs are present in both the OEM and ANSI character set and Windows has no way of knowing that some of your underscores and plus signs were originally box drawing characters. So converting box drawing characters to ANSI and then back to OEM would leave you with underscores and plus signs, which is not what you started with.

The details of which characters are available in both character sets depend on the DOS code page you are using. In most countries all of the characters represented on the keyboard are available in both character sets and can be freely converted between OEM and ANSI. Problems most often occur with special characters, such as box drawing characters and mathematical symbols.

Advanced conversion options

Advanced users of KEDIT may want to use SET TRANSLATEIN to have KEDIT convert a file from ANSI to OEM as it reads the file in. With the default of TRANSLATEIN NONE, no translation is done. But with TRANSLATEIN OEMTOANSI in effect, KEDIT translates text from OEM to ANSI as it loads new files into the ring and when processing the GET command.

A related command, SET TRANSLATEOUT, can be used to convert a file from ANSI to OEM as it writes the file to disk. With the default of TRANSLATEOUT NONE, no translation is done. But with TRANSLATEOUT ANSITOOEM in effect, KEDIT translates text from ANSI to OEM as it writes files to disk during File Save and related operations and when processing the PUT and PUTD commands.

You should be cautious about using SET TRANSLATEIN and SET TRANSLATEOUT, because OEM to ANSI conversion on a file that is already in ANSI, and ANSI to OEM conversion on a file that is already in OEM, can leave you with a garbled file.

So that it can be in effect before a file is read in, SET TRANSLATEIN is normally issued from your profile. It is unlikely that you would want to automatically convert all files from OEM to ANSI, because at least some of the files that you edit are likely to already be in the ANSI character set. Because of this, the values of the TRANSLATEIN and TRANSLATEOUT options are not saved by Options Save Settings and are most often set via your profile. You would normally issue SET REPROFILE ON in your profile to be sure that your profile is re-executed for each new file added to the ring, and then examine the fileid for the file being edited to decide whether TRANSLATEIN ANSITOOEM should be in effect. For files maintained on disk in the OEM character set, you would also want to put TRANSLATEOUT ANSITOOEM into effect. So your profile would contain something like this:

```
'reprofile on'
/* assume .XXX files are stored in OEM */
if fext.1() = 'XXX' then do
   'set translatein oemtoansi'
   'set translateout ansitooem'
end
```

Notes

KEDIT uses the ANSI character set for the data in DIR.DIR files, and that DIR.DIR files with fileids containing accented characters will be displayed properly only if you are using an ANSI font.

KEDIT for Windows assumes that your macros (that is, .KEX and .KML files) are in the ANSI character set. If you have text mode KEDIT macros that use characters with codes above 127 (these would be legal only within comments and character strings), you may need to convert the macros from the OEM to ANSI character set.

Character set conversions are sometimes necessary when you send data to the printer. For example, if you attempt to print a file that uses the OEM character set with PRINTER WINDOWS in effect and an ANSI printer font selected, accented characters will print incorrectly unless they are converted from OEM to ANSI.

You can use the SET PRINTER command to control whether KEDIT automatically attempts to make any necessary conversions. If SET PRINTER's NOCONVERT option is in effect, no conversion is done. If SET PRINTER's CONVERT option is in effect, as it is by default, and you are using an ANSI screen font and an OEM printer font, or if you are using PRINTER LPT1:, etc., in which case KEDIT assumes that the default printer font is an OEM font, text sent to the printer is converted from ANSI to OEM. If you are using an OEM screen font and an ANSI printer font, text sent to the printer is converted from OEM to ANSI. If either the printer or screen font uses some other character set, such as a character set consisting of special symbols, no conversion is done.

Two functions are provided to allow macro writers to convert strings between the OEM and ANSI character sets within KEXX macros. They are

OEMTOANSI(*string*)

and

ANSITOOEM(*string*)

There is not actually a single OEM character set. Instead, the OEM character set depends on the DOS code page loaded on your system. Most U.S. users of KEDIT work with code page 437, which corresponds to the original IBM PC's character set. But other code pages, such as the multilingual code page 850, are also in common use.

It is also not always true that the native Windows character set is the ANSI character set, although this is the case for most KEDIT for Windows users. Other Windows character sets include the Eastern European, Cyrillic, Greek, and Turkish character sets.

3.8 International Support

KEDIT for Windows is basically an English-language application, in that all of its menus, dialog boxes, and help files are in English. However, KEDIT for Windows does try to adjust to conventions for date and time handling, uppercasing and lowercasing, and sorting that vary depending on your country and language.

KEDIT for Windows does this by examining your Windows settings, as determined by the language drivers installed via the Windows Setup program and by the Windows Control Panel's International settings.

KEDIT for Windows does not include support for Unicode, for right-to-left languages such as Hebrew and Arabic, or for languages with very large character sets, such as Japanese, Chinese, and Korean.

3.8.1 Uppercase and Lowercase

KEDIT's handling of the UPPERCASE and LOWERCASE commands, and of case-insensitive string searches, is affected by the setting of the first operand of SET INTERNATIONAL. This operand can be either NOCASE (the default, which means there is no international uppercase and lowercase handling) or CASE (which means that there is).

When INTERNATIONAL NOCASE is in effect, only the 26 letters of the English alphabet are handled by the UPPERCASE and LOWERCASE commands and given special handling in case-insensitive comparisons.

When INTERNATIONAL CASE is in effect, KEDIT asks Windows which characters are alphabetic and how they are uppercased or lowercased, and handles the UPPERCASE and LOWERCASE commands and case-insensitive comparisons accordingly. This allows accented characters, etc. to be treated properly, according to rules determined by the Windows language drivers installed on your system.

The SORT command is also affected by the value of SET INTERNATIONAL; for the details on this see the description of SET INTERNATIONAL in the Reference Manual.

Note that the handling done by SET INTERNATIONAL assumes that the characters you are working with are in the ANSI character set, and it would not yield the expected results for text that is in the OEM character set. SET INTERNATIONAL therefore has no effect if the current screen font is an OEM font; international case processing is always bypassed in this situation.

Two related KEXX functions are available to macro writers:

ANSIUPPER(*string*) is a KEXX function that uppercases a string in the ANSI character set. Both accented and non-accented alphabetic characters are handled, regardless of the value of SET INTERNATIONAL.

ANSILOWER(*string*) is a KEXX function that lowercases a string in the ANSI character set. Both accented and non-accented alphabetic characters are handled, regardless of the value of SET INTERNATIONAL.

Note that the KEXX UPPER() and LOWER() functions, and uppercasing done by KEXX instructions like PULL and PARSE UPPER, affect only the 26 letters of the English alphabet, regardless of the value of SET INTERNATIONAL.

3.8.2 Date and Time

The date format, as displayed in response to the QUERY TIME command and in DIR.DIR files, is controlled by the Date Format values in the Windows Control Panel's International settings.

The time-of-day format, as displayed on the status line and in response to QUERY TIME, is controlled by the Time Format values in the Windows Control Panel's International settings.

The file times in DIR.DIR files are always based on a 24-hour clock format, to allow the files to be easily sorted according to date and time.

You can use the SET DIRFORMAT command to control whether a 2- or 4-digit year (for example, "95" or "1995") is displayed in DIR.DIR files.

The values returned in macros by the KEXX DATE() and TIME() functions are in a standard format that does not depend on your country or language. However, options on these functions allow you to access the date and time in several different formats, and the DATECONV() function allows you to convert back and forth between the date formats.

3.9 The DIR.DIR File

The DIR.DIR file is a special file that you can create within KEDIT to view a directory listing. You can sort the contents of the DIR.DIR file by date, by size, or by the name or

extension of the files involved. You can also select a file from within a DIR.DIR listing and have KEDIT add it to the ring of files that you are editing.

For each file, the DIR.DIR listing gives the drive involved, the file name and extension, the size of the file, the date and time that the file was last modified, and finally the file's directory path. The DIR.DIR listing can include both files and subdirectories; subdirectories are indicated by the string "<dir>" appearing in place of the file size.

Creating a DIR.DIR file

You can use the File Directory dialog box to create a DIR.DIR file. Open the File Directory dialog box, select the drive and directory whose contents you want to examine, select the Show DIR.DIR File radio button, and then click on OK.

You can also create a DIR.DIR file by issuing the DIR command from KEDIT's command line. If you issue the DIR command with no operands, you will get a listing of the current directory. But you can specify a different directory, and can specify file names and extensions, which can include asterisks ("*") and question marks ("?") as wildcard characters.

Some examples:

```
DIR
```

List all files in the current directory.

```
DIR D:\TEMP
```

List all files in the D:\TEMP directory.

```
DIR *.C *.H
```

List all files in the current directory that have an extension of .C or .H.

If a DIR.DIR file is already in the ring when you use File Directory or the DIR command to get a directory listing, the new directory listing replaces the existing contents of the DIR.DIR file. You can use the DIRAPPEND command, which accepts the same operands as the DIR command, if you instead want to add information to an existing DIR.DIR file.

It is possible to save the DIR.DIR file to disk, but this is normally not done. Instead, you let KEDIT create the DIR.DIR file in memory, and when you have finished with it, you close it by using, for example, the File Close menu item or the QUIT command. Because the DIR.DIR file is a special file that is normally never saved to disk, KEDIT gives it special handling: it is not autosaved or locked, regardless of the settings of SET AUTOSAVE or SET LOCKING, and even if you have made changes to it, File Close and the QUIT command treat the DIR.DIR file as if it were an unmodified file, and immediately remove it from the ring.

Sorting DIR.DIR files

The SET DEFSORT command controls the order in which the DIR.DIR file is initially sorted. By default, DIR.DIR is sorted according to the name and extension of the files involved.

When a DIR.DIR file is the current file, KEDIT's default toolbar changes to include buttons that allow you to re-sort the DIR.DIR file by name, extension, size, or date. You can also re-sort DIR.DIR files by issuing the DIRSORT command from the KEDIT command line.

Editing files

To edit one of the files listed in a DIR.DIR file, double-click on the line describing the file you want to edit. Alternatively, with the default key definitions, you can place the cursor on the line describing the file you want to edit and press Alt+X.

If you double-click on or press Alt+X for a line of the DIR.DIR file that describes a subdirectory, KEDIT will replace the current DIR.DIR file with a listing of the contents of that subdirectory. You can use the Parent Directory button on the DIR.DIR file's toolbar button, or you can press Shift+Ctrl+X, to get a listing of the contents of a file's parent directory.

Displaying long filenames

KEDIT for Windows can work with the long filenames supported by all current versions of Windows. The display of long filenames in DIR.DIR files is affected by the SET DIRFORMAT command, whose first two operands control the amount of space set aside in DIR.DIR files for file names and for file extensions. (The third operand of SET DIRFORMAT controls whether KEDIT uses 2 or 4 digits for the year in the date field of DIR.DIR files.) By default, KEDIT sets aside 30 columns in DIR.DIR files for filenames and 10 columns for file extensions. This corresponds to the command

```
SET DIRFORMAT 30 10
```

As a special case, you can specify 0 as the value for file extensions. This causes KEDIT to display the name and extension together as a unit in the columns normally set aside for the file name.

3.10 Printing

To print your file from within KEDIT, you can use the File Print dialog box. You can print either your entire file, or a marked block within your file. File Print also lets you determine the printer margins and printer font that KEDIT will use when printing your file.

When you send syntax-colored text to a color printer, KEDIT will normally print it in color.

Note that KEDIT's printer font, controlled through the File Print dialog box, is separate from KEDIT's screen font, which is controlled through Options Screen Font. One reason that KEDIT does not automatically use the same font on the screen and on the printer is that some screen fonts are not available as printer fonts. Another reason is that the font that looks best on your screen is not necessarily the font that looks best on your printer, and the font size that works best on your monitor may be larger or smaller than what you would like for your printer.

You can also print your file by using the Print File button on the toolbar. By default, the Print File button brings up the File Print dialog box, so that you can choose whether to

print the entire file or the currently marked block, can change your printer font, etc. You can instead choose to have the Print File toolbar button print your file immediately, without displaying the File Print dialog box; simply uncheck the box at the bottom of the File Print dialog box that is labeled ‘‘Print File Toolbar Button Shows This Dialog’’. The Print File button will then cause KEDIT to print the marked block within your file if there is one, and to otherwise print your entire file.

Another way to print all or part of your file is to issue the PRINT command from the command line. For example, to print 200 lines of your file, beginning with the current line, you could issue the command

```
PRINT 200
```

Printer output is normally sent through your Windows printer drivers, and you choose the particular printer to be used via the File Printer Setup dialog box. With the SET PRINTER command you can instead specify that the your print output will go directly to a printer port, such as LPT1: or LPT2:, and will bypass the normal Windows printer handling. This is useful primarily if you have files that contain device-dependent printer escape codes, which are not handled properly by the device-independent printer handling used when the Windows printer drivers are used. Note that when you are bypassing the Windows printer drivers, you cannot use the File Print dialog box to control the printer margins or fonts; default fonts and margins built into your printer are used unless the data that you print includes device-dependent printer control codes that change these defaults.

3.11 Word Processing Facilities

KEDIT is primarily a text editor and does not provide the full text formatting facilities found in most word processing programs. For example, KEDIT uses only fixed-pitch fonts and does not support proportional fonts, or the use of multiple fonts within a document. KEDIT does, however, have a useful subset of the functions typically associated with ‘‘word processing’’, sufficient for working with simple notes and memos, and with e-mail. KEDIT has a wordwrap feature to assist text entry, and control keys and commands to format and justify paragraphs within margins, center text between margins, and left- and right-adjust text.

3.11.1 Margins

To make use of KEDIT’s word processing capabilities, you should know how to set the margins used by KEDIT. The *left margin* value determines where the leftmost column of text will be placed by KEDIT and the *right margin* value determines where the rightmost column of text will be placed. For example, if you set the left margin to 10 and the right margin to 70, KEDIT will know that you want text within paragraphs to be placed between columns 10 and 70 of your file, with blanks in columns 1 to 9 and beyond column 70. You use the *paragraph indent* value to tell KEDIT what column the first line of a paragraph should start in, either as an absolute column number or as an offset relative to the left margin.

To change the margin settings, you can issue the SET MARGINS command from the KEDIT command line, or you can specify the margins via the Options SET Command dialog box.

The three operands of the SET MARGINS command are the desired left margin, right margin, and paragraph indent values. For example,

```
SET MARGINS 5 60 9
```

tells KEDIT that you want a left margin of 5, a right margin of 60, and that you want new paragraphs to begin in column 9.

```
SET MARGINS 5 60 +4
```

would initially have the same effect, telling KEDIT that you want a left margin of 5, a right margin of 60, and that you want new paragraphs to begin four columns to the right of column 5, which is column 9.

Although the two examples above seem at first to be equivalent, a difference shows up if you later change the margins. Suppose you issued the command

```
SET MARGINS 10 70
```

after originally specifying the paragraph indent as 9. From then on, the left margin will be 10, the right margin will be 70, and new paragraphs will still begin in column 9. However, if you originally specified the paragraph indent as +4, the left margin will be 10, the right margin will be 70, and new paragraphs will begin in column 14—four columns to the right of the left margin.

The default margin setting is 1 72 +0, which means that the left margin is column 1, the right margin is column 72, and new paragraphs start in the same column as the left margin.

You are free to enter text within or outside of the current margin settings; the margin settings have an effect only when you use the word processing features described in the following sections. Simply changing the margin settings does not have any immediate effect on the contents of your file.

Displaying margins

You can use the SET BOUNDMARK command to create a visual reminder of the left and right margin settings. KEDIT will draw vertical lines in the document window just before the left margin column and just after the right margin column if you issue the command

```
SET BOUNDMARK MARGINS
```

Note that additional SET BOUNDMARK options, which can be specified instead of or in addition to the MARGINS operand, control whether lines are drawn for the ZONE, TRUNC and VERIFY columns, for tab columns, and for the window margin area.

3.11.2 Wordwrap

The wordwrap feature is off by default. You can enable it by putting WORDWRAP ON into effect via Options SET Command.

Wordwrapping allows you to enter text into your file without concern for text going beyond the right margin. When you enter text with WORDWRAP OFF, you have to be aware as you enter text what column the cursor is in. When you are about to enter a word that won't fit within your margins, you must instead add a new line and start the next word at the left margin column of the new line.

With WORDWRAP ON, you can simply enter the text that you want and not worry about your margins. When you attempt to enter a word that goes beyond the right margin, KEDIT adds a new line for you, moves the part of the word you've already typed to the left margin column of the new line, and places the cursor on the new line in position for the next character to be typed. This is all automatic and happens fast enough for you to continue typing without missing a beat. You can enter a full paragraph at a time without having to press any special keys to start new lines and without having to look at the screen to worry about how close you're getting to the right margin.

3.11.3 Starting a New Paragraph

Shift+Ctrl+P is one of six word processing operations assigned to keys. All six are invoked by pressing the Shift and Ctrl keys in combination with an alphabetic character.

Shift+Ctrl+P is used to begin a new paragraph. It causes KEDIT to add a blank line to your file (to separate the old paragraph from the new one), and then to add a second blank line with the cursor positioned in the paragraph indent column, ready for you to enter the first line of the new paragraph.

Note that Shift+Ctrl+F, the paragraph formatting operation described below, normally assumes that all paragraphs are separated from each other by at least one blank line. Shift+Ctrl+P inserts this blank line for you.

3.11.4 Adjusting Text

Several Shift+Ctrl key combinations are useful for adjusting text.

Shift+Ctrl+C centers a line of text. When you press Shift+Ctrl+C, the text is centered between the left and right margins. For example, if the left margin is 1 and the right margin is 67 and the text on the line consists of the word ''Hello'', Shift+Ctrl+C will place the ''Hello'' in columns 32 through 36 of the line, exactly halfway between the left and right margins. You can also use the CENTER command to center text.

Shift+Ctrl+R right-adjusts the text in a line. The rightmost nonblank character of the text is placed in the right margin column. Again assuming left and right margin settings of 1 and 67 and text consisting of ''Hello'', Shift+Ctrl+R will move the ''Hello'' to

columns 63 through 67. You can also use the RIGHTADJUST command to right-adjust text.

Shift+Ctrl+L left-adjusts the text in a line. The leftmost nonblank character of text is placed in the left margin column. In our example, the word ‘‘Hello’’ will be placed in columns 1 through 5. You can also use the LEFTADJUST command to left-adjust text.

Shift+Ctrl+A adjusts text on the cursor line to wherever the cursor is located. The leftmost nonblank character of the text is placed in the same column as the cursor. In our example, if the cursor was in column 18 and you press Shift+Ctrl+A, the word ‘‘Hello’’ will be placed in columns 18 through 22.

3.11.5 Formatting Text

You can format a paragraph by pressing Shift+Ctrl+F, which issues the FLOW command. This causes KEDIT to rearrange text within a paragraph so that it fits as neatly as possible within the current margin settings.

How FLOW works

The first word of the paragraph is placed starting at the paragraph indent column of the first line of the paragraph. Then as many words as will fit on the line without going beyond the right margin are also placed on the line. Then the second line of the paragraph is built up, starting from the left margin column and continuing as long as the text doesn’t overflow the right margin column. This process is repeated until KEDIT has placed the entire paragraph properly within the current margins.

During the formatting process, words at the end of one line might move to the beginning of the next line, or words from the beginning of one line might fit before the right margin of the previous line. Shift+Ctrl+F does not affect the content of the words in a paragraph, or their order. It simply re-divides the text into lines that fit within the current margin settings.

Pressing Shift+Ctrl+F to format a paragraph causes the paragraph to look as if you had just entered it with WORDWRAP ON. You can use this function to ‘‘neaten up’’ paragraphs that you entered with WORDWRAP OFF or paragraphs that you entered with WORDWRAP ON but which you have changed because, for example, you inserted or deleted some words. If you have used the SET MARGINS command to change your margin settings, you can also use Shift+Ctrl+F to reformat existing paragraphs within your new margins.

As far as KEDIT is concerned, a paragraph is any group of nonblank lines of text. KEDIT assumes that you have separated paragraphs from each other with blank lines. (When entering text, using Shift+Ctrl+P to start new paragraphs will cause the blank line to be automatically inserted.)

If you press Shift+Ctrl+F with the cursor on a blank line, KEDIT assumes that you want to format a paragraph starting at the next nonblank line. Otherwise, the paragraph in which the cursor is located is formatted. After formatting a paragraph, KEDIT moves to the blank line following the newly-formatted paragraph. Simply pressing Shift+Ctrl+F will then cause the next paragraph to be formatted. An easy way to format a group of paragraphs is to press Shift+Ctrl+F repeatedly.

You might not want to format all of your text. You may have tables, lists, or headings that should not be reformatted to look like paragraphs. This need not be a problem, since Shift+Ctrl+F formats only a paragraph at a time, not your entire document. You can use Shift+Ctrl+F selectively, formatting text that should be formatted and moving past text that shouldn't.

Justifying text

An additional KEDIT formatting function is enabled if you issue the command

```
SET FORMAT JUSTIFY
```

After you issue this command, Shift+Ctrl+F will justify all paragraphs that it formats. This means that KEDIT will sprinkle enough extra blanks between words in the paragraph to make each line in the paragraph end exactly at the right margin. Text will be evenly lined up in the right margin column and will not have the "ragged right" format that you get with FORMAT NOJUSTIFY, the default, in effect. Text justification does not take place when you enter text, regardless of the settings of FORMAT or WORDWRAP. Justification only occurs when you format a paragraph with Shift+Ctrl+F or the FLOW command.

SET FORMAT has other operands that allow you to control how KEDIT determines paragraph boundaries and how many spaces are put at the end of a sentence when paragraphs are reformatted.

3.12 Syntax Coloring

When you use KEDIT to edit a computer program, it is often useful to make different types of text within the program, such as keywords, quoted strings, and comments stand out by displaying them in different colors. This makes it easier, for example, to tell which text is part of a comment and which text is part of the program, and to tell whether a string is properly quoted. This capability is available in KEDIT, and is referred to as the *syntax coloring* facility.

In addition to showing the text in color on your display, KEDIT will print syntax-colored text in color when you print to a color printer with PRINTER WINDOWS and PRINTCOLORING ON in effect.

Built into KEDIT are *parsers* that process the following languages: C and C++, REXX and KEXX, HTML, Java, COBOL, FORTRAN, BASIC, Pascal, C#, and dBase. Syntax coloring is applied by default to the following extensions:

Extension	Parser	Description
.BAS	BASIC	BASIC language
.C	C	C/C++ language
.CBL	COBOL	COBOL language
.COB	COBOL	COBOL language
.COBOL	COBOL	COBOL language

Extension	Parser	Description
.CPP	C	C/C++ language
.CS	CSHARP	C# language
.CXX	C	C/C++ language
.DLG	RESOURCE	Windows Resource file
.FOR	FORTRAN	FORTRAN language
.FORTRAN	FORTRAN	FORTRAN language
.FRM	BASIC	Used with Visual BASIC
.F90	FORTRAN	FORTRAN language
.F	FORTRAN	FORTRAN language
.H	C	C/C++ language
.HPP	C	C/C++ language
.HXX	C	C/C++ language
.HTM	HTML	HyperText Markup Language
.HTML	HTML	HyperText Markup Language
.INI	INIT	INI file
.JAV	JAVA	Java language
.JAVA	JAVA	Java language
.KEX	REXX	REXX/KEXX language
.KLD	KLD	KEDIT Language Definition file
.KML	REXX	REXX/KEXX language
.DPR	PASCAL	Delphi Project file
.DPK	PASCAL	Delphi Package file
.DPR	PASCAL	Delphi Project file
.PRG	XBASE	dBase or similar language
.RC	RESOURCE	Windows Resource file
.REX	REXX	REXX/KEXX language

Nesting

One unique aspect of the syntax coloring facility is its ability to handle nested parentheses and similar items. For example, in a statement like

```
x = (b + (c * (d + 2)))
```

there are three levels of parentheses, and each level is displayed in a different color. This makes it easier to understand the logic of an expression, and easier to see if the parentheses are not properly matched.

In C and C++ programs, syntax coloring also uses different colors for nested braces and for nested preprocessor commands like #ifdef and #endif. Nested control structures in languages like REXX are also highlighted. For example, in a REXX program like the following:

```
do i = 1 to 10
    do j = 1 to 10
       say i + j
    end
end
```

The outer DO—END pair would be displayed in a different color than the inner DO—END pair.

A related command that can be very useful is the CMATCH command, which is assigned by default to Shift+F3, You can use Shift+F3 to move the cursor between matching items, like matching parentheses, and matching DO—END pairs.

Accuracy

It is important to understand that the parsers that handle syntax coloring are not as complete as the parsers built into a typical compiler. Syntax coloring operates very quickly, processing text in a fairly simple-minded way, without building symbol tables, processing header files, or checking for errors in your text. The goal is to be as efficient as possible, handling normal situations correctly, but accepting that in some unusual cases, especially in files that contain syntax errors, text may be colored incorrectly.

Controlling syntax coloring

KEDIT's syntax coloring facility is user configurable. The details of each language are specified in KEDIT Language Definition files that you can change by, for example, adding your own keywords. You can also develop your own KEDIT Language Definition files to support additional languages. KEDIT Language Definition files are discussed in detail in Reference Manual Chapter 8, "KEDIT Language Definition Files".

The following commands are used to control the syntax coloring facility:

SET AUTOCOLOR tells KEDIT that files of a specified extension should automatically be colored using the specified parser. For example

```
SET AUTOCOLOR .FOR FORTRAN
```

tells KEDIT to use the FORTRAN parser for all files with an extension of .FOR.

SET COLORING turns coloring on or off for a particular file, and controls which language-specific parser to use, or specifies that a default parser determined by SET AUTOCOLOR should be used.

SET PARSER lets you define your own language-specific parser. You give the name of the parser you want to define and the name of a KEDIT Language Definition file (with an extension of .KLD) that contains rules describing how to parse your language. For example

```
SET PARSER LANG MYLANG.KLD
```

tells KEDIT that, whenever the SET COLORING or SET AUTOCOLOR commands are used to specify that a file should be colored using a parser called LANG, the coloring should be done according to the rules in the file MYLANG.KLD.

SET ECOLOR controls the colors that the syntax coloring facility uses to highlight keywords, numbers, comments, etc. on your display.

SET PRINTCOLORING determines whether KEDIT uses color or black and white when sending syntax-colored text to a color printer.

SET PCOLOR controls the colors used to print syntax-colored text on a color printer when PRINTCOLORING ON is in effect.

3.13 The Undo Facility

This section discusses KEDIT's undo facility. The commands involved are:

UNDO, available through the Edit Undo menu item, the Undo toolbar button, and the Ctrl+Z key combination, which will undo the most recent change to a file.

REDO, available through the Edit Redo menu item, the Redo toolbar button, and the Ctrl+Y key combination, which will redo a change after an undo.

SET UNDOING, which controls the amount of undo information saved by KEDIT.

The basics

KEDIT is a powerful text editor that makes it easy to change a file in many different ways. Sometimes you will make a change and then decide that it was a mistake. You might delete some text and then have second thoughts about it, reword a paragraph and then not be happy with the results, or issue a CHANGE command that affects more text than you expected.

Because of this, KEDIT has an undo facility that lets you reverse the effects of most changes to your file. KEDIT keeps an internal log of all changes that you make, and by using this information, KEDIT is able to restore lines that were changed, add back lines that were deleted, and delete lines that were added.

To undo the effect of a change, you can simply click on the Undo toolbar button or press Ctrl+Z, both of which issue the UNDO command. To undo several changes, click the Undo button or press Ctrl+Z repeatedly.

If you undo too many changes, you can use the Redo toolbar button or press Ctrl+Y, repeatedly if necessary, to redo changes, reversing the effect of previous UNDO commands. REDO is only available from the time that you use the UNDO command to the time that you make a further change to your file.

In some cases you can undo hundreds of changes to your file, each possibly affecting many lines of your file. But the undo facility needs to keep an in-memory log of your changes, and the size of this internal log is affected by the value of the SET UNDOING

option. So while you can undo most changes to your file, especially small changes, you should understand that memory limitations do sometimes come into play.

Undo levels

While KEDIT tracks changes to your file in its internal log, it groups the changes into ''undo levels'', which can be undone or redone as a unit. For example, if you issue a CHANGE command that affects 10 lines in your file, the 10 changed lines form one undo level, and clicking on the Undo button on the toolbar or pressing Ctrl+Z will undo the changes to all 10 lines. When you run a macro, all changes made while the macro is executing also form a single undo level.

Undo only keeps track of changes to your file on a line-at-a-time basis. If, for example, you add a line to your file, move the cursor to the new line, and type in a line of new text, you cannot individually undo the entry of each character that you typed. All editing changes made to a single line are normally grouped by KEDIT into a single undo level, and can only be undone as a unit.

Status line

Information about the undo facility is displayed on the status line at the bottom of the frame window. In the third box within the status line, KEDIT displays ''Alt='' followed by three numbers. The first two numbers indicate the number of changes to your file since the last autosave, and since the last save. The third number indicates the number of undo levels available for the UNDO command. Additionally, after you have issued one or more UNDO commands, the third number is followed by an asterisk for as long as it is possible to use the REDO command. For example,

```
Alt=2,10;5
```

would indicate that 2 changes had been made since the last autosave, 10 changes since the last save, and that 5 levels of changes could be undone by the UNDO command.

```
Alt=2,10;4*
```

would indicate that 4 levels of changes were available to the UNDO command, and that it was possible to REDO the effect of one or more previous UNDO commands.

What you can undo

In the course of an editing session, you are likely to issue many KEDIT commands. Some of these commands, like the CHANGE command or the DELETE command, affect the contents of your file. Other commands, such as the LOCATE command or the SET ZONE command, affect your position within a file or affect some aspect of how KEDIT will process your file, but do not make any change to the contents of the file. It is important to understand that the undo facility does not deal with the effects of all KEDIT commands, but only with commands like CHANGE and DELETE that affect the contents of your file. Undo keeps track of lines that have been added, deleted, or changed. It also keeps track of data closely tied to individual lines, such as the selection levels used with the ALL command, the tag bits set with the TAG command, and the line names used with the SET POINT command. Undo does not keep track of the contents of the command line or prefix area, or of changes in the focus line location, the cursor position, the screen layout, or the values of SET options.

For example, assume that you issue the following commands from the command line:

```
DELETE 3
SET ZONE 10 20
SET COLOR FILEAREA GREEN
ADD 5
UNDO
UNDO
```

The first UNDO command will delete from your file the 5 lines inserted by the ADD command. The second UNDO command will add back to your file the 3 lines removed by the DELETE command. Since the SET COLOR command did not change the contents of your file, it is not handled by undo, and despite the UNDO commands, the file area ends up displayed in green. Similarly, even though the DELETE command that precedes it is undone, the SET ZONE command is not, and ZONE 10 20 remains in effect.

SET UNDOING

The SET UNDOING command controls, on a per-file basis, whether the undo facility is on or off, the maximum number of undo levels that KEDIT will attempt to keep, and the maximum amount of memory that will be used to hold undo information.

By default, KEDIT keeps a maximum of 512K of undo information in memory for each file that you are editing, and tries to save 200 undo levels per file. These default values are adequate for most purposes. You can, however, use the SET UNDOING command to specify higher or lower values. For example, if you want to save up to 300 levels of changes and allow up to a 2048K of memory for a file's undo log, you would use the following command:

```
SET UNDOING ON 300 2048
```

QUERY UNDOING gives you the value of UNDOING for the current file: whether the undo facility is turned on, the maximum number of undo levels that can be kept, and the maximum amount of memory that the undo log can occupy.

QUERY UNDO can also be useful. It gives additional information about the status of the undo facility for the current file. The first is the number of undo levels available for the UNDO command (this is the same as the third number after ''Alt='' on the status line). Next is the number of undo levels available for the REDO command (this will be nonzero when the third number after ''Alt='' is followed by an asterisk). Third is the amount of memory (in kilobytes) currently being used for the current file's undo log.

Notes

Saving the undo information does not noticeably slow down most KEDIT operations. The main overhead is in the memory used to hold the undo log. Commands or macros that change large portions of your data can naturally generate large amounts of undo information. However, this does not cause problems for normal editing operations. If the limits controlled by SET UNDOING are reached, it will automatically throw away undo information as necessary to free up memory for normal command processing.

While you are editing a file with KEDIT, you are actually working with an in-memory copy of the file. The version of the file on disk is updated only when you issue a FILE or SAVE command. The UNDO command only affects the

in-memory copy of your file, and does not affect the version of the file on disk. An example:

```
CHANGE /Y/Z/ ALL *
SAVE
UNDO
```

The CHANGE command changes all ‘‘Y’’s in the in-memory copy of your into ‘‘Z’’s. The SAVE command writes this version of the file to disk. The UNDO command changes the ‘‘Z’’s back into ‘‘Y’’s in the in-memory copy of the file, but has no effect on the disk version of the file, which still has only ‘‘Z’’s.

Undo keeps separate track of each file that you are editing. When you issue the UNDO command, KEDIT will only undo changes to the current file. For example, if you move 10 lines from file A to file B, the undo facility sees this as the deletion of 10 lines from file A and the addition of 10 lines to file B. If, after the move, you make file A the current file and then issue the UNDO command, KEDIT will restore the 10 deleted lines to file A, but file B will be unaffected. Similarly, if you make file B the current file and issue the UNDO command, KEDIT will delete the 10 lines from file B without affecting file A.

If you issue the UNDO command repeatedly, it will work backwards, restoring the contents of your file to what they were at earlier and earlier points in time. If you undo too many changes, you can use the REDO command to move forward, restoring the contents to what they were at later and later points in time. But note that UNDO and REDO cannot be used selectively. If you make 10 changes to your file, you can UNDO the tenth and then the ninth and then the eighth, etc. You cannot, however, UNDO only the eighth change, or UNDO back to the first change and then REDO all but the fifth change.

When you issue the UNDO command, it is not always easy to see exactly what lines have been affected. KEDIT does try to restore the current line and cursor position in effect when the change was made, but some of the lines affected might be off the screen, and there could have been small changes to many lines. To give you some help, the UNDO and REDO commands display counts of the number of lines affected. But these counts are not always exact. If, for example, you run a macro that changes only one line, but makes 10 separate changes to it, UNDO will tell you that 10 lines have been affected.

Chapter 4. Keyboard and Mouse

This chapter describes KEDIT's default keyboard assignments and mouse actions. It primarily covers keystrokes and mouse actions used within KEDIT document windows and does not focus on accessing menus and toolbars, which are covered in a separate chapter.

CUA and Classic Interfaces

KEDIT supports two sets of keyboard and mouse conventions, controlled through the Options Interface dialog box or by the SET INTERFACE command. INTERFACE CUA, the default setting, provides conventions compatible with other Windows applications, while INTERFACE CLASSIC makes KEDIT for Windows work like earlier text mode versions of KEDIT.

Since keyboard and mouse usage differs significantly depending on whether INTERFACE CUA or INTERFACE CLASSIC is in effect, the two cases are discussed separately. KEDIT's CUA interface is described first, followed by KEDIT's CLASSIC interface, and this is followed by a summary of the differences between the two.

4.1 Using the CUA Interface

With the default of INTERFACE CUA in effect, KEDIT for Windows supports many of the mouse and keyboard conventions of other Windows applications. These conventions were originally based on IBM's Common User Access (CUA) guidelines, but the version adopted in KEDIT comes mainly from Microsoft's guidelines as of the time that the main design work was done on KEDIT for Windows in the mid-1990s. Note that you can use the Options Interface dialog box to adjust some details of KEDIT's keyboard and mouse behavior when INTERFACE CUA is in effect.

4.1.1 Moving the Cursor

This section covers positioning the cursor within a file while INTERFACE CUA is in effect.

Using the Keyboard

The following table lists keyboard methods of repositioning the cursor:

To move the cursor	Press
Left one character	Cursor Left
Right one character	Cursor Right
Up one line	Cursor Up
Down one line	Cursor Down
To the beginning of the line	Home or F7

To move the cursor	Press
To the end of the line	End
One page backward	Page Up
One page forward	Page Down
To the next word on a line	Ctrl+Cursor Right
To the previous word on a line	Ctrl+Cursor Left
To the beginning of the file	Ctrl+Home
To the end of the file	Ctrl+End
To the upper left of the window	Ctrl+Page Up
To the bottom right of the window	Ctrl+Page Down
To the beginning of the next line	Ctrl+Enter
To the next Tab column	Tab (in Overtype Mode) or F4
To the previous Tab column	Shift+Tab
To the command line	F12 or Numeric Pad Plus
To the current line	Shift+F2

Using the Mouse

To position the cursor with the mouse:

1. If necessary, use the scrollbars to reach the part of the file where you want to position the cursor.

2. Move the mouse pointer to the desired cursor location and click mouse button 1.

4.1.2 Entering and Editing Text

This section describes keyboard methods for inserting, replacing, and deleting individual characters and lines of text, and for undoing and redoing changes that you make to your file while INTERFACE CUA is in effect.

Inserting characters in a line of text

1. The status line should have INS as the Insert/Overtype indicator. If the indicator shows OVR, press the Insert key to toggle from Overtype Mode to Insert Mode. (By default, the cursor is thicker when you are in Insert Mode and thinner when you are in Overtype Mode.)

2. Position the cursor where you want to insert text and begin to type. Characters that you type will be inserted into the file, shifting existing characters to the right.

Overtyping characters in a line of text

1. The status line should have OVR as the Insert/Overtype indicator. If the indicator shows INS, press the Insert key to toggle Insert Mode to Overtype Mode.

2. Position the cursor where you want to overtype text and begin to type.

Typing replaces selection

1. Select text by dragging with mouse button 1 or using Shift+Cursor keys.

2. If you are currently in Overtype Mode, press the Insert key to switch from Overtype Mode to Insert Mode. Typing-replaces-selection, in KEDIT and in other Windows applications, works out ‘‘right’’ only when you are in Insert Mode.

3. Type the replacement text, which will replace the selected text.

Inserting and manipulating lines of text

To	Press
Add a new line at the cursor position	Enter (with cursor in file area)
Add a new line below the focus line	F2
Duplicate a line	F8 or Alt+Equal Sign
Split a line	Alt+S
Join a line	Alt+J
If beyond end of line, join lines Otherwise, split line	F11
Left-adjust a line to left margin column	Shift+Ctrl+L or Ctrl+L
Right-adjust a line to right margin column	Shift+Ctrl+R or Ctrl+R
Center a line within margin columns	Shift+Ctrl+C
Adjust a line to cursor position	Shift+Ctrl+A
Format text of paragraph within margins	Shift+Ctrl+F
Begin a new paragraph by adding two lines, moving to paragraph indent column of second	Shift+Ctrl+P

Deleting text

To	Press
Delete the character at the cursor position	Delete
Delete the character to the left of the cursor	Backspace
Delete the word at the cursor position	Shift+Ctrl+W
Delete the line at the cursor position	Alt+D

To	Press
Delete from the cursor position to the end of the line	Ctrl+Delete
Delete selection	Backspace or Delete (when text is selected)
Delete persistent block	Backspace or Delete (immediately after marking block) or Alt+G
Cut selection or persistent block to the clipboard	Ctrl+X or Shift+Delete

Undoing and redoing changes

To	Press
Undo the last change to the file	Ctrl+Z or Alt+Backspace
Redo (reverse the effect of an undo)	Ctrl+Y or Ctrl+Backspace
Undo recent typing within a line	Escape
Recover the last changed or deleted line	Alt+R

4.1.3 Selecting Text

With INTERFACE CUA in effect, KEDIT supports three types of ''selections'' (which are sometimes referred to as non-persistent blocks, to distinguish them from the persistent blocks that KEDIT also supports): stream selections, line selections, and box selections.

Stream selections involve all text from some starting character through some ending character and, as in most other Windows applications, are marked by dragging with mouse button 1 or pressing Shift+Cursor key. Line selections involve all text in a consecutive group of lines. Box selections involve a rectangular section of text within your file, for example the text in columns 10 through 20 of lines 50 through 80 of your file.

KEDIT's selections work much like selections in other Windows applications, in that they are unmarked as soon as you reposition the cursor and they support typing-replaces-selection. Within a file, you can mark line, box, or stream selections. You can also mark command line selections; these are always stream selections.

Using the Keyboard

The following table lists keyboard methods of selecting text. In all cases, if text is already selected, the existing selection is extended from its anchor point (that is, the original starting point) of the selection. If text is not already selected, a new stream selection is begun, extending from the cursor position.

To Extend a Selection	Press
Left one character	Shift+Cursor Left
Right one character	Shift+Cursor Right
Up one line	Shift+Cursor Up
Down one line	Shift+Cursor Down
To the beginning of the line	Shift+Home
To the end of the line	Shift+End
One page backward	Shift+Page Up
One page forward	Shift+Page Down
To the next word on a line	Shift+Ctrl+Cursor Right
To the previous word on a line	Shift+Ctrl+Cursor Left
To the beginning of the file	Shift+Ctrl+Home
To the end of the file	Shift+Ctrl+End
To the upper left of the window	Shift+Ctrl+Page Up
To the bottom right of the window	Shift+Ctrl+Page Down

To select the entire file

Press Ctrl+A or Ctrl+Numeric Pad 5 key to mark the entire file as a line selection.

To unmark a selection

To unmark a selection with the keyboard, press any cursor-movement key.

To delete a selection

Press the Delete key or the Backspace key.

Using the Mouse

By default, the mouse marks selections when INTERFACE CUA is in effect. However, you can use the Options Interface dialog box or the SET MARKSTYLE command to specify that the mouse should mark persistent blocks rather than selections.

All text selection with the mouse involves button 1. Button 2 does not select text, but instead displays a pop-up menu of useful actions taken from the Edit menu.

Using the mouse to select text with the default Options Interface settings in effect:

To	Do This
Select text (stream selection)	Drag with mouse button 1
Select a word (stream selection)	Double-click button 1
Select one word at a time (stream selection)	Double-click and drag with mouse button 1

To	Do This
Mark a line selection	Drag with Ctrl+mouse button 1, or drag with mouse button 1 in the window margin area or in the prefix area
Mark a box selection	Drag with Alt+mouse button 1
Mark the entire file as a line selection	Click Ctrl+button 1 in the window margin area or in the prefix area
Extend selection to the mouse pointer	Click Shift+button 1
Unmark a selection	Click button 1
Delete a selection	Click button 2 and choose Delete from the pop-up menu

The "window margin area" used for marking line selections is an area a few pixels wide at the left edge of the document window. When the mouse pointer is over the window margin area, it changes to an arrow pointing to the upper right. The window margin area lets you mark lines without having to simultaneously press the Ctrl key, which you must do when you drag with button 1 in the file area to mark lines. You can use the SET WINMARGIN command to control whether this feature is enabled and how many pixels wide the window margin will be.

4.1.4 Marking Persistent Blocks

With INTERFACE CUA in effect, you can use Windows-style selections (that is, non-persistent blocks), but KEDIT also supports persistent line, box, and stream blocks. Unlike selections, persistent blocks remain marked, regardless of any typing or cursor movement, until you specifically unmark them.

A persistent line block, the most frequently used type of persistent block, is a set of consecutive lines of text. A persistent box block (sometimes referred to as a "column block") is a rectangular area of text in your file. For example, the text in columns 10 through 20 of lines 50 through 80 of your file might make up a box block. A persistent stream block is a stream of consecutive characters that can span one or more lines. A persistent stream block would typically be a phrase, sentence, or group of sentences that you would like to work with as a unit.

Using the Keyboard

To mark a persistent line block

1. Move the cursor to one end of the group of lines involved.
2. Press Alt+L.
3. Move the cursor to the other end of the group of lines.
4. Press Alt+L again.

To mark a persistent box block

1. Move the cursor to one corner of the box.
2. Press Alt+B.
3. Move the cursor to the other corner of the box.
4. Press Alt+B again.

To mark a persistent stream block

1. Move the cursor to one end of the stream of characters.
2. Press Alt+Z.
3. Move the cursor to the other end of the stream of characters.
4. Press Alt+Z again.

To unmark a persistent block

Press Alt+U.

To delete a persistent block

Press Alt+G

To convert a selection to a persistent block

Press Ctrl+M

Using the Mouse

By default, the mouse marks selections when INTERFACE CUA is in effect and does not directly mark persistent blocks. However, you can mark a persistent block by first marking a selection with the mouse and then clicking mouse button 2 and choosing Make Persistent from the resulting pop-up menu.

You can also use the Options Interface dialog box or the SET MARKSTYLE command to specify that the mouse should mark persistent blocks rather than selections.

Assuming that the default Options Interface settings are in effect and that the mouse does not directly mark persistent blocks:

To	Use
Mark a persistent block	Use the mouse to mark a selection Click mouse button 2 Choose Make Persistent from the pop-up menu
Extend a persistent block to the mouse pointer	Click Shift+button 1

To	Use
Unmark a persistent block	Click Alt+button 1 or Click button 2 and choose Unmark from the pop-up menu
Delete a persistent block	Click button 2 and choose Delete from the pop-up menu

4.1.5 Moving and Copying Text

With INTERFACE CUA in effect, there are two ways to move or copy blocks or selections from one location to another: through the clipboard and through drag-and-drop editing. A third method, that works only with persistent blocks, is also discussed.

Clipboard Method

The clipboard method lets you move text within a file, between files, and even between different applications. It also works with text on the KEDIT command line.

1. Use the mouse or keyboard to select the text that you plan to move or copy.
2. For a move operation, cut the text to the clipboard by using the Edit Cut menu item, the Cut to Clipboard toolbar button, or Ctrl+X.

   For a copy operation, copy the text to the clipboard by using the Edit Copy menu item, the Copy to Clipboard toolbar button, or Ctrl+C.
3. Position the cursor at the desired new location of the text.
4. Paste the text from the clipboard to its new location by using the Edit Paste menu item, the Paste from Clipboard toolbar button, or Ctrl+V.

Drag and Drop Method

The drag-and-drop method works only for moving text within a single file, and does not work between files or with text on the command line.

1. Use the mouse or keyboard to select the text that you plan to move or copy.
2. For a move operation, position the mouse pointer inside the selected text and, with mouse button 1 down, drag the mouse pointer to the desired new location of the text.

   For a copy operation, position the mouse pointer inside the selected text, press the Ctrl key and, with the Ctrl key still down, press mouse button 1 and drag the mouse pointer to the desired new location of the text.
3. Release mouse button 1. The text will be moved or copied to the mouse pointer location.

Once you have begun a drag-and-drop operation, you can cancel it by moving the mouse pointer back inside the selected text and then releasing the mouse button, or by pressing the Escape key before releasing the mouse button.

Persistent Blocks

The clipboard and drag-and-drop methods used to move or copy selections will also work with persistent blocks. In addition, you can use the following method to move or copy a persistent block within a file or between files:

1. Use the mouse or keyboard to mark the block involved.
2. Position the cursor at the desired new location of the text.
3. To move the text to the new location, use the Move Block button on the bottom toolbar, or press Alt+M. (The Move Block button leaves the block marked, while Alt+M unmarks it.)

   To copy the text to the new location, use the Copy Block button on the bottom toolbar or press Alt+C or Alt+K. (The Copy Block button and Alt+K leave the copied block marked, while Alt+C unmarks it.)

4.1.6 Other Block Operations

In addition to moving and copying blocks, KEDIT provides other useful block operations, most of which are accessible with either the keyboard or the mouse. This section describes how to perform these operations when you are using the CUA interface.

Using the Keyboard

The following table lists keyboard methods of accessing block operations:

To	Press
Delete a block	Delete or Backspace (immediately after marking block), or Alt+G
Uppercase a block Lowercase a block	Shift+F5 Shift+F6
Shift a block left Shift a block right	Shift+F7 Shift+F8
Fill a block	Ctrl+I
Overlay text at cursor position with block contents	Shift+Ctrl+O
Unmark a block	Alt+U

Using the Mouse

Most block operations done with the mouse involve accessing the menu or toolbar:

To	Use
Delete a block	Delete Block button on bottom toolbar, or click mouse button 2 and choose Delete from the pop-up menu
Uppercase a block Lowercase a block	Actions Uppercase menu item Actions Lowercase menu item or Uppercase Block button on bottom toolbar Lowercase Block button on bottom toolbar
Shift a block left Shift a block right	Shift Block Left button on bottom toolbar Shift Block Right button on bottom toolbar
Left-adjust a block Right-adjust a block	Leftadjust Block button on bottom toolbar Rightadjust Block button on bottom toolbar
Fill a block	Actions Fill menu item, or Fill Block button on bottom toolbar
Overlay text at cursor position with block contents	Overlay Block button on bottom toolbar

4.1.7 Menus, Files, and Windows

This section covers methods for accessing menus, moving between files and between windows, etc., while using the CUA interface.

Using the Keyboard

Keyboard methods:

To	Press
Cycle to the next file in the ring	Shift+F4
Cycle to next document window Cycle to previous document window	Ctrl+F6 or Ctrl+Tab Shift+Ctrl+F6 or Shift+Ctrl+Tab
Cycle to next Windows application	Alt+Tab
Maximize document window	Ctrl+F10
Restore document window	Ctrl+F5
Close document window	Ctrl+F4
Maximize frame window	Alt+F10
Restore frame window	Alt+F5
Close KEDIT	Alt+F4
Close current file	File Close menu item

To	Press
Close current file if it is unmodified	F3
Enter menu mode	Alt, F10, or Shift+F11
Leave menu mode	Escape
Access frame window's system menu	Alt+Spacebar
Access document window's system menu	Alt+Minus or Alt+Numeric Pad Minus
Activate Windows Task Manager	Ctrl+Escape
Access File menu	Alt+F
Access Edit menu	Alt+E
Access Actions menu	Alt+A
Access Options menu	Alt+O
Access Window menu	Alt+W
Access Help menu	Alt+H
Use shortcut for File New	Ctrl+N
Use shortcut for File Open	Ctrl+O
Use shortcut for File Print	Ctrl+P
Use shortcut for File Save	Ctrl+S
Use shortcut for Edit Select All	Ctrl+A
Use shortcut for Edit Make Persistent	Ctrl+M
Use shortcut for Edit Find	Ctrl+F
Use shortcut for Edit Replace	Ctrl+H
Use shortcut for Edit Go To	Ctrl+G
Use shortcut for Actions Bookmark	Ctrl+B

Using the Mouse

In addition to the items in the following table, the standard Windows conventions for accessing menus and for moving and resizing windows are all available:

To	Use
Cycle to next/previous file in the ring	Next File or Previous File button on the toolbar
Move between document windows	Click in a different document window, or use the Window menu
Close KEDIT	Double-click on frame window system menu
Close document window	Double-click on document window system menu
Close current file	File Close menu item
Open new view of current file	Window New menu item
Maximize/restore frame window	Double-click on frame window title bar
Maximize document window	Double-click on document window title bar
Restore document window	Click on restore icon at the right of maximized document window's menu bar

4.1.8 Command Line and Prefix Area

This section lists keyboard methods of accessing KEDIT's command line and prefix area while using the CUA interface.

Command line

Working with the command line:

To	Press
Move to the command line from the file area (also executes any pending prefix commands)	F12 or Numeric Pad Plus
Move to the file area from the command line	Cursor Up or Cursor Down
Toggle between the file area and the command line	Shift+F12
Execute commands on the command line	Enter, with the cursor on the command line
Cycle backward through previously issued commands	Ctrl+Cursor Up or F6
Cycle forward through previously issued commands	Ctrl+Cursor Down
Repeat last command issued from the command line	F9
Repeat last LOCATE command	Shift+F1

Prefix area

When PREFIX ON is in effect, you can use the following to work with the prefix area:

To	Press
Move to the command line and execute pending prefix commands (which may reposition cursor)	F12 or Numeric Pad Plus
Tab forward between file area and prefix area	Tab
Tab backward between file area and prefix area	Shift+Tab
Toggle between a line's prefix and file area, executing pending prefix commands	Ctrl+Numeric Pad Enter or Shift+Ctrl+Enter

4.1.9 Miscellaneous

The following actions, relating to the CUA interface, do not fit neatly into any of the other categories:

To	Press
Interrupt execution of a long-running command or macro	Hold down Ctrl+Break or Alt+Ctrl+Shift
Activate Help system	F1
Scroll to make the focus line become the current line	F5
Begin editing file named at cursor position of non-DIR.DIR file	Alt+X
Begin editing a file listed on the focus line of a DIR.DIR file	Alt+X or double-click with mouse button 1
List the files in a directory listed on the focus line of a DIR.DIR file	Alt+X or double-click with mouse button 1
List the files in the parent directory of a file listed on the focus line of a DIR.DIR file	Shift+Ctrl+X or Parent Directory toolbar button
Set Bookmark1 on the focus line	Alt+1 or Set Bookmark1 button on bottom toolbar
Set Bookmark2 on the focus line	Alt+2
Set Bookmark3 on the focus line	Alt+3
Go to Bookmark1	Alt+4 or Go to Bookmark1 button on bottom toolbar
Go to Bookmark2	Alt+5
Go to Bookmark3	Alt+6

To	Press
Toggle between viewing the entire file and viewing selected lines	Alt+Numeric Pad Plus
Search forward for the Quick Search string	Alt+F1
Search backward for the Quick Search string	Alt+F2
Go to Quick Search toolbar item	Alt+F3
Locate matching brace	Shift+F3
Scroll half-window to the left	Shift+F9
Scroll half-window to the right	Shift+F10
Display mouse button 2 pop-up menu	Application key on Windows-specific keyboards

4.2 Summary of CUA Interface

This section summarizes the keyboard assignments and mouse actions used within a document window when INTERFACE CUA is in effect, as it is by default.

Cursor area

Keyboard assignments for cursor and numeric pad keys:

Key	Action
Cursor Up	Cursor up one line
Cursor Down	Cursor down one line
Cursor Left	Cursor left one character
Cursor Right	Cursor right one character
Home	Cursor to beginning of line (adjustable via Options Interface dialog box)
End	Cursor to end of line
Page Up	Backward one page in the file
Page Down	Forward one page in the file
Insert	Toggle Insert/Overtype Mode
Delete	Delete character at cursor position or delete selection (adjustable via Options Interface dialog box)
Ctrl+Cursor Up	Cycle forward in command line history
Ctrl+Cursor Down	Cycle backward in command line history

Key	Action
Ctrl+Cursor Left	Cursor to previous word on a line
Ctrl+Cursor Right	Cursor to next word on a line
Ctrl+Home	Cursor to beginning of file
Ctrl+End	Cursor to end of file
Ctrl+Page Up	Cursor to upper left of window
Ctrl+Page Down	Cursor to bottom right of window
Ctrl+Insert	Copy text to clipboard
Ctrl+Delete	Delete text from the cursor position to the end of the line
Ctrl+Numeric Pad 5	Mark entire file as line selection
Shift+Cursor Up	Extend selection up one line
Shift+Cursor Down	Extend selection down one line
Shift+Cursor Left	Extend selection one character left
Shift+Cursor Right	Extend selection one character right
Shift+Home	Extend selection to start of line
Shift+End	Extend selection to end of line
Shift+Page Up	Extend selection one window backward
Shift+Page Down	Extend selection one window forward
Shift+Insert	Paste text from clipboard
Shift+Delete	Cut text to clipboard
Shift+Ctrl+Cursor Left	Extend selection to previous word on a line
Shift+Ctrl+Cursor Right	Extend selection to next word on a line
Shift+Ctrl+Home	Extend selection to beginning of file
Shift+Ctrl+End	Extend selection to end of file
Shift+Ctrl+Page Up	Extend selection to upper left of window
Shift+Ctrl+Page Down	Extend selection to lower right of window
Numeric Pad Plus	Cursor to command line
Alt+Numeric Pad Plus	Toggle between viewing entire file and viewing selected lines
Alt+Numeric Pad Minus	Access document window’s system menu
Numeric Pad Enter	On command line: execute command In file area: Insert new line (adjustable via Options Interface dialog box)

Key	Action
Ctrl+Numeric Pad Enter	Toggle between file area and prefix area, executing any pending prefix commands
Ctrl+Break	Press and hold to interrupt long-running command or macro
Alt+Numeric Pad digits	Enter special characters: Alt+*nnn* for OEM code (which is then converted to ANSI) Alt+0*nnn* to avoid code conversion

Typewriter area

Keyboard assignments for keys in the typewriter area:

Key	Action
Enter	On command line: execute command In file area: Insert new line (adjustable via Options Interface dialog box)
Ctrl+Enter	Move to start of next line
Shift+Ctrl+Enter	Toggle between file area and prefix area, executing any pending prefix commands
Tab	In Insert Mode: Insert spaces until next Tab column In Overtype Mode: Move cursor to next Tab column If PREFIX ON: Tab forward to file area or prefix area
Shift+Tab	If PREFIX OFF: Move cursor to previous Tab column If PREFIX ON: Tab backward to file area or prefix area
Ctrl+Tab	Cycle to next document window
Shift+Ctrl+Tab	Cycle to previous document window
Alt+Tab	Cycle to next Windows application
Backspace	Delete character to left of cursor or delete selection (adjustable via Options Interface dialog box)
Ctrl+Backspace	Redo last undo operation
Alt+Backspace	Undo a change to the file
Escape	In Menu Mode: leave menu mode Otherwise: Undo recent typing on a line
Ctrl+Escape	Invoke Windows Task Manager
Alt+Spacebar	Access frame window's system menu
Alt	Activate Menu Mode (adjustable via Options Interface dialog box)

Key	Action
Alt+Shift+Ctrl	Press and hold to interrupt long-running command or macro (works in some cases where Ctrl+Break does not)
Ctrl+A	Shortcut for Edit Select All
Ctrl+B	Shortcut for Actions Bookmark
Ctrl+C	Copy text to clipboard
Ctrl+F	Shortcut for Edit Find
Ctrl+G	Shortcut for Edit Go To
Ctrl+H	Shortcut for Edit Replace
Ctrl+I	Fill block with a specified character
Ctrl+L	Left-adjust text of focus line to left margin column
Ctrl+M	Shortcut for Edit Make Persistent
Ctrl+N	Shortcut for File New
Ctrl+O	Shortcut for File Open
Ctrl+P	Shortcut for File Print
Ctrl+R	Right-adjust text of focus line to right margin column
Ctrl+S	Shortcut for File Save
Ctrl+V	Paste text from clipboard
Ctrl+X	Cut text to clipboard
Ctrl+Y	Redo last undo operation
Ctrl+Z	Undo a change to the file
Alt+A	Open the Actions menu
Alt+B	Mark box block
Alt+C	Copy block
Alt+D	Delete focus line
Alt+E	Open the Edit menu
Alt+F	Open the File menu
Alt+G	Delete block
Alt+H	Open the Help menu
Alt+J	Join two lines
Alt+K	Copy block and leave block marked
Alt+L	Mark line block
Alt+M	Move block
Alt+O	Open the Options menu

Key	Action
Alt+R	Recover a changed or deleted line
Alt+S	Split a line
Alt+U	Unmark block
Alt+W	Open the Window menu
Alt+X	Edit file named at cursor position or in DIR.DIR directory listing
Alt+Z	Mark stream block
Alt+1	Set Bookmark1 at focus line
Alt+2	Set Bookmark2 at focus line
Alt+3	Set Bookmark3 at focus line
Alt+4	Go to Bookmark1
Alt+5	Go to Bookmark2
Alt+6	Go to Bookmark3
Alt+Minus	Access document window's system menu
Alt+Equal Sign	Duplicate focus line
Shift+Ctrl+A	Adjust text to cursor position
Shift+Ctrl+C	Center focus line within margins
Shift+Ctrl+F	Format text in paragraph within margins
Shift+Ctrl+L	Left-adjust text of focus line to left margin column
Shift+Ctrl+O	Overlay text with contents of block
Shift+Ctrl+P	Begin a new paragraph
Shift+Ctrl+R	Right-adjust text of focus line to right margin column
Shift+Ctrl+W	Delete word
Shift+Ctrl+X	Display parent directory of file in DIR.DIR listing
Application key on Windows-specific keyboards	Display mouse button 2 pop-up menu

Function keys

Keyboard assignments for function keys:

Key	Action
F1	Access Help system
F2	Add new line below focus line
F3	QUIT current file

Key	Action
F4	Cursor to next Tab column
F5	Focus line becomes current line
F6	Cycle backward in command line history
F7	Cursor to beginning of line
F8	Duplicate focus line
F9	Reissue last command executed from current file's command line
F10	Enter Menu Mode
F11	Split/join line at cursor position
F12	Cursor to command line If PREFIX ON: Cursor to command line and execute pending prefix commands
Shift+F1	Reissue last LOCATE command
Shift+F2	Cursor to current line
Shift+F3	Find matching brace
Shift+F4	Cycle to next file in the ring
Shift+F5	Uppercase block
Shift+F6	Lowercase block
Shift+F7	Shift block one character to left
Shift+F8	Shift block one character to right
Shift+F9	Scroll half-window to the left
Shift+F10	Scroll half-window to the right
Shift+F11	Enter Menu Mode
Shift+F12	Toggle cursor between command line and file area
Ctrl+F1	Undo last change to file
Ctrl+F2	Redo last undo operation
Ctrl+F4	Close document window
Ctrl+F5	Restore maximized document window to normal size
Ctrl+F6	Cycle to next document window
Ctrl+F10	Maximize document window
Alt+F1	Search forward for Quick Search string
Alt+F2	Search backward for Quick Search string
Alt+F3	Move to Quick Search toolbar item
Alt+F4	Close KEDIT

Key	Action
Alt+F5	Restore maximized frame window to normal size
Alt+F10	Maximize frame window
Shift+Ctrl+F6	Cycle to previous document window

Mouse Actions

Here is a summary of mouse actions available by default within a document window. You can use Options Interface or the SET MARKSTYLE command to control whether the mouse marks selections (which is the default behavior) or persistent blocks. In addition, the normal Windows conventions for accessing menus, moving and resizing windows, etc. all apply:

Mouse Operation	Action
Click button 1	Reposition the cursor and unmark any selection
Click Alt+button 1	Unmark a persistent block
Click Shift+button 1	Extend a block to the mouse pointer position
Double-click button 1	Select a word (stream selection)
Drag with button 1	Select characters (stream selection)
Drag with Ctrl+button 1 Drag in margin area with button 1 Drag in prefix area with button 1	Mark a line selection
Click Ctrl+button 1 in margin area or prefix area	Mark entire file as line selection
Drag with Alt+button 1	Mark a box selection
Position mouse pointer within selection or persistent block and drag with button 1	Drag-and-drop block move
Position mouse pointer within selection or persistent block and drag with Ctrl+button 1	Drag-and-drop block copy
Double-click button 1 on entry in DIR.DIR file	Edit the file or list the directory
Click button 2	Display pop-up menu with Cut, Copy, Paste, Delete, Unmark, and Make Persistent items from Edit menu

4.3 Using the Classic Interface

With INTERFACE CLASSIC in effect, KEDIT's keyboard and mouse behavior is very close to that of the text mode KEDIT, even where this means that Windows conventions are not adhered to. For example, in most Windows programs, Alt+W will access the Window menu and the Home key will move the cursor to the beginning of a line. In KEDIT for Windows with INTERFACE CLASSIC, Alt+W deletes a word and the Home key moves the cursor to the command line.

If you are moving from the text mode version of KEDIT to the Windows version, and are accustomed to the text mode key assignments, you may want to use INTERFACE CLASSIC. As you use more and more Windows applications and you get used to the Windows conventions, you may eventually want to switch to INTERFACE CUA. If you are a new KEDIT user who is not already familiar with KEDIT's text mode interface, you will probably want to use INTERFACE CUA from the start.

All of the default keyboard assignments from text mode KEDIT are available with INTERFACE CLASSIC. Additionally, where there is no conflict with these text mode compatible keys, the default key assignments from the CUA interface are also available. For example, Ctrl+F10 has no default assignment in text mode KEDIT, but will maximize a document window in KEDIT for Windows, with both INTERFACE CUA and INTERFACE CLASSIC.

4.3.1 Moving the Cursor

This section covers positioning the cursor within a file while INTERFACE CLASSIC is in effect.

Using the Keyboard

The following table lists keyboard methods of repositioning the cursor:

To move the cursor	Press
Left one character	Cursor Left
Right one character	Cursor Right
Up one line	Cursor Up
Down one line	Cursor Down
To the beginning of the line	F7
To the end of the line	End
One page backward	Page Up
One page forward	Page Down
To the next word on a line	Ctrl+Cursor Right
To the previous word on a line	Ctrl+Cursor Left
To the beginning of the file	Ctrl+Page Up

To move the cursor	Press
To the end of the file	Ctrl+Page Down
To the upper left of the window	Ctrl+Home
To the beginning of the next line	Enter or Numeric Pad Enter
To the next Tab column	Tab or F4
To the previous Tab column	Shift+Tab
To the command line	Home
To the current line	Shift+F2

Using the Mouse

To position the cursor with the mouse:

1. If necessary, use the scrollbars to reach the part of the file where you want to position the cursor.

2. Move the mouse pointer to the desired cursor location and click mouse button 1.

4.3.2 Entering and Editing Text

This section describes keyboard methods for inserting, replacing, and deleting individual characters and lines of text, and for undoing and redoing changes that you make to your file while INTERFACE CLASSIC is in effect.

Inserting characters in a line of text

1. The status line should have INS as the Insert/Overtype indicator. If the indicator shows OVR, press the Insert key to toggle from Overtype Mode to Insert Mode. (By default, the cursor is thicker when you are in Insert Mode and thinner when you are in Overtype Mode.)

2. Position the cursor where you want to insert text and begin to type. Characters that you type will be inserted into the file, shifting existing characters to the right.

Overtyping characters in a line of text

1. The status line should have OVR as the Insert/Overtype indicator. If the indicator shows INS, press the Insert key to toggle Insert Mode to Overtype Mode.

2. Position the cursor where you want to overtype text and begin to type.

Inserting and manipulating lines of text

To	Press
Add a new line	F2
Duplicate a line	F8 or Alt+Equal Sign
Split a line	Alt+S
Join a line	Alt+J
If beyond end of line, join lines Otherwise, split line	F11
Left-adjust a line to left margin column	Ctrl+L
Right-adjust a line to right margin column	Ctrl+R
Center a line within margin columns	Ctrl+C
Adjust a line to cursor position	Ctrl+A
Begin a new paragraph by adding two lines, moving to paragraph indent column of second	Ctrl+P
Format text of paragraph within margins	Ctrl+F

Deleting text

To	Press
Delete the character at the cursor position	Delete
Delete the character to the left of the cursor	Backspace
Delete the word at the cursor position	Alt+W
Delete the line at the cursor position	Alt+D
Delete from the cursor position to the end of the line	Ctrl+End
Delete block	Alt+G
Cut block to the clipboard	Shift+Delete

Undoing and redoing changes

To	Press
Undo the last change to the file	Alt+Backspace or Ctrl+Z
Redo (reverse the effect of an undo)	Ctrl+Backspace or Ctrl+Y
Undo recent typing within a line	Escape
Recover the last changed or deleted line	Alt+R

4.3.3 Marking Blocks

KEDIT supports three types of blocks: line blocks, box blocks, and stream blocks. With INTERFACE CLASSIC, all blocks are persistent blocks and non-persistent blocks are not available. (Non-persistent blocks are also referred to as ''selections''. They are used instead of persistent blocks in many Windows applications and are available in KEDIT for Windows if INTERFACE CUA is in effect.) Persistent blocks remain marked, regardless of any typing or cursor movement, until you specifically unmark them.

A line block, the most frequently used type of block, is a set of consecutive lines of text. A box block (sometimes referred to as a ''column block'') is a rectangular area of text in your file. For example, the text in columns 10 through 20 of lines 50 through 80 of your file might make up a box block. A stream block is a stream of consecutive characters that can span one or more lines. A stream block would typically be a phrase, sentence, or group of sentences that you would like to work with as a unit.

Using the Keyboard

To mark a line block

1. Move the cursor to one end of the group of lines involved.
2. Press Alt+L.
3. Move the cursor to the other end of the group of lines.
4. Press Alt+L again.

To mark a box block

1. Move the cursor to one corner of the box.
2. Press Alt+B.
3. Move the cursor to the other corner of the box.
4. Press Alt+B again.

To mark a stream block

1. Move the cursor to one end of the stream of characters.
2. Press Alt+Z.
3. Move the cursor to the other end of the stream of characters.
4. Press Alt+Z again.

To unmark a block

Press Alt+U.

Using the Mouse

Marking blocks with the mouse:

To	Use
Mark a line block	Drag with button 2, or drag with button 1 in the window margin area or in the prefix area
Mark a box block	Drag with both button 1 and button 2
Mark a stream block	Drag with button 1
Extend a block to the mouse pointer	Click Shift+button 1
Mark a word (as a stream block)	Double-click button 1
Mark the entire file (as a line block)	Click Ctrl+button 1 in the window margin area or in the prefix area
Unmark a marked block	Click button 2

The ''window margin area'' used for marking line blocks is an area a few pixels wide at the left edge of the document window. When the mouse pointer is over the window margin area, it changes to an arrow pointing to the upper right. You can use the SET WINMARGIN command to control whether this feature is enabled and how many pixels wide the window margin will be.

4.3.4 Moving and Copying Text

Using the Classic interface, there are two ways to move or copy blocks from one location to another: through the clipboard and through Move and Copy commands accessed via the keyboard or toolbar.

Clipboard Method

The clipboard method lets you move text within a file, between files, and even between different applications.

1. Use the mouse or keyboard to mark the text that you plan to move or copy.
2. For a move operation, cut the text to the clipboard by using the Edit Cut menu item, the Cut to Clipboard toolbar button, or Shift+Delete.

   For a copy operation, copy the text to the clipboard by using the Edit Copy menu item, the Copy to Clipboard toolbar button, or Ctrl+Insert.
3. Position the cursor at the desired new location of the text.
4. Paste the text from the clipboard to its new location by using the Edit Paste menu item, the Paste from Clipboard toolbar button, Shift+Insert.

MOVE and COPY Commands

You can also use the MOVE and COPY commands, which are assigned to keys and to toolbar buttons, to move or copy a block of text:

1. Use the mouse or keyboard to mark the block involved.

2. Position the cursor at the desired new location of the text.

3. To move the text to the new location, use the Move Block button on the bottom toolbar, or press Alt+M. The Move Block button leaves the block marked, while Alt+M unmarks it.

   To copy the text to the new location, use the Copy Block button on the bottom toolbar or press Alt+C or Alt+K. The Copy Block button and Alt+K leave the copied block marked, while Alt+C unmarks it.

4.3.5 Other Block Operations

In addition to moving and copying blocks, KEDIT provides other useful block operations, most of which are accessible with either the keyboard or the mouse when using the Classic interface..

Using the Keyboard

The following table lists keyboard methods of accessing block operations:

To	Press
Delete a block	Alt+G
Uppercase a block Lowercase a block	Shift+F5 Shift+F6
Shift a block left Shift a block right	Shift+F7 Shift+F8
Fill a block	Alt+F or Ctrl+I
Overlay text at cursor position with block contents	Alt+O or Shift+Ctrl+O
Unmark a block	Alt+U

Using the Mouse

Block operations done with the mouse all involve accessing the menu or toolbar:

To	Use
Delete a block	Delete Block button on bottom toolbar
Uppercase a block Lowercase a block	Actions Uppercase menu item Actions Lowercase menu item or Uppercase Block button on bottom toolbar Lowercase Block button on bottom toolbar
Shift a block left Shift a block right	Shift Block Left button on bottom toolbar Shift Block Right button on bottom toolbar
Left-adjust a block Right-adjust a block	Leftadjust Block button on bottom toolbar Rightadjust Block button on bottom toolbar

To	Use
Fill a block	Actions Fill menu item, or Fill Block button on bottom toolbar
Overlay text at cursor position with block contents	Overlay Block button on bottom toolbar
Unmark a block	Click button 2

4.3.6 Menus, Files, and Windows

This section covers methods for accessing menus, moving between files and between windows, etc., while using the Classic interface.

Using the Keyboard

Keyboard methods:

To	Press
Cycle to the next file in the ring	Shift+F4
Cycle to next document window Cycle to previous document window	Ctrl+F6 or Ctrl+Tab Shift+Ctrl+F6 or Shift+Ctrl+Tab
Cycle to next Windows application	Alt+Tab
Maximize document window	Ctrl+F10
Restore document window	Ctrl+F5
Close document window	Ctrl+F4
Maximize frame window	Alt+F10
Restore frame window	Alt+F5
Close KEDIT	Alt+F4
Close current file	File Close menu item
Close current file if it is unmodified	F3
Enter menu mode	Shift+F11
Leave menu mode	Escape
Access frame window's system menu	Alt+Spacebar
Access document window's system menu	Alt+Minus or Alt+Numeric Pad Minus
Activate Windows Task Manager	Ctrl+Escape
Access Edit menu	Alt+E
Access Help menu	Alt+H
Access other menus	Use mouse or press Shift+F11 to enter Menu Mode

To	Press
Use shortcut for File New	Ctrl+N
Use shortcut for File Open	Ctrl+O
Use shortcut for File Save	Ctrl+S
Use shortcut for Edit Replace	Ctrl+H
Use shortcut for Edit Go To	Ctrl+G
Use shortcut for Actions Bookmark	Ctrl+B

Using the Mouse

In addition to the items in the following table, the standard Windows conventions for accessing menus and for moving and resizing windows are all available:

To	Use
Cycle to next/previous file in the ring	Next File or Previous File button on the toolbar
Move between document windows	Click in a different document window, or use the Window menu
Close KEDIT	Double-click on frame window system menu
Close document window	Double-click on document window system menu
Close current file	File Close menu item
Open new view of current file	Window New menu item
Maximize/restore frame window	Double-click on frame window title bar
Maximize document window	Double-click on document window title bar
Restore document window	Click on restore icon at the right of maximized document window's menu bar

4.3.7 Command Line and Prefix Area

This section lists keyboard methods of accessing KEDIT's command line and prefix area while using the Classic interface.

Command line

Working with the command line:

To	Press
Move to the command line from the file area (also executes any pending prefix commands)	Home
Move to the file area from the command line	Cursor Up or Cursor Down

To	Press
Toggle between the file area and the command line	F12
Execute commands on the command line	Enter, with the cursor on the command line
Cycle backward through previously issued commands	Ctrl+Cursor Up or F6
Cycle forward through previously issued commands	Ctrl+Cursor Down
Repeat last command issued from the command line	F9
Repeat last LOCATE command	Shift+F1

Prefix area

When PREFIX ON is in effect, you can use the following to work with the prefix area:

To	Press
Move to the command line and execute pending prefix commands (which may reposition cursor)	Home
Tab forward between file area and prefix area	Tab
Tab backward between file area and prefix area	Shift+Tab
Toggle between a line's prefix and file area, executing pending prefix commands	Ctrl+Enter or Numeric Pad Plus

4.3.8 Miscellaneous

The following actions, relating to the Classic interface, do not fit neatly into any of the other categories:

To	Press
Interrupt execution of a long-running command or macro	Ctrl+Break or Alt+Ctrl+Shift
Activate Help system	F1
Scroll to make the focus line become the current line	F5
Begin editing file named at cursor position of non-DIR.DIR file	Alt+X
Begin editing a file listed on the focus line of a DIR.DIR file	Alt+X or double-click with mouse button 1
List the files in a directory listed on the focus line of a DIR.DIR file	Alt+X or double-click with mouse button 1
List the files in the parent directory of a file listed on the focus line of a DIR.DIR file	Ctrl+X or Parent Directory toolbar button

To	Press
Set Bookmark1 on the focus line	Alt+1 or Set Bookmark1 button on bottom toolbar
Set Bookmark2 on the focus line	Alt+2
Set Bookmark3 on the focus line	Alt+3
Go to Bookmark1	Alt+4 or Go to Bookmark1 button on bottom toolbar
Go to Bookmark2	Alt+5
Go to Bookmark3	Alt+6
Toggle between viewing the entire file and viewing selected lines	Alt+Numeric Pad Plus
Search forward for the Quick Search string	Alt+F1
Search backward for the Quick Search string	Alt+F2
Go to Quick Search toolbar item	Alt+F3
Locate matching brace	Shift+F3
Scroll half-window to the left	Shift+F9
Scroll half-window to the right	Shift+F10
Display pop-up menu of actions from Edit menu	Application key on Windows-specific keyboards

4.4 Summary of Classic Interface

This section summarizes the default keyboard assignments and mouse actions used within a document window when INTERFACE CLASSIC is in effect.

Cursor area

Keyboard assignments for cursor and numeric pad keys:

Key	Action
Cursor Up	Cursor up one line
Cursor Down	Cursor down one line
Cursor Left	Cursor left one character
Cursor Right	Cursor right one character
Home	Cursor to command line (also executed any pending prefix commands)

Key	Action
End	Cursor to end of line
Page Up	Backward one page in the file
Page Down	Forward one page in the file
Insert	Toggle Insert/Overtype Mode
Delete	Delete character at cursor position
Ctrl+Cursor Up	Cycle backward in command line history
Ctrl+Cursor Down	Cycle forward in command line history
Ctrl+Cursor Left	Cursor to previous word on a line
Ctrl+Cursor Right	Cursor to next word on a line
Ctrl+Home	Cursor to upper left of window
Ctrl+End	Delete text through end of line
Ctrl+Page Up	Cursor to beginning of file
Ctrl+Page Down	Cursor to end of file
Ctrl+Insert	Copy text to clipboard
Ctrl+Delete	Delete text through end of line
Ctrl+Numeric Pad 5	Mark entire file as line block
Shift+Insert	Paste text from clipboard
Shift+Delete	Cut text to clipboard
Numeric Pad Plus	If PREFIX ON, toggle between prefix area and file area, executing any pending prefix commands
Alt+Numeric Pad Plus	Toggle between viewing entire file and viewing selected lines
Alt+Numeric Pad Minus	Access document window's system menu
Numeric Pad Enter	On command line: execute command In file area: cursor to beginning of next line
Ctrl+Numeric Pad Enter	Toggle between file area and prefix area,, executing any pending prefix commands
Ctrl+Break	Press and hold to interrupt long-running command or macro
Alt+Numeric Pad digits	Enter special characters: Alt+*nnn* for OEM code (which is then converted to ANSI) Alt+0*nnn* to avoid code conversion

Typewriter area

Keyboard assignments for keys in the typewriter area:

Key	Action
Enter	On command line: execute command In file area: Move to beginning of next line
Ctrl+Enter Shift+Ctrl+Enter	Toggle between file area and prefix area, executing any pending prefix commands
Tab	If PREFIX OFF: Move cursor to next Tab column If PREFIX ON: Tab forward to file area or prefix area
Shift+Tab	If PREFIX OFF: Move cursor to previous Tab column If PREFIX ON: Tab backward to file area or prefix area
Ctrl+Tab	Cycle to next document window
Shift+Ctrl+Tab	Cycle to previous document window
Alt+Tab	Cycle to next Windows application
Backspace	Delete character to left of cursor
Ctrl+Backspace	Redo last undo operation
Alt+Backspace	Undo a change to the file
Escape	In Menu Mode: leave menu mode Otherwise: Undo recent typing on a line
Ctrl+Escape	Invoke Windows Task Manager
Alt+Spacebar	Access frame window's system menu
Alt+Shift+Ctrl	Press and hold to interrupt long-running command or macro (works in some cases where Ctrl+Break does not)
Ctrl+A	Adjust text to cursor position
Ctrl+B	Shortcut for Actions Bookmark
Ctrl+C	Center focus line within margins
Ctrl+F	Format text in paragraph within margins
Ctrl+G	Shortcut for Edit Go To
Ctrl+H	Shortcut for Edit Replace
Ctrl+I	Fill block with a specified character
Ctrl+L	Left-adjust text of focus line to left margin column
Ctrl+N	Shortcut for File New
Ctrl+O	Shortcut for File Open
Ctrl+P	Begin a new paragraph
Ctrl+R	Right-adjust text of focus line to right margin column
Ctrl+S	Shortcut for File Save

Key	Action
Ctrl+V	Paste text from clipboard
Ctrl+X	Display parent directory of file in DIR.DIR listing
Ctrl+Y	Redo last undo operation
Ctrl+Z	Undo a change to the file
Alt+A	Add a line
Alt+B	Mark box block
Alt+C	Copy block
Alt+D	Delete focus line
Alt+E	Open the Edit menu
Alt+F	Fill block with a specified character
Alt+G	Delete block
Alt+H	Access Help system
Alt+J	Join two lines
Alt+K	Copy block and leave block marked
Alt+L	Mark line block
Alt+M	Move block
Alt+O	Overlay text with contents of block
Alt+R	Recover a changed or deleted line
Alt+S	Split a line
Alt+U	Unmark block
Alt+W	Delete word
Alt+X	Edit file named at cursor position or in DIR.DIR directory listing
Alt+Z	Mark stream block
Alt+1	Set Bookmark1 at focus line
Alt+2	Set Bookmark2 at focus line
Alt+3	Set Bookmark3 at focus line
Alt+4	Go to Bookmark1
Alt+5	Go to Bookmark2
Alt+6	Go to Bookmark3
Alt+Minus	Access document window’s system menu
Alt+Equal Sign	Duplicate focus line
Shift+Ctrl+A	Adjust text to cursor position

Key	Action
Shift+Ctrl+C	Center focus line within margins
Shift+Ctrl+F	Format text in paragraph within margins
Shift+Ctrl+L	Left-adjust text of focus line to left margin column
Shift+Ctrl+O	Overlay text with contents of block
Shift+Ctrl+P	Begin a new paragraph
Shift+Ctrl+R	Right-adjust text of focus line to right margin column
Shift+Ctrl+W	Delete word
Shift+Ctrl+X	Display parent directory of file in DIR.DIR listing
Application key on Windows-specific keyboards	Display pop-up menu of actions from Edit menu

Function keys

Keyboard assignments for function keys:

Key	Action
F1	Access Help system
F2	Add new line below focus line
F3	QUIT current file
F4	Cursor to next Tab column
F5	Focus line becomes current line
F6	Cycle backward in command line history
F7	Cursor to beginning of line
F8	Duplicate focus line
F9	Reissue last command executed from current file's command line
F10	Cycle to next document window
F11	Split/join line at cursor position
F12	Toggle cursor between command line and file area
Shift+F1	Reissue last LOCATE command
Shift+F2	Cursor to current line
Shift+F3	Find matching brace
Shift+F4	Cycle to next file in the ring
Shift+F5	Uppercase block
Shift+F6	Lowercase block

Key	Action
Shift+F7	Shift block one character to left
Shift+F8	Shift block one character to right
Shift+F9	Scroll half-window to the left
Shift+F10	Scroll half-window to the right
Shift+F11	Enter Menu Mode
Shift+F12	Toggle cursor between command line and file area
Ctrl+F1	Undo last change to file
Ctrl+F2	Redo last undo operation
Ctrl+F4	Close document window
Ctrl+F5	Restore maximized document window to normal size
Ctrl+F6	Cycle to next document window
Ctrl+F10	Maximize document window
Alt+F1	Search forward for Quick Search string
Alt+F2	Search backward for Quick Search string
Alt+F3	Move to Quick Search toolbar item
Alt+F4	Close KEDIT
Alt+F5	Restore maximized frame window to normal size
Alt+F10	Maximize frame window
Shift+Ctrl+F6	Cycle to previous document window

Mouse Actions

Here is a summary of mouse actions used within a document window. In addition, the normal Windows conventions for accessing menus, moving and resizing windows, etc. all apply:

Mouse Operation	Action
Click button 1	Reposition the cursor
Click button 2	Reposition the cursor and unmark any marked block
Click Shift+button 1	Extend a block to mouse pointer position
Drag with button 1	Mark a stream block
Drag with button 2 in file area Drag with button 1 in window margin area or prefix area	Mark a line block
Drag with button 1+button 2	Mark a box block

Mouse Operation	Action
Click Ctrl+button 1 in window margin area or prefix area	Mark entire file as line block
Double-click button 1	Mark a word as a stream block
Double-click button 1 on entry in DIR.DIR file	Edit the file or list the directory

4.5 Summary of Differences Between Classic and CUA Interfaces

4.5.1 Overview

Here is an overview of the differences between KEDIT for Windows' behavior with INTERFACE CUA in effect and with INTERFACE CLASSIC in effect, followed by a more detailed comparison of the differences in keyboard and mouse behavior.

With INTERFACE CUA, you get the Windows-style behavior:

Default behavior for most keys is based on CUA conventions. For example, the Home key moves the cursor to the beginning of a line and Alt+W opens the Window menu.

Windows-style selections, marked by dragging with mouse button 1 or using keystrokes involving Shift+Cursor key, are available. Text that you type after marking a selection replaces the selection, and moving the cursor away from the selection unmarks the selection. In addition, persistent line, box, and stream blocks are available.

You can mark command line selections, useful for editing text on the command line and for moving it to and from the clipboard.

By default, KEDIT uses a vertical text cursor.

With INTERFACE CLASSIC, you get text mode-compatible behavior:

Default behavior for most keys is very close to the behavior in text mode KEDIT. For example, the Home key moves the cursor to the command line, and Alt+W deletes a word.

Persistent line, box, and stream blocks are available, but Windows-style non-persistent selections are not available.

Command line selections are not available.

By default, KEDIT uses a horizontal text cursor, resembling the text mode cursor.

4.5.2 Keyboard Comparison

The following table lists all keys whose behavior differs between INTERFACE CLASSIC and INTERFACE CUA, giving the name of the key, its behavior with INTERFACE CLASSIC, and its default behavior with INTERFACE CUA:

Key	CLASSIC	CUA
Home	Cursor to command line, process pending prefix commands	Cursor to beginning of line
Delete	Delete character	Delete character or selection; Beyond end of line, join with following line
Ctrl+Home	Cursor to upper left of window	Cursor to beginning of file
Ctrl+End	Delete through end of line	Cursor to end of file
Ctrl+Page Up	Cursor to beginning of file	Cursor to upper left of window
Ctrl+Page Down	Cursor to end of file	Cursor to bottom right of window
Shift+Cursor Up Shift+Cursor Down Shift+Cursor Left Shift+Cursor Right Shift+Home Shift+End Shift+Page Up Shift+Page Down Shift+Ctrl+Cursor Left Shift+Ctrl+Cursor Right Shift+Ctrl+Home Shift+Ctrl+End Shift+Ctrl+Page Up Shift+Ctrl+Page Down	unused (Selections not available with INTERFACE CLASSIC)	Extend selection
Numeric Pad Plus	Toggle between file and prefix areas	Cursor to command line
Enter Numeric Pad Enter	In file area: Move to beginning of next line	In file area: Insert new line
Ctrl+Enter	Toggle between file and prefix areas	Cursor to beginning of next line
Tab	Move to next Tab column	Overtype Mode: Move to next tab column Insert Mode: Insert spaces through next Tab column

Key	CLASSIC	CUA
Backspace	Delete character to left of cursor	Delete character to left, or selection; In first column of line, join it with preceding line
Ctrl+A	Adjust text to cursor position	Shortcut for Edit Select All
Ctrl+C	Center line	Copy text to clipboard
Ctrl+F	Format paragraph	Shortcut for Edit Find
Ctrl+P	Begin a new paragraph	Shortcut for File Print
Ctrl+X	Display parent directory in DIR.DIR	Cut text to clipboard
Alt+A	Add a line	Open the Actions menu
Alt+F	Fill block	Open the Fill menu
Alt+H	Access Help system	Open the Help menu
Alt+O	Overlay block	Open the Options menu
Alt+W	Delete word	Open the Window menu
F10	Cycle to next document window	Enter Menu Mode
F12	Toggle between command line and file area	Cursor to command line

Remapped keys

The following table lists the keys whose function changes between INTERFACE CLASSIC and INTERFACE CUA, giving the name of the key involved, the function it performs with INTERFACE CLASSIC, and the name of the key to which that function has been moved under INTERFACE CUA.

CLASSIC Key	Function	Equivalent CUA key
Home	Cursor to command line (also executes pending prefix commands)	F12 or Numeric Pad Plus
Ctrl+Home	Cursor to upper left of window	Ctrl+Page Up
Ctrl+End	Delete through end of line	Ctrl+Delete
Ctrl+Page Up	Cursor to beginning of file	Ctrl+Home
Ctrl+Page Down	Cursor to end of file	Ctrl+End
Numeric Pad Plus or Ctrl+Enter	Toggle between file and prefix areas	Ctrl+Numeric Pad Enter or Shift+Ctrl+Enter
Enter or Numeric Pad Enter	In file area: Move to beginning of next line	Ctrl+Enter
Tab (in Insert Mode)	Move to next Tab column	F4

CLASSIC Key	Function	Equivalent CUA key
Ctrl+A	Adjust text to cursor position	Shift+Ctrl+A
Ctrl+C	Center line	Shift+Ctrl+C
Ctrl+F	Format paragraph	Shift+Ctrl+F
Ctrl+P	Begin a new paragraph	Shift+Ctrl+P
Ctrl+X	Display parent directory in DIR.DIR	Shift+Ctrl+X
Alt+A	Add a line	F2
Alt+F	Fill block	Ctrl+I
Alt+H	Access Help system	F1
Alt+O	Overlay block	Shift+Ctrl+O
Alt+W	Delete word	Shift+Ctrl+W
F10	Cycle to next document window	Ctrl+Tab or Ctrl+F6
F12	Toggle between command line and file area	Shift+F12

4.5.3 Mouse Comparison

Since selections are not available in INTERFACE CLASSIC, all blocks marked with the mouse in INTERFACE CLASSIC are persistent blocks.

In INTERFACE CUA, all blocks marked with the mouse are selections, but you can change this default behavior via the Options Interface dialog box. Also, you can convert a selection to a persistent block by using the Edit Make Persistent menu item, which is also available on the menu displayed when you click mouse button 2.

Activity	CLASSIC	CUA
Mark stream block	Drag with button 1	Drag with button 1
Mark line block	Drag with button 2, or drag with button 1 in window margin or prefix area	Drag with Ctrl+button 1, or drag with button 1 in window margin or prefix area
Mark box block	Drag with button1+button2	Drag with Alt+button 1
Mark word	Double-click button 1	Double-click button 1
Extend block	Click Shift+button 1	Click Shift+button 1
Unmark selection	Selections unavailable	Click button 1
Unmark persistent block	Click button 2	Click Alt+button 1 or click button 2 and choose Unmark from menu

Activity	CLASSIC	CUA
Drag-and-drop move	Unavailable	Position mouse pointer in selection, drag with button 1
Drag-and-drop copy	Unavailable	Position mouse pointer in selection, drag with Ctrl+button 1
Display pop-up menu	Application key on Windows-specific keyboards	Click button 2

4.6 Entering Special Characters

Alt key+numeric pad

Windows provides a method for entering special characters not found on your keyboard, similar to the ''Alt key+Numeric Pad'' method that is provided by DOS. This Alt key-numeric pad method of entering text is built into Windows and is not specific to KEDIT. For example, it can be used with the Write program included as part of Windows. The details depend on whether you are working with files in the ANSI or OEM character set, so these two cases are considered separately.

ANSI character set

If you are using an ANSI font to work with text in the ANSI character set, you can enter a special character by holding down the Alt key and entering the decimal value of the OEM or ANSI code for the desired character via the digits on the numeric keypad. The code should be in the range 1 to 255; you can't use this method to enter null characters with character code 0. If you enter the code with no leading zeros, Windows assumes that you have entered the OEM code for the character involved. Windows translates this to the equivalent ANSI code and then passes the character to KEDIT. To enter an ANSI code directly, the code that you enter should be preceded by a leading zero.

To enter, for example, a formfeed character (character code 12 in both OEM and ANSI) into your file, you can hold down the Alt key and press 1 and 2 on the numeric pad. You would get the same result by holding down the Alt key and pressing 0, 1, and 2. Both methods would pass character code 12 to KEDIT.

For an example of a character whose ANSI and OEM codes differ, consider the British pound symbol, which is not found on the standard U.S. keyboard. With the code pages used by most KEDIT users, its code in the OEM character set is 156 and its code in the ANSI character set is 163. You could therefore enter the British pound symbol by holding down Alt and pressing 1, 5, and 6 on the numeric keypad or by holding down Alt and pressing 0, 1, 6, and 3. Both methods would pass character code 163 to KEDIT.

OEM character set

If you are using an OEM font to work with text in the OEM character set, you can enter a special character by holding down the Alt key and entering a 0 and then the decimal value of the OEM code for the desired character via the digits on the numeric keypad. The code should be in the range 1 to 255; you can't use this method to enter null characters with character code 0. You must precede the character code involved with a leading 0, or else Windows will assume that you want the character translated from OEM to ANSI, and you will end up with an incorrect result.

To enter, for example, a formfeed character (character code 12) into your file, you can hold down the Alt key and press 0, 1, and 2 on the numeric pad. To enter a British pound symbol (OEM character code 156), you can hold down the Alt key and press 0, 1, 5, and 6.

Character code 0

You cannot use the Alt key+Numeric Pad method to enter a null character into your file. (The null character is the character whose code is 0. Note that this is not the same as the character ‘‘0’’, whose character code is 48.) The easiest way to enter a null character is to define a macro that enters a null character and assign this macro to a key. For example, adding the following line to your profile will assign to Ctrl+F3 a macro that enters the null character:

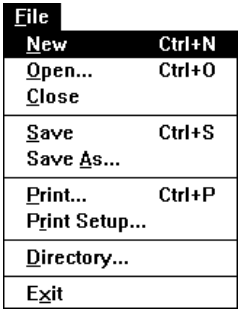
```
"define ctrl+f3 'text' d2c(0)"
```

Chapter 5. Menus and Toolbars

This chapter has information on each of KEDIT's menu items and toolbar buttons.

5.1 File Menu

Use the File menu to open, close, save, and print files that you are editing.



5.1.1 New

Edits a new file. The new file will have as a temporary name the first available name of the form UNTITLED.1, UNTITLED.2, etc. If you make changes to the file and then save these changes to disk, KEDIT will prompt you for a permanent name for the file.

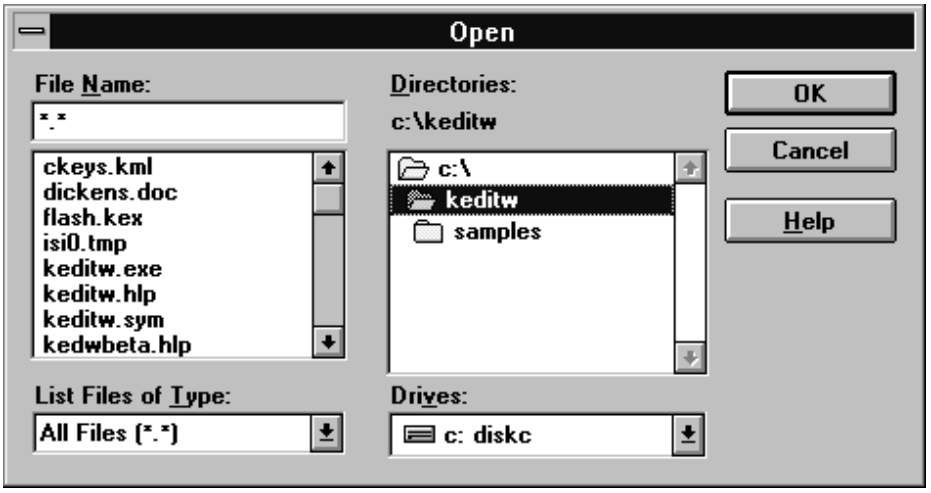
Shortcuts

Click on the New File button  on the top toolbar.

Press Ctrl+N.

See also Section 3.5, "Editing Multiple Files"

5.1.2 Open...



Begins editing an existing file or group of files.

Shortcuts

Click on the Open File button  on the top toolbar.

Press Ctrl+O.

Dialog box options

The appearance and behavior of this dialog box vary depending on the version of Windows that you are using.

File Name

Type the name of the file that you want to edit, or select a file from the list box. The box displays a list of files based on the types of files that you select in the List Files of Type box.

You can use mouse button 2 or Ctrl+button 1 to select multiple files.

List Files of Type

Select the type of files that you want listed in the File Name list box.

All Files (*.*)	All files in the selected directory are displayed.
Text Files (*.TXT)	Files in the selected directory with a file extension of .TXT are displayed.
Development (*.C, *.CPP, *.H)	Files in the selected directory with file extension .C, .CPP, or .H are displayed.

KEDIT Macros (*.KEX, *.KML)	Files in the selected directory with a file extension of .KEX (a KEDIT macro) or .KML (a KEDIT macro library) are displayed.

You can use the SET OPENFILTER command to customize the list of file types that you can choose from.

Directories

Select the directory containing the file that you want to open.

Drives

Select the drive containing the file that you want to open.

Open

Opens the file you have selected in File Name.

Update List

Updates the list of filenames shown in the File Name list based on any wildcard fileid specifications you have entered into the File Name field.

See also SET FILEOPEN (Options SET Command), KEDIT (Command Line), DIR (Command Line), Directory (File Menu), Section 3.5, "Editing Multiple Files"

5.1.3 Close

Closes the current file. If you have made changes to the file but have not saved them to disk, KEDIT will ask if you want to save the changes.

Shortcuts If the file has not been altered, you can press F3.

See also SAVE (Command Line), FILE (Command Line), QUIT (Command Line)

5.1.4 Save

Saves the current file to disk using its current fileid. If the current file is an UNTITLED file, the Save As dialog box will be displayed so you can specify a permanent fileid.

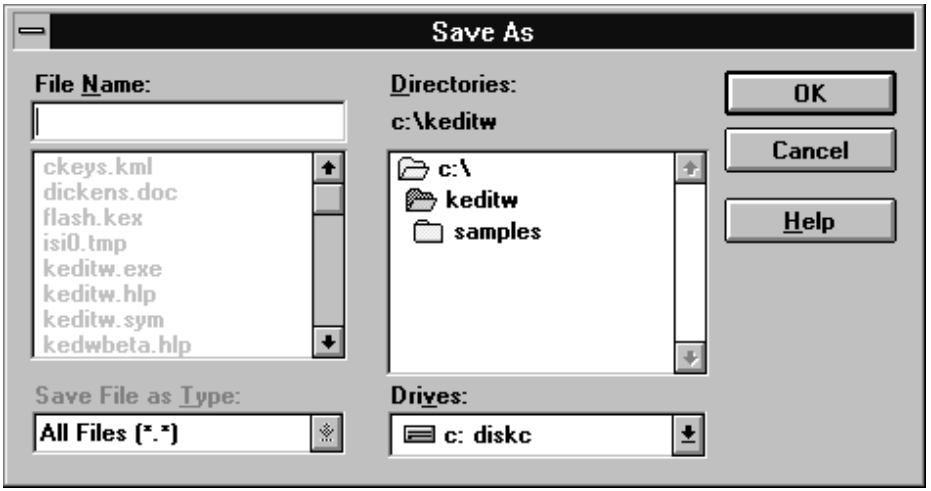
Shortcuts

Click on the Save File button  on the top toolbar.

Press Ctrl+S.

See also SAVE (Command Line), FILE (Command Line)

5.1.5 Save As...



Saves and gives a new filename to the current file.

Dialog box options

The appearance and behavior of this dialog box vary depending on the version of Windows that you are using.

File Name

Type the new filename for the current file.

Save File as Type

This item is not used by the Save As dialog box.

Directories

Select the directory in which you want to save your file.

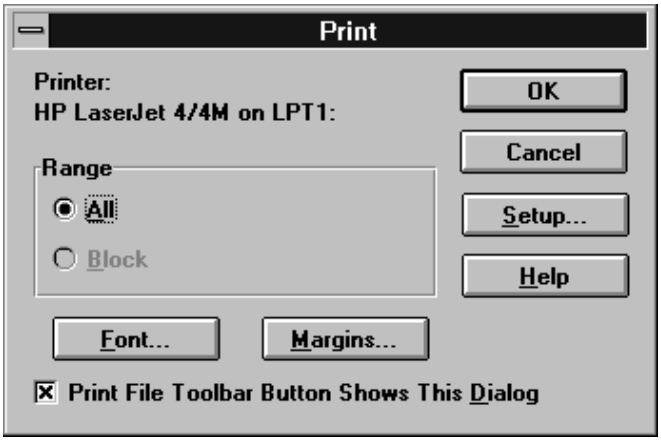
Drives

Select the drive to use for saving your file.

See also

SAVE (Command Line), FILE (Command Line)

5.1.6 Print...



Prints your entire file, or the currently-marked portion of your file.

Shortcuts

Click on the Print File button  on the top toolbar.

Press Ctrl+P.

Dialog box options

All

The entire file will be printed when you press the Print button.

Block

Enabled only if there is a marked block within the file; the marked area of the file will be printed when you press the Print button.

Font...

Displays a dialog box that lets you select KEDIT's printer font from a list of fixed-pitch printer fonts installed on your system. The Font dialog box is only available when the SET PRINTER option specifies that your printer output will be handled by Windows.

Components of the Font dialog box:

Font	Select your desired printer font from a list of fixed-pitch printer fonts installed on your system.
Font Style	Select your desired font style from a list of font styles available for the current font.
Size	Type in or select your desired printer font size.

Margins...

Displays a dialog box that lets you select the size of the margin area that KEDIT will use when sending your file to the printer. The Margins dialog box is only available when the SET PRINTER option specifies that your printer output will be handled by Windows.

Components of the Margins dialog box:

Left	Specify the size of the left margin.
Right	Specify the size of the right margin.
Top	Specify the size of the top margin.
Bottom	Specify the size of the bottom margin.
Inches	This radio button indicates that your margin areas are specified in inches.
Centimeters	This radio button indicates that your margin areas are specified in centimeters.
KEDIT Default	Resets your printer margins to the default values, which are .25 inches for the left and right margins and .50 inches for the top and bottom margins.

Setup

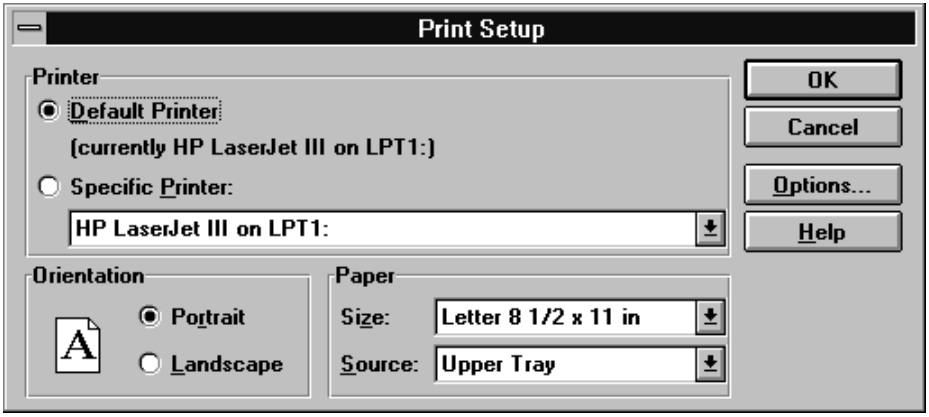
Opens the Print Setup dialog box. Print Setup is only available when the SET PRINTER option specifies that your printer output will be handled by Windows.

Print File Toolbar Button Shows This Dialog

When this box is checked, as it is by default, clicking on the Print File button  on the top toolbar causes the Print dialog box to be displayed. When this box is not checked, the Print File toolbar button does not display the Print dialog box. Instead, it immediately prints the marked block, if there is one, or else prints the entire file.

See also PRINT (Command Line), SET PRINTER (Options SET Command), Section 3.10, "Printing"

5.1.7 Print Setup...



Lets you select the printer and printer options that you want to use when printing from within KEDIT. This dialog box is only available when the SET PRINTER option specifies that your printer output will be handled by Windows.

Dialog box options

The appearance and behavior of this dialog box vary depending on the version of Windows that you are using.

Printer

Default Printer — Use this radio button to route printer output to your default Windows printer.

Specific Printer — Use this radio button to route printer output to a printer that you select from a list of available Windows printers.

Orientation

Portrait — Text is printed from left to right across the narrower dimension of the page so that the output page is, like a portrait painting, taller than it is wide.

Landscape — Text is printed from left to right across the wider dimension of the page so that the output page is, like a landscape painting, wider than it is tall.

Paper

Size — Indicates the paper size to be used on a printer that supports multiple paper sizes.

Source — Indicates the paper source to be used on a printer that supports multiple paper sources.

Options...

Displays a dialog box that lets you specify additional printer options. The Option dialog box is different for each printer driver and is provided by the printer manufacturer.

See also SET PRINTER (Options SET Command)

5.1.8 Directory...

This dialog box lets you select a directory, and then make that directory the current directory and/or create a DIR.DIR file listing the contents of that directory.

Dialog box options

Directory

Select the directory that is of interest to you. That is, select the directory that should become the current directory or whose contents you want to display in a DIR.DIR file.

Drives

Select the drive containing the directory that is of interest to you.

Set Current Directory

When this radio button is selected, pressing the OK button makes the drive specified in the Directory field become the current directory.

Show DIR.DIR file

When this radio button is selected, pressing the OK button creates a DIR.DIR file listing the contents of the drive specified in the Directory field.

Set Current Directory And Show DIR.DIR

When this radio button is selected, pressing the OK button makes the drive specified in the Directory field become the current directory and also creates a DIR.DIR file listing the contents of that directory.

Notes

The current directory is used within KEDIT for several purposes. For example, when you use File New to begin editing an untitled file, the current directory is used as the path specification for that file. When you use the DOS command to shell to an MS-DOS command session, that session inherits KEDIT's current directory. And when you issue the KEDIT command and do not give a path specification for the file you want to edit, KEDIT begins its search in the current directory. Note that changing the current directory does not affect the default directory for the File Open dialog box, which KEDIT keeps separate track of.

See also

CHDIR (Command Line), DIR (Command Line)

5.1.9 Exit

Exits from KEDIT. If you are editing any files that have unsaved changes, KEDIT will ask if you want to save the changes, and then KEDIT will remove all files from the ring and terminate the editing session.

Shortcuts

Press Alt+F4.

Double-click on the frame window's system menu icon.

If there are no open documents, press F3.

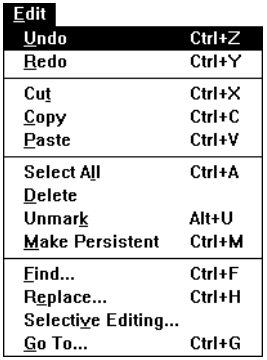
5.1.10 Recently Edited File List

At the bottom of the File menu is a list of recently-edited files. If you select one of the files in the list, KEDIT will begin editing that file. By default the list holds up to 9 recently-edited files, but you can increase this number to as high as 16 by using the RECENTFILES option of Options SET Command.

See also

SET RECENTFILES (Options SET Command), KEDIT (Command Line)

5.2 Edit Menu



Use this menu for assorted editing operations, such as find and replace, undo and redo, and clipboard cut, copy, and paste.

5.2.1 Undo

Undo one level of changes to the current file. This menu item can be used repeatedly to undo multiple levels of changes.

Shortcuts

Click on the Undo button  on the top toolbar.

Press Ctrl+Z.

Press Alt+Backspace.

See also

Redo (Edit Menu), UNDO (Command Line), SET UNDOING (Options SET Command), Section 3.13, "The Undo Facility"

5.2.2 Redo

Redo a change to your file, reversing the effect of Edit Undo. If you have used Edit Undo repeatedly to undo multiple changes, then you can use Edit Redo repeatedly to redo those changes. Redo is only available if you have recently used Edit Undo and have not made any intervening changes to your file.

Shortcuts

Click on the Redo button  on the top toolbar.

Press Ctrl+Y.

Press Ctrl+Backspace.

See also UNDO (Edit Menu), REDO (Command Line), SET UNDOING (Options SET Command), Section 3.13, "The Undo Facility"

5.2.3 Cut

Cuts text to the clipboard. The contents of the current block or selection are placed in the clipboard, replacing any existing clipboard contents, and the block or selection is deleted from your file. This menu item is unavailable if no text is currently selected.

Shortcuts

Click on the Cut to Clipboard button  on the top toolbar.

Press Ctrl+X. (CUA interface only)

Press Shift+Delete.

Click mouse button 2 and then select Cut from the resulting pop-up menu. (CUA interface only)

5.2.4 Copy

Copies the contents of the current block or selection to the clipboard, replacing any existing clipboard contents. This menu item is unavailable if no text is currently selected.

Shortcuts

Click on the Copy to Clipboard button  on the top toolbar.

Press Ctrl+C. (CUA interface only)

Press Ctrl+Insert.

Click mouse button 2 and then select Copy from the resulting pop-up menu. (CUA interface only)

5.2.5 Paste

Copies the text that is in the clipboard into your file or to the command line, inserting it at the cursor position. Multi-line clipboard text can only be pasted into your file, and not to the command line.

Shortcuts

Click on the Paste from Clipboard button  on the top toolbar.

Press Ctrl+V.

Press Shift+Insert.

Click mouse button 2 and then select Paste from the resulting pop-up menu. (CUA interface only)

5.2.6 Select All

Marks the entire file. In INTERFACE CUA, a line selection is marked. In INTERFACE CLASSIC, a persistent line block is marked.

Shortcuts

Press Ctrl+A. (CUA interface only)

Press Ctrl+5 on the numeric pad.

Click with Ctrl+button 1 in the margin area or in the prefix area.

5.2.7 Delete

Deletes the currently-marked selection or persistent block without copying its contents to the clipboard.

Shortcuts

Press the Delete or Backspace keys immediately after marking a selection or a persistent block. (CUA interface only)

Press Alt+G.

Click mouse button 2 and then select Delete from the resulting pop-up menu. (CUA interface only)

5.2.8 Unmark

Unmarks the currently-marked selection or persistent block. This menu item is primarily useful for unmarking persistent blocks, since selections can be unmarked by simply repositioning the cursor.

Shortcuts

Press Alt+U.

Click mouse button 2 and then select Unmark from the resulting pop-up menu. (CUA interface only)

5.2.9 Make Persistent

Converts the currently-marked selection into a persistent block. This is only available when INTERFACE CUA is in effect, since selections are not used with INTERFACE CLASSIC.

Shortcuts

Press Ctrl+M.

Click mouse button 2 and then select Make Persistent from the resulting pop-up menu.

5.2.10 Find...

Finds a particular word or character string in your file.

Shortcuts

Press Ctrl+F. (CUA interface only)

Click on the Find Dialog Box button  on the top toolbar.

To repeat the last search, click on the Find Next button  on the top toolbar. To search backward, hold down the Shift key when you click on the Find Next button.

Enter your search string in the Quick Find field on the top toolbar  or select a recent search string from the list displayed by clicking the down arrow beside the edit field. Then, press Enter.

Dialog box options

Find What

Specify the text to be searched for by doing one of the following:

Type the text to be searched for into the edit field.

Select an item from the drop down list box with the text of your most recent search strings. Click on the down arrow beside the entry field to view this list.

Paste a search string from the clipboard. (Ctrl+V)

Match Case

When this box is checked, the search is case sensitive. For example, ‘‘ABC’’ is not treated as matching ‘‘abc’’. When this box is unchecked, the search is case insensitive. ‘‘ABC’’ and ‘‘abc’’ are then treated as matching strings.

Find Whole Words Only

Check this box when you want to find occurrences of your search string that are complete words. For example, when this box is checked, a search for ‘‘birth’’ would match occurrences of the word ‘‘birth’’ within your file, but would not match in ‘‘rebirth’’ or ‘‘birthday’’.

Regular Expressions

Check this box to allow regular expression notation in the search string. For information on regular expressions, see Section 6.6, “Regular Expressions”, and Section 6.6.5, “Regular Expression Summary”.

Forward

Use the Forward button to search forward in your file for the next occurrence of the specified search string. The search ends at the bottom of the file unless SET WRAP ON is in effect, in which case the search continues on from the top of the file so that the entire file is searched.

Backward

Use the Backward button to search backward in your file for an occurrence of the specified search string. The search ends at the top of the file unless SET WRAP ON is in effect, in which case the search continues on from the bottom of the file so that the entire file is searched.

Close

When you have finished searching, use the Close button or press the Escape key to close the dialog box.

Notes

A number of KEDIT’s SET options affect the search done by the Find dialog box. These options can be controlled by using the Options SET Command dialog box or by issuing SET commands from the command line. Among the options involved are:

STAY option — controls whether the position of the focus line remains the same after an unsuccessful search or whether the top-of-file or end-of-file line becomes the focus line.

WRAP option	controls whether the search ends when the top or bottom of the file is reached, or instead wraps around so that the entire file is searched.
VARBLANK option	controls whether a blank in the search string will match only a single blank in your file, or any sequence of one or more blanks.
ZONE option	limits searches to a certain column range. For example, you can specify that searches should examine only columns 10 through 20 of each line in the file.

Command line equivalent

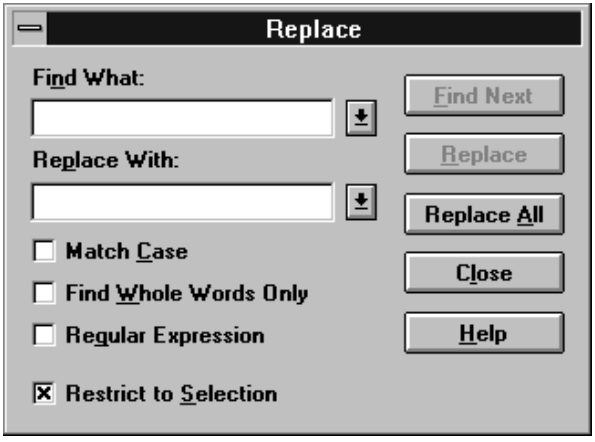
Using this dialog box is roughly equivalent to using the command

```
LOCATE /string/
```

See also

LOCATE (Command Line), SET ZONE (Options SET Command), SET WRAP (Options SET Command), SET STAY (Options SET Command), SET VARBLANK (Dialog Box)

5.2.11 Replace...



Finds and replaces specified character strings in your file.

Shortcuts

Press Ctrl+H.

Dialog box options

Find What

Specify the text to be searched for by doing one of the following:

Type the text to be searched for into the edit field.

Select an item from the drop down list box with the text of your most recent search strings. Click on the down arrow beside the entry field to view this list.

Paste the text from the clipboard. (Ctrl+V)

Replace With

Specify the new text that will replace the existing text by doing one of the following:

Type the replacement text into the box.

Select an item from the drop down list box with the text of your most recent replacement strings. Click on the down arrow beside the entry field to view this list.

Paste the text from the clipboard. (Ctrl+V)

Match Case

When this box is checked, the search is case sensitive. For example, ‘‘ABC’’ is not treated as matching ‘‘abc’’. When this box is unchecked, the search is case insensitive. ‘‘ABC’’ and ‘‘abc’’ are then treated as matching strings.

Find Whole Words Only

Check this box when you want to find occurrences of your search string that are complete words. For example, when this box is checked, a search for ‘‘birth’’ would match occurrences of the word ‘‘birth’’ within your file, but would not match in ‘‘rebirth’’ or ‘‘birthday’’.

Regular Expressions

Check this box to allow regular expression notation in the search string. For information on regular expressions, see Section 6.6, “Regular Expressions”, and Section 6.6.5, “Regular Expression Summary”.

Restrict to Selection

When selected, this option causes the search string to be replaced only in the currently-marked selection or persistent block. Every occurrence in the selection or block is replaced, so the Replace All button is available but the Replace button is grayed out. This option is selected by default if the cursor is in a selection or block. This option is grayed out if there is no current selection or block.

Find Next

The Find Next button searches forward through your file for the next occurrence of the search string. The search ends at the bottom of the file unless SET WRAP ON is in effect, in which case the search continues on from the top of the file so that the entire file is searched.

Replace

The Replace button replaces the highlighted occurrence of the search string with the new text in the Replace With box.

Undo

When the Replace button is chosen and the old text is replaced with new text, the Replace button changes to an Undo button. Use the Undo button to undo the effect of the most recent Replace operation.

Replace All button

The Replace All button immediately changes the old text specified in the Find What box to the new text specified in the Replace With box. All occurrences are changed either in the entire file or in the currently-marked block, depending on whether Restrict to Selection is checked.

Close

When you have finished replacing text, use the Close button or press the Escape key to close the Replace dialog box.

Notes

A number of KEDIT's SET options affect the search done by the Replace dialog box. These options can be controlled by using the Options SET Command dialog box or by issuing SET commands from the command line. Among the options involved are:

STAY option	controls whether the position of the focus line remains the same after an unsuccessful search or whether the end-of-file line becomes the focus line.
WRAP option	controls whether the search ends when the top or bottom of the file is reached, or instead wraps around so that the entire file is searched.
ZONE option	limits searches to a certain column range. For example, you can specify that searches should examine only columns 10 through 20 of each line in the file.

Command line equivalent

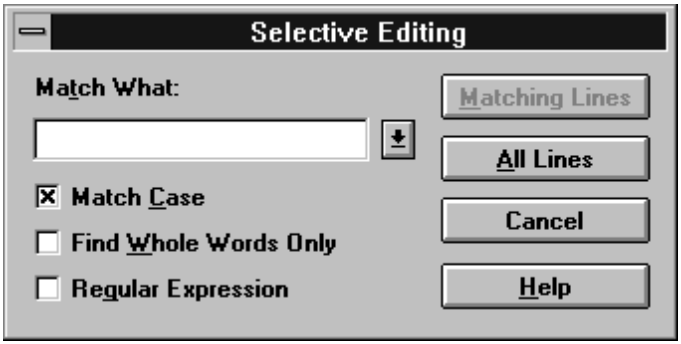
Using the Replace dialog box is roughly equivalent to using the command

```
CHANGE /string1/string2/ target n
```

See also

CHANGE (Command Line), SET ZONE (Options SET Command), SET WRAP (Options SET Command), SET STAY (Options SET Command)

5.2.12 Selective Editing...



Selects a subset of your file consisting of all lines matching a specified string. All other lines of your file are temporarily excluded from display and are unaffected by most KEDIT commands.

Dialog box options

Match What

Enter the character string that is of interest to you by doing one of the following:

Type the text into the edit field.

Select an item from the drop down list box of recently-matched strings. Click on the down arrow beside the entry field to view this list.

Match Case

When this box is checked, the search is case sensitive. For example, ‘‘ABC’’ is not treated as matching ‘‘abc’’. When this box is unchecked, the search is case insensitive. ‘‘ABC’’ and ‘‘abc’’ are then treated as matching strings.

Find Whole Words Only

Check this box when you want to find occurrences of your search string that are complete words. For example, when this box is checked, a search for ‘‘birth’’ would match occurrences of the word ‘‘birth’’ within your file, but would not match in ‘‘rebirth’’ or ‘‘birthday’’.

Regular Expressions

Check this box to allow regular expression notation in the search string. For information on regular expressions, see Section 6.6, “Regular Expressions”, and Section 6.6.5, “Regular Expression Summary”.

Matching Lines

Use this button to display only the lines that match the character string in the Match What box.

Lines that do not contain the matched string are not displayed and are represented by shadow lines if the default setting SET SHADOW ON is in effect.

All Lines

Use this button to leave selective editing mode and to begin displaying all lines in the file.

Notes

You can limit the string search to certain columns using the ZONE option (Options SET Command). For example, you can specify that the search only examine columns 10 through 20 of each line in the file.

Command line equivalent

Using the Matching Lines button of this dialog box to select the lines containing a particular string is roughly equivalent to using the command

```
ALL /string/
```

Using the All Lines button to leave selective editing mode is equivalent to using the command

```
ALL
```

Tips

To toggle between seeing only the selected lines and seeing all of the lines in the file:

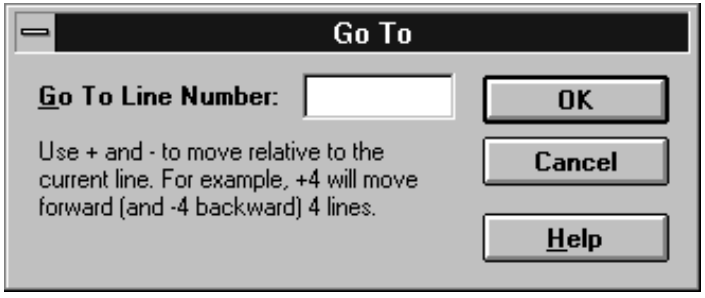
Click on the Hide Excluded Lines button  on the optional bottom toolbar to edit only the subset of lines you have already selected using the Selective Editing Dialog Box (or the ALL command).

Click on the Show All Lines button  on the optional bottom toolbar to edit all of the lines in the file.

See also

ALL (Command Line), Chapter 8, “Selective Line Editing and Highlighting”

5.2.13 Go To...



Allows you to move elsewhere in your file by specifying either a specific line number or an offset from your current location.

Shortcuts Press Ctrl+G.

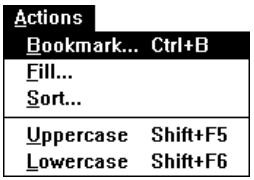
Dialog box options

Go To Line Number

Specify the line number of the new focus line. To move to a specific line number, simply enter that line number. To give an offset from your current location, precede the number with a + or -. For example, 4 goes to line 4 of your file, +4 moves down four lines in your file, and -4 moves up four lines in your file.

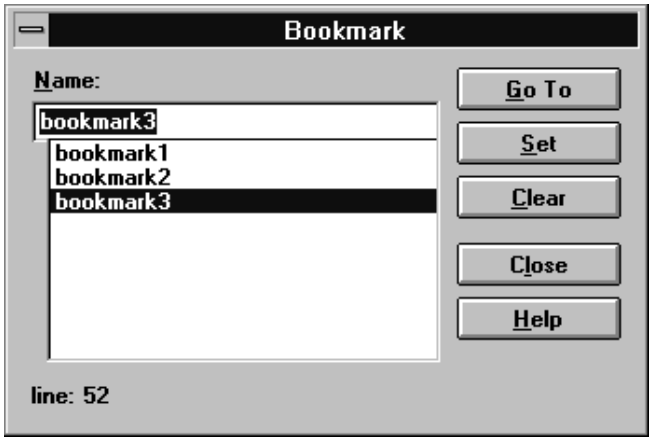
See also Bookmark (Actions Menu), LOCATE (Command Line)

5.3 Actions Menu



Use this menu to mark lines with bookmarks, sort lines, uppercase or lowercase text, or fill a block with a specified character.

5.3.1 Bookmark...



Associates a name, referred to as a bookmark, with the focus line or allows you to return to a line that you have previously named.

Shortcuts Press Ctrl+B.

Click on the Set Bookmark1 button  on the optional bottom toolbar to give the name Bookmark1 to the focus line.

Click on the Go to Bookmark1 button  on the optional bottom toolbar to return to the line that you have previously given the name Bookmark1.

Dialog box options

Name

Specify a bookmark name by typing in a name or selecting it from the list of existing names.

Go To

KEDIT moves to the line corresponding to the bookmark that you have selected.

You can also move to a specific bookmark by double-clicking on an item in the list of currently-defined bookmarks.

Set

A bookmark will be set at the focus line, with the name specified in the Name field.

Clear

Removes a bookmark. If a line has previously been given the name specified in the Name box, it is no longer associated with that name.

Command line equivalent

Using the Set button to define a bookmark is equivalent to using the command

```
SET POINT .name
```

Using the Go To button to move to a bookmark is equivalent to using the command

```
LOCATE .name
```

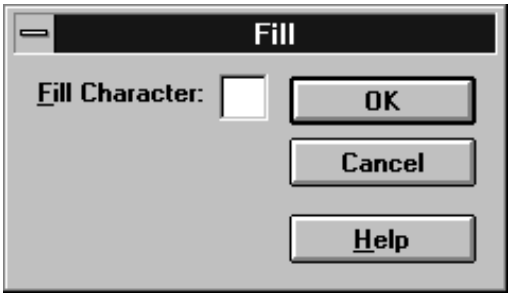
Using the Clear button to remove a bookmark is equivalent to using the command

```
SET POINT .name OFF
```

See also

SET POINT (Command Line)

5.3.2 Fill...



Fills a block with multiple copies of a specified character. This menu item is grayed out if no block is marked.

Fill Character

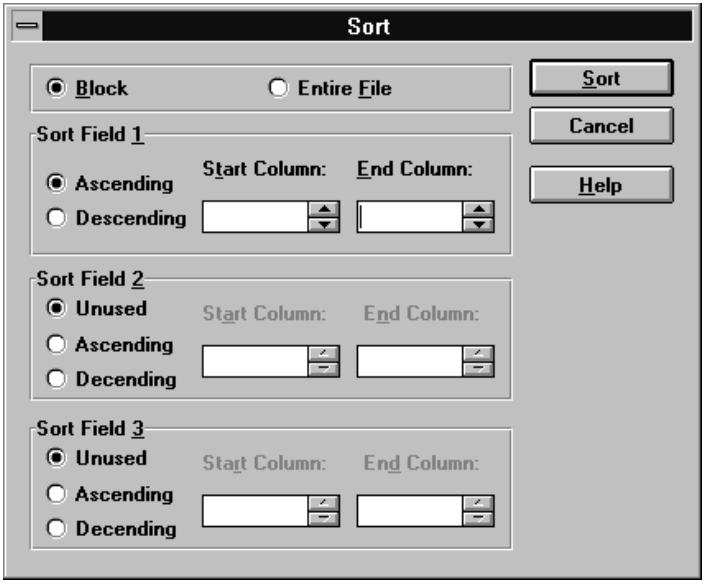
Specify the character to be used to fill the block.

Shortcuts

Click on the Fill Block button  on the optional bottom toolbar.

See also FILL (Command Line)

5.3.3 Sort...



Sorts all or part of your file.

Dialog box options

Block

Choose this radio button to sort all lines within the currently-marked line or box block. This button is grayed out if there is no line or box block.

Entire File

Choose this radio button to sort the entire file.

Sort Fields

KEDIT decides what order to put lines in by comparing characters in the columns specified by your sort fields. Each field is expressed as a pair of numbers giving the leftmost and rightmost columns of the field. There can be up to 3 sort fields.

By default, KEDIT sorts on all columns of your file unless a box block is marked, in which case KEDIT sorts on the columns contained within the box block.

Unused	Indicates that the particular sort field is not being used in this sort.
Start Column	Specify the leftmost column of the sort field.
End Column	Specify the rightmost column of the sort field.
Ascending	The field will be sorted in ascending order.
Descending	The field will be sorted in descending order.

Notes

Sorts are affected by the SET CASE and SET INTERNATIONAL options. If the second operand of SET CASE specifies case insensitive text searches, text comparisons done during the sort will also be case insensitive. SET INTERNATIONAL determines whether KEDIT uses your Windows language drivers to sort text according to country-specific conventions.

See also

SORT (Command Line), SET CASE (Options SET Command), SET INTERNATIONAL (Options SET Command)

5.3.4 Uppercase

Uppercases text within the currently-marked block. This menu item command is grayed out if no block is marked.

Shortcuts

Press Shift+F5.

Click on the Uppercase Block button  on the optional bottom toolbar.

See also UPPERCASE (Command Line)

5.3.5 Lowercase

Lowercases text within the currently-marked block. This menu item command is grayed out if no block is marked.

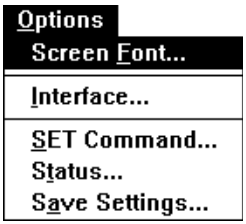
Shortcuts

Press Shift+F6.

Click on the Lowercase Block button  on the optional bottom toolbar.

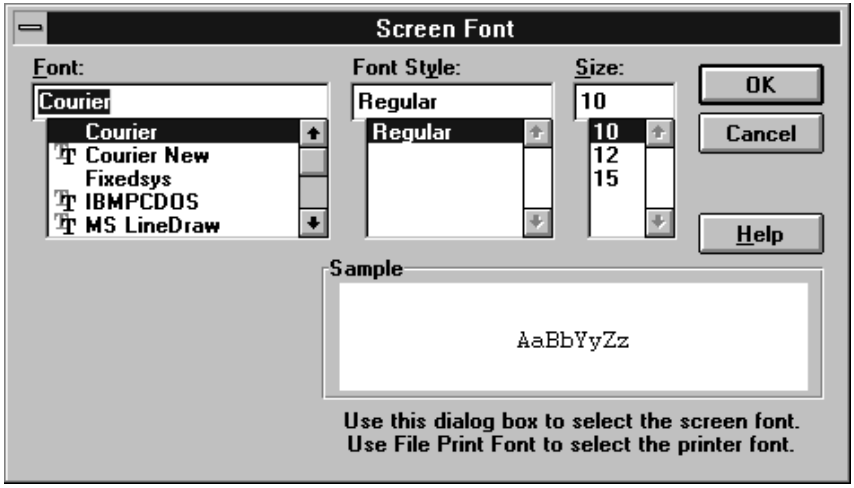
See also LOWERCASE (Command Line)

5.4 Options Menu



Use this menu to select your screen font, and to work with KEDIT's SET options.

5.4.1 Screen Font...



Use this dialog box to control the font that KEDIT uses within your document windows. Note that this dialog box does not affect the font used for printing files; the printer font is controlled through the File Print dialog box.

Dialog box options

The appearance and behavior of this dialog box vary depending on the version of Windows that you are using.

Font	Select your desired font from a list of fixed-pitch screen fonts installed on your system.
Font Style	Select your desired font style from a list of font styles available for the currently selected font.
Size	Type in or select your desired font size.

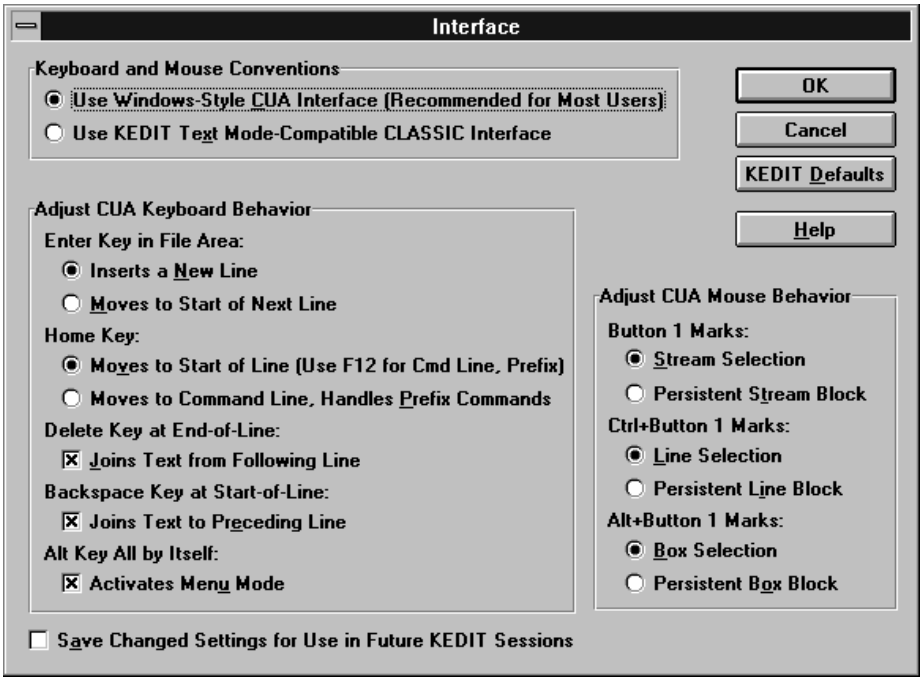
Notes

Note that KEDIT uses only fixed-pitch fonts within its document windows. If a font that is properly installed on your system is not listed in the Screen Font dialog box, either the font is not a fixed-pitch font, or the supplier of the font did not mark the font as a fixed-pitch font. Another possibility is that you have used the Fonts item in the Windows Control Panel to specify that Windows should use only TrueType fonts; in this case, fixed-pitch fonts that are not TrueType fonts are not listed.

See also

Section 3.6, “Fonts”

5.4.2 Interface...



Controls whether certain aspects of KEDIT for Windows' user interface, primarily involving keyboard and mouse usage, work according to the CUA ("Common User Access") conventions used by most other Windows applications, or whether they instead work according to the conventions used in the text mode version of KEDIT.

Dialog box options

Keyboard and Mouse Conventions

Use Windows-Style CUA Interface (Recommended for Most Users)

When selected, you get the Windows-style keyboard and mouse behavior. The behavior of most keys is based on CUA conventions. Blocks marked with the mouse or with CUA-compatible keys behave like selections in that any text that you type after marking a block replaces the contents of the block and moving the cursor away from the block unmarks the block. Some aspects of this behavior can be adjusted via the other items in this dialog box.

Use KEDIT Text Mode-Compatible CLASSIC Interface

When selected, you get text mode-compatible keyboard and mouse behavior. All blocks in INTERFACE CLASSIC are persistent blocks: typing a character does not replace the block's contents, and cursor movement does not unmark the block.

Adjust CUA Keyboard Behavior

Only available when KEDIT's Windows-style CUA interface has been selected.

Allows you to adjust some of KEDIT's keys to behave more like they did in text mode KEDIT. These are the keys that users of text mode KEDIT sometimes find hard to get used to when switching to the CUA interface.

Enter Key in File Area

Controls the behavior of the Enter key when the cursor is in the file area.

Inserts a New Line

Enter inserts a new line following the character at the cursor position. This is the standard CUA behavior.

Moves to Start of Next Line

Enter moves the cursor to the left margin column of the next line of the file area.

Home Key

Controls the behavior of the Home key.

Moves to Start of Line (Use F12 for Cmd Line, Prefix)

The Home key moves the cursor to the start of the line it is currently on. This is the standard CUA behavior. (The F12 key will move the cursor to the command line and if you have the prefix area turned on, F12 will also execute any pending prefix area commands.)

Moves to Command Line, Handles Prefix Commands

The Home key moves the cursor to the command line and executes any pending prefix commands.

Delete Key at End-of-Line

Controls the behavior of the Delete key when the cursor is at the end of a line.

Joins Text from Following Line

If this box is checked, pressing Delete with the cursor at the end of a line will join the text from the next line to the cursor position. This is the standard CUA behavior.

If this box is unchecked, the Delete key will do nothing if pressed when the cursor is at the end of a line.

Backspace Key at Start-of-Line

Controls the behavior of the Backspace key when the cursor is at the start of a line.

Joins Text to Preceding Line

If this box is checked, pressing Backspace with the cursor at the start of a line will join the text from the line the cursor is on to the end of the preceding line. This is the standard CUA behavior.

If this box is unchecked, the Backspace key will do nothing if pressed when the cursor is at the start of a line.

Alt Key All by Itself

Controls the behavior of the Alt key when it is pressed and released alone and not in combination with any other key.

Activates Menu Mode

When this box is checked, pressing and releasing the Alt key activates the menu bar at the top of the frame window. This is the standard CUA behavior.

When this box is unchecked, pressing and releasing the Alt key has no effect.

Adjust CUA Mouse Behavior

Only available when KEDIT's Windows-style CUA interface has been selected.

Specifies whether, when you use the mouse to mark text, you mark selections or persistent blocks.

In addition to Windows-style selections, KEDIT also supports persistent line, box, and stream blocks. Unlike selections, persistent blocks remain marked, regardless of any typing or cursor movement, until you specifically unmark them.

Button 1 Marks

Stream Selection

If selected, dragging in the file area with button 1 marks a stream selection. This is the standard CUA behavior.

Persistent Stream Block

If selected, dragging in the file area with button 1 marks a persistent stream block.

Ctrl+Button 1 Marks

Line Selection

If selected, dragging in the file area with Ctrl+button 1, or dragging with button 1 in the window margin or in the prefix area, marks a line selection. This is the normal CUA behavior.

Persistent Line Block

If selected, dragging in the file area with Ctrl+button 1, or dragging with button 1 in the window margin or in the prefix area, marks a persistent line block.

Alt+Button 1 Marks

Box Selection

If selected, dragging with Alt+button 1 in the file area marks a box selection. This is the normal CUA behavior.

Persistent Box Block

If selected, dragging with Alt+button 1 in the file area marks a persistent box block.

Save Changed Settings for Use in Future KEDIT Sessions

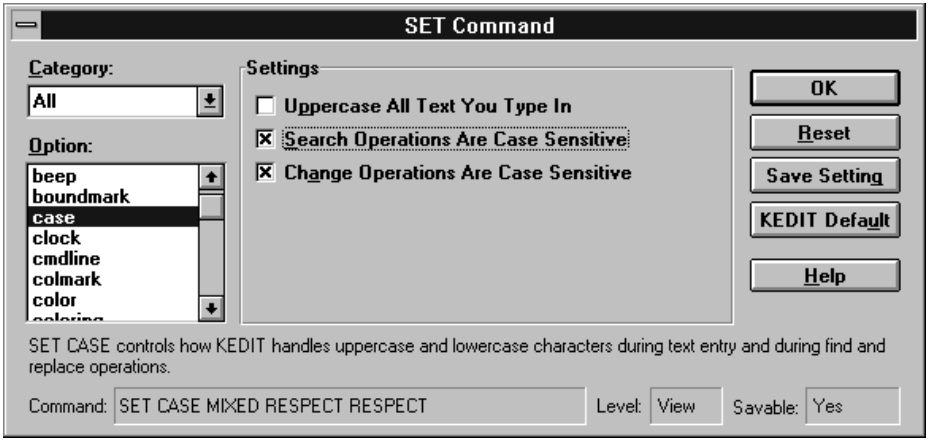
If this box is checked, the changes you have made to the Interface dialog box will be saved to the Windows registry and will apply to future KEDIT sessions. If this box is unchecked, your interface changes will only affect the current editing session.

KEDIT Defaults

Changes all of the options in the dialog box back to KEDIT's built-in defaults.

See also Section 3.2, "CUA and Classic Interfaces", Section 3.3, "Blocks and Selections"

5.4.3 SET Command...



Displays most of KEDIT's SET options. The SET options let you tailor KEDIT to match your preferences.

You can have KEDIT automatically start up each time with your preferred settings by establishing a KEDIT profile called WINPROF.KEX or by using Options Save Settings.

Dialog box options

Category

Select the category for the type of setting you are interested in. The category you select determines which SET options are displayed in the Option list box.

All	All KEDIT settings that are controllable via this dialog box.
Command	Settings that affect how commands are processed, for example whether an error causes a beep or if synonyms are processed.
File Processing	Settings that control how files are read in and written out.
Initialization	Settings that affect KEDIT initialization, for example whether Insert Mode is initially on and whether the initial document window is maximized.
Interface	Settings related to the CUA and Classic interfaces.
Macro	Settings that affect how macros are processed.
Miscellaneous	Miscellaneous KEDIT settings.

Target	Settings that control how targets are handled, for example whether searches must match the case of the search string or if the search should be restricted to certain columns.
Text Display	Settings that affect which lines of your file are displayed and how they are displayed, for example whether syntax coloring is enabled, which columns of each line are displayed, and whether the highlighting facility is enabled.
Text Entry	Settings that affect text entry and word processing facilities, such as your margins and whether wordwrap is in effect.
Window Layout	Settings that control your frame window and document window layout, for example whether scroll bars are displayed, and whether a scale line is displayed.

Option

A list of the KEDIT settings belonging to the category selected in the Category list box is displayed. The value of the option selected in this list box is displayed in the Settings area.

Settings

Displays the value of the currently-selected Option, and lets you make changes to this value. A short description of the current Option is shown near the bottom of the dialog box. If you move the mouse pointer over any of the items in the dialog box, the description will change to information specific to that item. A detailed description of the current Option is shown if the Help button is pressed.

When you change the value of an option, the change will take effect when you use the OK button to close the dialog box, or when you use the Settings list box to select another option. Note that if you want to change the values of several options, you do not need to press the OK button and close the dialog box after setting each option; if you select a new value for one option, that value is immediately put into effect if you use the Settings list box to move to a different option.

Unless you press the Save Setting button, changes to most options affect only the current editing session and do not automatically affect future KEDIT sessions.

Reset

Resets the current Option to the value it had when you initially began to display the option.

Save Setting

Saves the value of the current setting in the Windows registry so that the new value will be put into effect at the start of future editing sessions. The Savable box at the bottom right of the dialog box indicates whether the current option can be saved for future sessions, cannot be saved for future sessions, or is automatically saved for future sessions.

KEDIT Default

Resets the value of the current setting to the KEDIT's built-in default.

Command

A read-only field at the bottom of the dialog box that displays the SET command, as it would be issued from the KEDIT command line, that corresponds to the currently-displayed setting.

Level

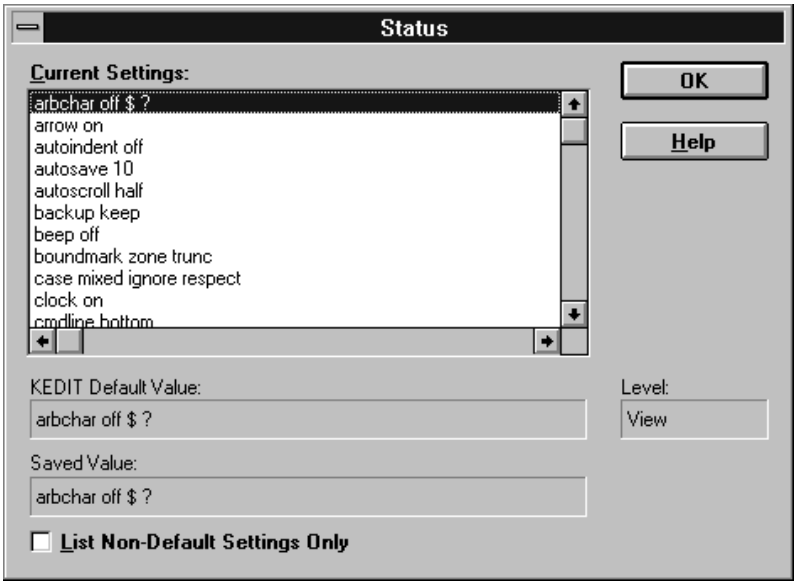
A read-only field at the bottom of the dialog box that displays the ''level'' of the currently-displayed setting. Some SET options are at the Global level, affecting your entire KEDIT session. Some options are at the File level, affecting only the current file. Other options are at the View level, and can be different for each view you have of a file that is displayed in multiple windows.

Savable

A read-only field at the bottom of the dialog box that indicates whether the current setting can be saved in the Windows registry so that it will affect future sessions. ''Yes'' is displayed for most settings, indicating that you can use the Save Setting button to save the value in the Windows registry and that the option is one of those saved when you use Options Save Settings. ''No'' is displayed for settings that cannot be saved in the Windows registry. ''Auto'' is displayed for the few options that are automatically saved in the Windows registry whenever you change their value.

See also Status (Options Menu), Save Settings (Options Menu), Chapter 9, "Tailoring KEDIT"

5.4.4 Status...



Displays the status of most KEDIT options, as well as KEDIT's default setting for each option and the value saved in the Windows registry. For those settings that can vary from file to file or from view to view of a single file, the setting for the current file or the current view of the file is given.

Dialog box options

Current Settings

Select the setting that you are interested in from the list of current settings.

KEDIT Default Value

Displays KEDIT's built-in default for the setting you selected in the Current Settings list.

Saved Value

Displays the saved setting in the Windows registry for the setting you selected in the Current Settings list.

List Non-Default Settings Only

When this box is checked, only the settings that are different from KEDIT's built-in defaults will be listed in the Current Settings list.

Notes

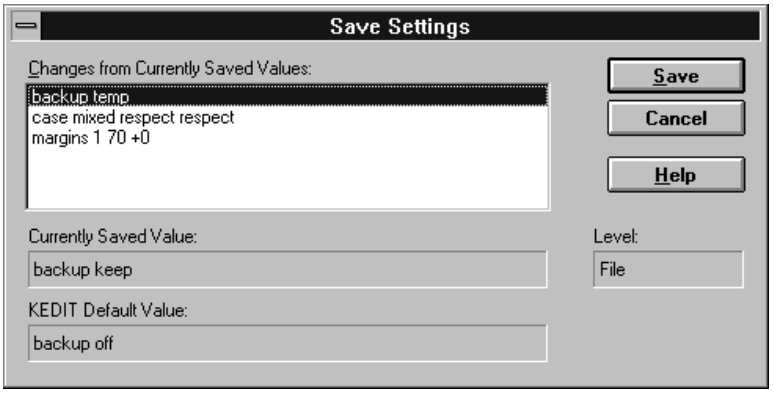
As you start up KEDIT for Windows, settings are changed from their KEDIT defaults in this order:

Any settings in the Windows registry are applied. These are values saved in previous editing sessions via Options Save Settings or via the Save Setting button of the Options SET Command dialog box.

Your KEDIT profile is processed and any SET commands that it contains take effect.

During your KEDIT session, you can make additional changes to your settings by using the Options SET Command dialog box, by issuing SET commands from the command line, or by running macros that issue SET commands.

5.4.5 Save Settings...



Saves the current values of most KEDIT settings to the Windows registry so that they can be put into effect at the start of future editing sessions.

Options Save Settings displays a list of all savable settings whose values differ from their currently-saved values, so that you can be sure of exactly which values will be saved for future sessions. You can then either go ahead with or cancel the save operation.

Dialog box options

Changes from Currently Saved Settings

Displays a list of all settings whose values differ from those currently saved in the Windows registry.

Currently Saved Value

Displays the currently-saved Windows registry value for a selected setting.

KEDIT Default Value

Displays KEDIT's built-in default value for a selected setting.

Save

Saves all of the values in the Changes from Currently Saved Values list box to the Windows registry. These values will then be put into effect at the start of future KEDIT sessions.

Level

Indicates the level which the option affects. Some SET options are at the Global level, affecting your entire KEDIT session. Some options are at the File level, affecting only the current file. Other options are at the View level, and can be different for each view you have of a file that is displayed in multiple windows.

Notes

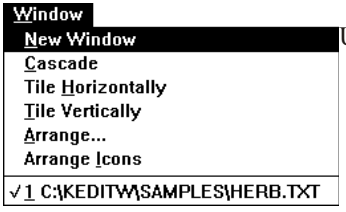
If the values of all of your settings match the values that are already saved in the Windows registry, no changes need to be made to the registry, so KEDIT displays a message box informing you about this and does not display the Save Settings dialog box.

You can also save individual settings from the Options SET Command dialog box by pressing the Save Setting button.

Not all of the option values are actually written to the Windows registry. To speed things up, KEDIT only stores in the registry values that differ from KEDIT's built-in defaults.

See Also SET Command (Options Menu), Chapter 9, "Tailoring KEDIT"

5.5 Window Menu



Use this menu to create, rearrange, and move between document windows.

5.5.1 New Window

Creates a new document window displaying an additional view of the file in the current window.

5.5.2 Cascade

Cascades your document windows within the frame window, with each window slightly offset from the others.

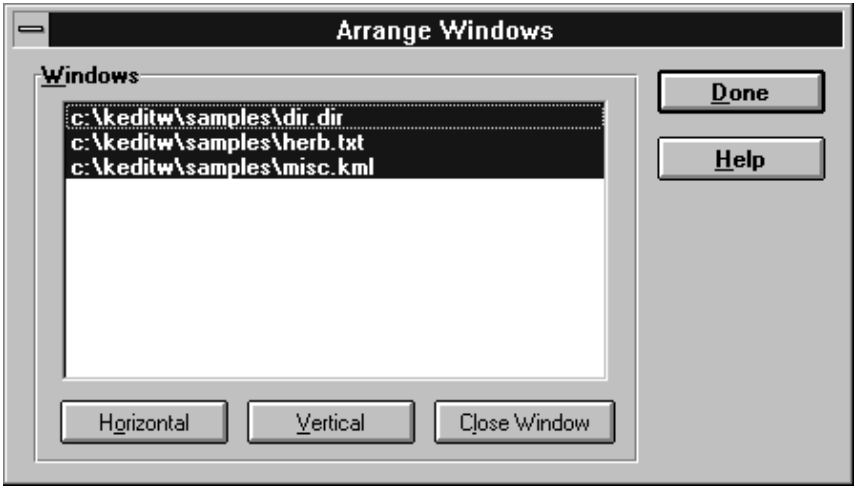
5.5.3 Tile Horizontally

Arranges your document windows horizontally within the frame window.

5.5.4 Tile Vertically

Arranges your document windows vertically within the frame window.

5.5.5 Arrange...



Arranges the document windows that you select by placing them either vertically or horizontally within the frame window. Also lets you close selected document windows.

Dialog box options

Windows

Displays a list of all the document windows so you can select which windows you want to arrange on the screen.

Initially, all document windows are selected. You can use cursor up, cursor down, or mouse button 1 to select an individual document window, and can use button 2 or Ctrl+button 1 to select additional windows.

Horizontal

Arranges the selected windows horizontally.

Vertical

Arranges the selected windows vertically.

Close Window

Closes the selected document windows. If you close the last window viewing a file and the default of SET OFPW ON is in effect, KEDIT will remove the file from the ring after asking whether to write any unsaved changes to disk.

Done

Closes the dialog box.

5.5.6 Arrange Icons

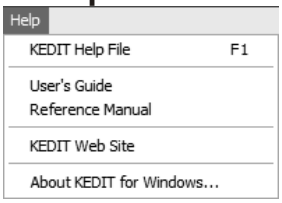
Arranges any minimized document window icons at the bottom of the frame window.

5.5.7 Document Window List

At the bottom of the Window menu is a list of all your document windows. You can select a window from this list to make it the active document window.

If there are more than nine document windows, only the first nine windows are listed, and a More Windows... item displays a dialog box that lets you choose from a complete list of all document windows.

5.6 Help Menu



Use this menu to access KEDIT's online Help information.

5.6.1 KEDIT Help File

Access the KEDIT Help file, which has full documentation on KEDIT for Windows in the standard Microsoft HTML Help format.

Shortcuts Press F1.

5.6.2 User's Guide

Displays the printable KEDIT for Windows User's Guide. This is a PDF file, and to view it you will need to have Adobe Acrobat or an equivalent program installed on your computer. Note that, aside from some of the details from the Selective Editing and Highlighting chapter of the User's Guide, the entire contents of the User's Guide is built into the KEDIT Help file.

5.6.3 Reference Manual

Displays the printable KEDIT for Windows Reference Manual. This is a PDF file, and to view it you will need to have Adobe Acrobat or an equivalent program installed on your computer. Note that the entire contents of the User's Guide is built into the KEDIT Help file.

5.6.4 KEDIT Web Site

Opens your Internet browser and displays the KEDIT home page from http://www.kedit.com.

5.6.5 About KEDIT for Windows...

Displays KEDIT for Windows' copyright and version information.

5.7 Top Toolbar

This toolbar appears by default for most files you edit. A different default toolbar is displayed when you are editing DIR.DIR files.

5.7.1 New File

Shortcut for the File New menu item, used to edit a new file. The file will have as a temporary name the first available name of the form UNTITLED.1, UNTITLED.2, etc.

If you make changes to the file and then save these changes to disk, KEDIT will prompt you for a permanent name for the file.

 New File button

See also New (File Menu), Section 3.5, "Editing Multiple Files"

5.7.2 Open File

Invokes the File Open dialog box, which allows you to begin editing additional files.

 Open File button

See also Open (File Menu), Section 3.5, "Editing Multiple Files"

5.7.3 Save File

Shortcut for the File Save menu item, used to save the current file to disk.

 Save File button

See also Save (File Menu), SAVE (Command Line)

5.7.4 Print File

By default, this displays the File Print dialog box, which lets you print either your entire file or a marked area within your file.

If you would prefer that the Print File button print your file immediately, and not display the File Print dialog box, uncheck the ''Print File Toolbar Button Shows This Dialog'' checkbox within the File Print dialog box. The Print File button will then print the marked block within your file if there is one, and otherwise will print your entire file.

 Print File button

See also Print (File Menu), PRINT (Command Line)

5.7.5 Quick Find

 Provides a fast way to access most of the features of the Edit Find dialog box.

To use this toolbar item:

Enter a search string in the edit field or select a recent search string from the list displayed by clicking the down arrow beside the edit field.

Press Enter to search forward through the file for the string.

Press Shift+Enter to search backward through the file for the string.

Press Escape to leave Quick Find mode.

See also Find (Edit Menu), Find Next (Top Toolbar), LOCATE (Command Line)

5.7.6 Find Next

Searches forward or backward through the file for the next or previous occurrence of the search string that is displayed in the Quick Find toolbar item.

 Find Next button

To search forward for the search string, click on the Find Next button. To search backward, hold down the Shift key and click on the Find Next button.

See also Find (Edit Menu), Quick Find (Top Toolbar), LOCATE (Command Line)

5.7.7 Find Dialog Box

Activates the Edit Find dialog box.

 Find Dialog Box button

See also Find (Edit Menu)

5.7.8 Previous File

Makes the previous file in the ring of files you are editing become the current file.

 Previous File button

5.7.9 Next File

Makes the next file in the ring of files you are editing become the current file.

 Next File button

5.7.10 Undo

Shortcut for the Edit Undo menu item. Use it to undo one level of changes to the current file. Undo can be used repeatedly to undo multiple levels of changes.

 Undo button

See also Redo (Top Toolbar), Undo (Edit Menu), Section 3.13, “The Undo Facility”

5.7.11 Redo

Shortcut for the Edit Redo menu item. Use it to redo one level of changes to the current file. Redo is only available if you have recently used the undo facility and have not made any intervening changes to your file. If you have used Undo repeatedly to undo multiple changes, you can use the Redo button repeatedly to redo those changes.

 Redo button

See also Undo (Top Toolbar), Redo (Edit Menu), REDO (Command Line), Section 3.13, "The Undo Facility"

5.7.12 Cut to Clipboard

Shortcut for Edit Cut menu item. The contents of the current block or selection is placed in the clipboard, replacing any existing clipboard contents, and the block or selection is deleted from your file.

 Cut button

See also Cut (Edit Menu)

5.7.13 Copy to Clipboard

Shortcut for Edit Copy menu item. Copies the contents of the current block or selection to the clipboard, replacing any existing clipboard contents.

 Copy button

See also Copy (Edit Menu)

5.7.14 Paste from Clipboard

Shortcut for Edit Paste menu item. Copies the text that is in the clipboard into your file or to the command line, inserting it at the cursor position. Multi-line clipboard text can only be pasted into your file, and not to the command line.

 Paste button

See also Paste (Edit Menu)

5.8 Bottom Toolbar

This toolbar does not appear by default. To turn it on, use the TOOLBAR option within the Options SET Command dialog box or issue SET TOOLBAR ON BOTH from the command line.

5.8.1 Copy Block

Copies a persistent block to the cursor position. The block remains marked at its new location.

 Copy Block button

See also COPY (Command Line)

5.8.2 Move Block

Moves a persistent block to the cursor position. The block remains marked at its new location.

 Move Block button

See also MOVE (Command Line)

5.8.3 Overlay Block

Overlays the text at the cursor position with the contents of a persistent block. The block remains marked at its new location.

 Overlay Block button

See also OVERLAYBOX (Command Line)

5.8.4 Delete Block

Deletes a KEDIT block or selection.

 Delete Block button

See also DELETE (Command Line)

5.8.5 Shift Block Left

Shifts the text in a block one character to the left.

 Shift Block Left button

See also SHIFT (Command Line)

5.8.6 Shift Block Right

Shifts the text in a block one character to the right.

 Shift Block Right button

See also SHIFT (Command Line)

5.8.7 Uppercase Block

Shortcut for the Actions Uppercase menu item. Uppercases the text in a KEDIT block or selection.

 Uppercase Block button

See also UPPERCASE (Command Line)

5.8.8 Lowercase Block

Shortcut for the Actions Lowercase menu item. Lowercases the text in a KEDIT block or selection.

 Lowercase Block button

See also LOWERCASE (Command Line)

5.8.9 Leftadjust Block

Left-adjusts the text in a block.

 Leftadjust Block button

If the block is a line block, the text is left-adjusted to the left margin column. The contents of box blocks and one-line stream blocks are left-adjusted to the leftmost column of the block and text outside the block is not affected. You cannot left-adjust a multi-line stream block.

See also LEFTADJUST (Command Line)

5.8.10 Rightadjust Block

Right-adjusts the text in a block.

 Rightadjust Block button

If the block is a line block, the text is right-adjusted to the right margin column. The contents of box blocks and one-line stream blocks are right-adjusted to the rightmost column of the block and text outside the block is not affected. You cannot right-adjust a multi-line stream block.

See also RIGHTADJUST (Command Line)

5.8.11 Fill Block

Prompts you to enter a fill character, and then fills the currently-marked block with copies of that character.

 Fill Block button

If the fill character is a space character, the block will be blanked out.

See also FILL (Command Line)

5.8.12 Set Bookmark1

Names the focus line as Bookmark1.

 Set Bookmark1 button

You can later use the Go to Bookmark1 button on the bottom toolbar to return to this line.

See also Bookmark (Actions Menu)

5.8.13 Go to Bookmark1

After using the Set Bookmark1 button to give a line the name Bookmark1, you can later use the Go to Bookmark1 button to return to that line.

 Go to Bookmark1 button

See also Bookmark (Actions Menu)

5.8.14 Hide Excluded Lines

Displays only the subset of the file that you have previously selected using the Edit Selective Editing dialog box or the ALL command.

 Hide Excluded Lines button

You can use this button in conjunction with the Show All Lines button  to toggle between editing only the lines you previously selected with Edit Selective Editing or the ALL command and editing all of the lines in the file.

See also Selective Editing (Edit Menu), ALL (Command Line), Chapter 8, "Selective Line Editing and Highlighting"

5.8.15 Show All Lines

Displays all lines of your file, even lines that have been excluded via the Edit Selective Editing dialog box or the ALL command.

 Show All Lines button

You can use this button in conjunction with the Hide Excluded Lines button to toggle between editing only the lines you previously selected with Edit Selective Editing or the ALL command and editing all the lines in the file.

See also Selective Editing (Edit Menu), ALL (Command Line), Chapter 8, "Selective Line Editing and Highlighting"

5.9 Top Toolbar for DIR.DIR File

This toolbar appears by default when the active file is a DIR.DIR file. DIR.DIR files contain directory listings and can be created through the File Directory dialog box and by issuing the DIR command from the command line. Most items on the DIR.DIR toolbar are the same as on the top toolbar that is used for files other than DIR.DIR files, but the clipboard cut, copy, and paste buttons are replaced by buttons that help you sort and manipulate directory listings.

5.9.1 Sort by Name

Sorts a DIR.DIR file into alphabetical order according to the file name.

 Sort by Name button

See also Section 3.9, “The DIR.DIR File”, DIRSORT (Command Line)

5.9.2 Sort by Extension

Sorts a DIR.DIR file into alphabetical order according to the file extension.

 Sort by Extension button

See also Section 3.9, “The DIR.DIR File”, DIRSORT (Command Line)

5.9.3 Sort by Size

Sorts a DIR.DIR file according to the size of each file that is listed, with the largest files listed first.

 Sort by Size button

See also Section 3.9, “The DIR.DIR File”, DIRSORT (Command Line)

5.9.4 Sort by Date

Sorts a DIR.DIR file according to the date and time that each file was last changed, with the newest files listed first.

 Sort by Date button

See also Section 3.9, “The DIR.DIR File”, DIRSORT (Command Line)

5.9.5 Parent Directory

Replaces the contents of the DIR.DIR file with a listing of the parent directory of the file listed at the cursor position.

 Parent Directory button

See also Section 3.9, “The DIR.DIR File”

5.10 Top Toolbar for Empty Ring

This toolbar appears by default when no files are being edited.

5.10.1 New File

Shortcut for the File New menu item, used to edit a new file. The file will have as a temporary name the first available name of the form UNTITLED.1, UNTITLED.2, etc. If you make changes to the file and then save these changes to disk, KEDIT will prompt you for a permanent name for the file.

 New File button

See also New (File Menu), Section 3.5, "Editing Multiple Files"

5.10.2 Open File

Invokes the File Open dialog box, which allows you to select an existing file to edit.

 Open File button

See also Open (File Menu), Section 3.5, "Editing Multiple Files"

5.10.3 Directory

Invokes the File Directory dialog box, which lets you change KEDIT's current directory or obtain a DIR.DIR directory listing.

 Directory button

See also Directory (File Menu)

5.10.4 Exit KEDIT

Terminates your KEDIT session.

 Exit KEDIT button

Chapter 6. Targets

When using KEDIT's command line and when writing KEDIT macros, you often need to refer KEDIT to some location in your file. You might want KEDIT to do something to the 127th line of the file, or to the line 82 lines below where you are now, or to the next line containing the word "Washington". To pick out locations in your file, you give KEDIT a *target* describing the line you want. This chapter discusses the different kinds of targets used by KEDIT and how to work with them.

When you issue commands from KEDIT's command line, the most common use of targets is to specify which line you want to be the *current line*, which is normally displayed in the middle of the screen and which has a box drawn around it when the cursor is on the command line.

You can think of the current line as being pointed to by a *current line pointer*, which can move up or down in your file to make different lines become the current line. By specifying which line is to be the current line, you determine which portion of the file you will see through the screen's window into the file. The window will show you the current line and the lines immediately above and immediately below it.

6.1 Types of Targets

6.1.1 Absolute Line Number Targets

A colon (":") followed immediately by a number is used to specify a target by its *absolute line number*. For example, to move the current line pointer to line 16 of your file, making that line the current line, you could enter

```
locate :16
```

from the command line. The LOCATE command tells KEDIT to locate a target and to make the target line become the current line. LOCATE is performed so frequently that KEDIT provides a shortcut: if you omit the word LOCATE and simply enter a target, KEDIT assumes that you want to locate the target. So an equivalent way to make line 16 be the current line would simply be

```
:16
```

Semicolons (";") are also accepted with absolute line number targets. ";16" is equivalent to ":16".

6.1.2 Relative Line Number Targets

You can also determine the current line by giving a *relative line number.* For example, entering the command

```
locate 5
```

or the equivalent command

```
5
```

tells KEDIT to move the current line pointer down five lines. The line that is five lines below the current line becomes the new current line. If line 20 of the file is the current line when you enter the ''5'', line 25 will become the current line. '':5'', on the other hand, would make line 5 become the current line. This is the difference between using absolute and relative line numbers in a target. With absolute line number targets, you give KEDIT the number of the line that you want to make the current line. With relative line number targets, you give KEDIT a number that is added to the line number of the current line, yielding the line number of the new current line.

You can also use negative numbers as relative line numbers. For example,

```
-5
```

would cause 5 to be subtracted from the current line number. So if line 20 is the current line, ''-5'' will make line 15 the new current line.

An asterisk (''*'') can be used as a special target meaning the end-of-file line. ''-*'' is a special target meaning the top-of-file line. (The target '':0'' would also make the top-of-file line become the current line.)

6.1.3 String Targets

String targets are a third form of target. You give KEDIT a string of characters that is contained in the line that you want to become the new current line. KEDIT searches through the file for you, starting one line below the current line and examining each succeeding line, until it finds a line containing the desired string. This saves you the trouble of looking for the string yourself and having to tell KEDIT what line it is in.

For example, assume you know that somewhere in your file is the string ''Hello''. You are not sure where it is, but you would like the line containing it to be the current line so that you can make some changes in it. To locate it, you could enter the following on the command line:

```
locate /Hello/
```

or, equivalently,

```
/Hello/
```

This tells KEDIT to look through the file, examining in turn each line after the current line, until it finds one containing the string ‘‘Hello’’. That line is then made the new current line.

Note that KEDIT’s target highlighting facility, controlled by SET THIGHLIGHT, highlights string targets that you find using LOCATE and CLOCATE commands. So, in our example, the string ‘‘Hello’’ would now be highlighted. (If you do not like this effect you can turn off target highlighting with SET THIGHLIGHT OFF.)

String target delimiters

The slash (‘‘/’’) characters around the string ‘‘Hello’’ in our example are *delimiters* that are not part of the string searched for. KEDIT uses the delimiters to determine exactly what string of characters you want it to search for. The string you are looking for is always surrounded by a pair of delimiter characters; these are normally slashes. For example, the following two string targets

```
/a/
/a /
```

are different. The first string target tells KEDIT to look for the next line containing the letter ‘‘a’’. The second, since a blank is also within its delimiters, tells KEDIT to look for an ‘‘a’’ followed by a blank. To see the difference, notice that the letter ‘‘a’’ is found in the word ‘‘bank’’, while ‘‘a’’ followed by a blank is not.

Although the slash character (‘‘/’’) is normally used as the delimiter character, other special characters can be used. Special characters are characters other than letters, numbers, and blanks. A number of special characters need to be avoided, however, because they have some special meaning to KEDIT: asterisk (‘‘*’’), colon (‘‘:’’), semi-colon (‘‘;’’), caret (‘‘^’’), ampersand (‘‘&’’), plus sign (‘‘+’’), period (‘‘.’’), minus sign (‘‘-’’), equal sign (‘‘=’’), and tilde (‘‘~’’). The final restriction on delimiters is that they cannot occur in the string that they delimit. So

```
/a/b/
```

is not a valid string target, since the string ‘‘a/b’’ contains the delimiter that was used, ‘‘/’’. To search for ‘‘a/b’’ you would have to use some other delimiter. A ‘‘,’’ would work:

```
,a/b,
```

6.1.4 Word Targets

String targets normally look for any occurrence of the string that you specify. For example

```
/age/
```

will locate the next occurrence of the string ‘‘age’’ wherever it occurs, even if it is part of another word. So, for example, ‘‘age’’ would be found in a line that contained the word ‘‘language’’.

Since it is sometimes useful to isolate just the occurrences of a particular word, as opposed to all occurrences of a given string, KEDIT provides *word targets*. Word targets work just like normal string targets, except that they are specified by preceding the string target with the word WORD (minimal truncation W). So, you can say

```
locate word /age/
```

to find a line containing the word "age" but not a line containing the word "language".

Note that when specifying targets for LOCATE that start with alphabetic characters, such as the word target in the preceding example, you must include LOCATE (or the minimal truncation L).

Prefix and suffix targets

Similar to word targets are *prefix* and *suffix* targets. When you specify a PREFIX target (minimal truncation P) KEDIT will look for occurrences of a string appearing at the start of a word. So, for example,

```
locate prefix /age/
```

would find the word "age" and the word "ageless", since both words start with the string "age", but it would not find the word "language".

When you specify a SUFFIX target (minimal truncation S) KEDIT will look for occurrences of a string appearing at the end of a word. So, for example,

```
locate suffix /age/
```

would find "age" and would find "language", but would not find "ageless".

6.1.5 More About String Targets

Backward string search

KEDIT normally starts at the line below the current line and searches forward in the file for a string target. It is sometimes useful to have KEDIT search backward starting at the line above the current line and looking in turn at each previous line. You tell KEDIT you want a *backward string search* by putting a minus sign ("-") before your string target. For example,

```
-/Hello/
```

will cause KEDIT to search backward in the file for the string "Hello".

```
-word /age/
```

will cause KEDIT to search backward in the file for the word "age".

Negative string search

It is also occasionally useful to look for the first line that does *not* contain a specified string. This is known as a *negative string search*, and you request it by preceding your string target with a tilde ("~") or a caret ("^"). For example,

```
~/Hello/
```

tells KEDIT to search for the first line below the current line that does *not* contain the string ‘‘Hello’’, and

```
~word /age/
```

tells KEDIT to search for the first line below the current line that does *not* contain the word ‘‘age’’.

Combining string targets

You can use logical operators to combine string targets. The ‘‘and’’ operation is indicated with an ampersand (‘‘&’’) and the ‘‘or’’ operation is indicated by a vertical bar (‘‘|’’). (The or operator is ASCII character 124, which appears on most U.S. keyboards as a split vertical bar, located above the backslash key.) Some examples:

```
/a/ & /b/ & /c/
```

This tells KEDIT to look for the next line containing an ‘‘a’’ and a ‘‘b’’ and a ‘‘c’’.

```
/Washington/ | /Lincoln/
```

This tells KEDIT to look for the next line that contains either ‘‘Washington’’ or ‘‘Lincoln’’.

```
/a/ | word /the/
```

This tells KEDIT to look for the next line that either contains the letter ‘‘a’’ or the word ‘‘the’’.

Regular expressions

Regular expressions are a special kind of string target that let you match more general string patterns than ordinary string targets. Regular expressions are more powerful but also more complicated than standard string targets and are discussed separately in Section 6.6, “Regular Expressions”.

Related SET options

Several SET command options affect how KEDIT carries out string searches. Be sure to read the full descriptions of the following SET command options in Reference Manual Chapter 4, “The SET Command”.

SET ARBCHAR Controls whether ‘‘wildcard’’ characters are allowed in string targets.

SET CASE Controls how uppercase and lowercase letters in your search string are handled.

SET HEX Lets you use special hexadecimal and decimal notation to specify string targets.

SET STAY Controls the location of the current line after you search for a string target that is not found.

SET VARBLANK Controls how KEDIT matches text that contains several blanks in a row.

SET WRAP Allows certain string target searches to wrap around to the top portion of the file if the target is not found below the current line.

SET ZONE Controls which columns of your text KEDIT searches when looking for string targets.

6.1.6 Named Line Targets

KEDIT also provides *named line* targets. You can give a line a name by using the SET POINT command. This name does not become a part of your file, but is merely a name tag that KEDIT associates with the line while you are editing the file.

```
set point .abc
```

tells KEDIT to give the current line the name tag ".ABC". (All such name tags must begin with a period (".").) Once you have named a line, and have then moved the current line pointer to somewhere else in the file, you can return to the line by using a named line target:

```
.ABC
```

This tells KEDIT to make the line that you named ".ABC" with some previous SET POINT command become the new current line.

You can also use the Actions Bookmark dialog box to work with named lines.

6.1.7 Line Class Targets

With *line class* targets you can look for a line based on some attribute of the line. The classes of lines that you can specify are: lines that are blank, lines whose flag bits are set to particular values, and lines that have a certain selection level.

BLANK targets Use the BLANK target (minimal truncation BLA) to look for a blank line. For example,

```
locate blank
```

will tell KEDIT to look for the next blank line in your file.

Note that when KEDIT is looking for a blank line it considers the line to be blank if it contains no nonblank characters within the current zone setting. (So a line can have nonblank characters outside the current zone and still be considered blank.)

```
locate -blank
```

tells KEDIT to look backward in your file for the first blank line occurring above the current line.

```
locate ~blank
```

tells KEDIT to look for the next line in your file that is not blank.

You can combine line class targets with string targets. So, for example, you can use the command

```
locate /Fred/ | blank
```

to locate the next line that either contains the string ‘‘Fred’’ or is blank.

Flag bit targets

Other line class targets are based on the flag bits that can be associated with a line. There are three flag bits that can be associated with a line: the new bit, which is set when a line is added during the current KEDIT session; the change bit, which is set when a line has been changed during the current KEDIT session; and the tag bit, which is set when you have used the TAG command to tag a line.

The targets corresponding to the flag bits are the NEW target, the CHANGED target (minimal truncation CHA), and the TAGGED target (minimal truncation TAG). Also available is the ALTERED target (minimal truncation ALT), which finds lines that have been either added or changed during the current KEDIT session. For example

```
locate altered
```

would look for the next line that is either new or changed.

For more information about flag bits, see the description of the SET LINEFLAG command in the Reference Manual, and the description of the KEDIT highlighting facility in Chapter 8, “Selective Line Editing and Highlighting”.

Selection level targets

Finally, you can also use line class targets to locate a line by its selection level. Selection levels are discussed in Chapter 8, “Selective Line Editing and Highlighting”. You can specify

```
locate select n
```

where *n* is the selection level you are concerned with. For example,

```
locate select 5
```

will locate the next line that has a selection level of 5.

Also available is

```
locate select n m
```

where you look for a line whose selection level falls in the specified range between *n* and *m*, for example

```
locate select 1 7
```

looks for the next line whose selection level is in the range from 1 to 7.

6.1.8 Some Further Examples

You have now seen most types of KEDIT targets. (KEDIT's group targets will be discussed in Section 6.3, "Group Targets", and regular expression targets in Section 6.6, "Regular Expressions".)

Here are some further examples:

*****	Make the end-of-file line become the current line.
-*	Make the top-of-file become the current line.
:18	Make line 18 become the current line.
5	The line that is five lines below the current line becomes the current line.
/abc/	The first line located below the current line which contains the string ''abc'' becomes the current line.
-~/abc/	The first line above the current line that does not contain the string ''abc'' becomes the current line.
-prefix /abc/	The first line above the current line that contains a word beginning with ''abc'' becomes the current line.
-2	The line two lines above the current line becomes the current line.
.XYZ	The line that was previously named ''.XYZ'' via the SET POINT command becomes the current line.
/John/ & ~/Tom/	The current line pointer moves to the first line below the current line that contains ''John'' but does not contain ''Tom''.
~tagged	The current line pointer moves to the first line below the current line that does not have its tag bit set..
/abc/ | altered	The current line pointer moves to the first line below the current line that either contains the string ''abc'' or has been altered (added or changed) during the current KEDIT session.
-select 3	The current line pointer moves to the first line above the current line that has a selection level of 3.
~reg /a[0-9]b/	The current line pointer moves to the first line below the current line that does not contain an "a" followed immediately by a digit followed immediately by a "b".

6.2 Other Uses for Targets

You have seen how targets can be used to determine which line will be the current line. You can also use targets as operands of many KEDIT commands. You use these targets to tell KEDIT what area of the file the command is supposed to affect. In general, KEDIT commands that take targets as operands will affect all lines from the current line down to, but not including, the target line. (If the target is above the current line, the command will affect all lines from the current line up to, but not including, the target line.) The portion of your file operated on by a command that handles target operands is known as the *target area*.

The UPPERCASE command is one of the KEDIT commands that takes a target as an operand. The UPPERCASE command is used to capitalize all letters in some part of your file. You specify a target as the operand of the UPPERCASE command to tell KEDIT which part of the file to uppercase.

```
uppercase 1
```

will uppercase one line, the current line.

```
uppercase /abc/
```

will uppercase lines, starting with the current line and continuing down to, but not including, the next line containing the string "abc".

```
uppercase :8
```

will uppercase lines, starting with the current line and continuing up or down to, but not including, line 8.

```
uppercase .xyz
```

will uppercase lines, starting with the current line and continuing up or down to, but not including, the line named .XYZ (via the SET POINT command).

The general form of the UPPERCASE command is

uppercase *target*

where *target* is any KEDIT target. The UPPERCASE command will uppercase all lines from the current line through the target line. It will not uppercase the target line itself.

Another example of a command that uses targets as operands is the DELETE command. The DELETE command deletes all text in the target area.

```
delete 2
```

will delete the current line and the line below it, for a total of two lines.

```
delete *
```

will delete all lines from the current line through the last line of the file.

6.3 Group Targets

When targets are used as the operand of a command like UPPERCASE or DELETE (that is, a command that uses a target to determine the portion of your file that it will operate on), a special kind of target known as a *group target* is available. There are three group targets: ALL, BLOCK, and PARAGRAPH.

The ALL target causes the command to operate on all of your file. For example,

```
uppercase all
```

uses your entire file as the target area. KEDIT uppercases the text in all lines of your file. Note the distinction between this command and

```
uppercase *
```

which causes KEDIT to uppercase the text from the current line down but does not affect text in lines above the current line. Also note that the target ALL is different from the ALL command.

Another type of group target is the BLOCK target, which uses the currently marked line, box, or stream block as the target area. For example,

```
delete block
```

deletes all text in the currently marked block, while

```
print block
```

sends all text in the currently marked block to the printer. If there is no currently marked block, you will get an error message if you attempt to use a BLOCK target.

If INTERFACE CUA is in effect and you have marked a selection, as opposed to a persistent block, in your file, the BLOCK target can still be used and it will operate on the selection. The command involved would, however, need to be issued from a macro run when the cursor is in the file area (for example, a macro assigned to a key). You cannot issue a command from the command line that affects a selection, because moving the cursor to the command line would unmark your selection.

The PARAGRAPH group target (minimal truncation PARA) causes the command to operate on text in the current paragraph. Paragraphs normally have blank lines above and below them, but with the SET FORMAT command, you can control how KEDIT defines "paragraphs".

```
delete paragraph
```

tells KEDIT to delete all lines of the paragraph in which the current line is contained.

While some commands, like DELETE and PRINT, operate when any type of block is defined, others have restrictions. For example, The FLOW command and the COMPRESS and EXPAND commands work only with line blocks.

6.4 Column Targets

Much as KEDIT keeps track of a current line with a current line pointer, KEDIT uses a column pointer to keep track of a current column. The current column can range from one column to the left of the left zone column to one column to the right of the right zone column. Just as most KEDIT commands issued from the command line act with respect to the current line, KEDIT has a set of column commands that act with respect to the current column. And just as you use the LOCATE command to determine the current line, you can use the CLOCATE command to determine the current column. (See the next section for a discussion of the related focus line and focus column concepts.)

When SCALE ON is in effect, KEDIT puts a vertical bar ("|") at the column pointer position on the scale line. You can also use the QUERY COLUMN command to determine the current column.

Column commands

KEDIT's column commands are CAPPEND, CDELETE, CFIRST, CINSERT, CLAST, CLOCATE, CMATCH, COVERLAY, and CREPLACE.

KEDIT has column targets, which pick out a particular column, that are analogous to the normal KEDIT targets that pick out a particular line. Column targets are valid only as operands of the CDELETE and CLOCATE commands. The three types of column target are:

absolute column targets Give a colon (":") followed by the column number you are interested in.

`:10` refers to column 10

`:38` refers to column 38

relative column targets Give a number (optionally preceded by a minus sign) to refer to a column relative to the current column pointer.

`8` refers to the column 8 columns to the right of the current column pointer

`-4` refers to the column 4 columns to the left of the current column pointer

string column targets Give the string that you want to locate, enclosing it in delimiter characters. KEDIT searches for the string starting one column beyond (or, if the target is preceded by a minus sign, one column before) the column pointer column of the current line. If STREAM OFF is in effect, KEDIT searches only the current line. If STREAM ON, the default, is in effect, KEDIT continues the search in succeeding lines of your file.

/abc/	refers to the column containing the ‘‘a’’ of the first ‘‘abc’’ to the right of the column pointer
-/abc/	refers to the column containing the ‘‘a’’ of the first ‘‘abc’’ to the left of the column pointer

6.5 The Focus Line

When you work with KEDIT, the cursor position usually determines the line of the file on which your attention is focused. When you issue a command from the command line, your attention is focused on the current line, which is normally displayed in the middle of the document window, and which has a box drawn around it when the cursor is on the command line. KEDIT commands issued from the command line act with respect to the current line.

When the cursor is in the file area, your attention is normally focused on the line in which the cursor is located. If you press a key while the cursor is in the file area, (for example, if you press Alt+L to mark a line) then commands issued from the macro assigned to the key will act with respect to the cursor line (that is, Alt+L will mark the cursor line).

The line on which your attention is focused, the line that KEDIT commands act with respect to, is called the *focus line*.

For example, if you issue the DUPLICATE command from the command line, or if the cursor is on the command line and you press the F8 key, which issues the DUPLICATE command, then the current line is duplicated. If the cursor is on a line of your file and you press F8, then the line that the cursor is on is duplicated, regardless of whether that line is the current line.

As another example, suppose you issue the command LOCATE :10. This command tells KEDIT that you want to shift the focus of your attention to line 10 of your file. If you issue this command from the command line KEDIT makes line 10 of your file become the new current line. Since the current line is displayed in a fixed location in the window, the only way for KEDIT to make a new line become the current line is to scroll the window so that the new line is displayed at the current line location.

If you press a key that issues the LOCATE :10 command while the cursor is on a line of your file, then you are again telling KEDIT that you want to shift your attention to line 10 of your file—that is, you want to make line 10 of your file the new focus line. When the cursor is in the file area, KEDIT does not necessarily have to scroll the window to make a different line become the focus line. If the new focus line is already being displayed in the window, KEDIT can simply move the cursor to that line. If the new focus line is not already in the window, then KEDIT does need to scroll the window; KEDIT makes the new focus line become the current line, and positions the cursor on that line.

In summary, KEDIT commands act with respect to the focus line. If the cursor is on the command line, the focus line is the current line. If the cursor is on a line of your file, then the focus line is the line that the cursor is on.

Focus column

Just as KEDIT uses the concepts of the current line and the focus line, KEDIT also uses the concepts of the current column and the focus column. Section 6.4, "Column Targets", discusses KEDIT's current column. KEDIT's column commands (CLOCATE, CDELETE, etc.) actually act relative to the focus column. The focus column is the same as the current column when the cursor is on the command line or in the prefix area. When the cursor is in the file area, the focus column is the cursor column.

6.6 Regular Expressions

6.6.1 Overview

This section describes KEDIT's regular expressions, which provide pattern matching facilities that go far beyond the capabilities of ordinary string targets.

While regular expressions are more powerful than ordinary string targets, they are also somewhat more complex to learn about and to work with. If you are just getting started with KEDIT, and are not already familiar with regular expressions, you probably do not need to know how to use regular expressions yet. Instead, you might want to learn about regular expressions after you gain more experience with ordinary string targets and with commands like SET ARBCHAR.

Regular expressions, which evolved in the UNIX world, provide a powerful method for matching character patterns and have been incorporated into KEDIT as a special type of string target. We will first take an informal look at how regular expressions are used, and then give descriptions of each of the components of regular expressions.

If you have already worked with regular expressions in other applications, you may notice some differences between KEDIT's version of regular expressions and the version that you are familiar with. The general concepts involved are the same in all versions of regular expressions, but the exact details of which operations are supported and which special characters invoke those operations tend to vary.

Regular expressions as targets

Since regular expressions are a type of target, you can use them with any KEDIT commands that use targets, such as the LOCATE, CLOCATE, and ALL commands. Like other string targets, regular expressions used in KEDIT commands are enclosed in delimiters, which are most often a pair of slashes. To let KEDIT know that you are using a regular expression target, as opposed to some other type of target, you precede the expression with REGEXP, or any truncation of REGEXP, such as R or REG. So

```
locate /ab+c/
```

uses a normal string target that matches the string ‘‘ab+c’’, while

```
locate reg /ab+c/
```

uses a regular expression target that turns out to match strings consisting of an ‘‘a’’, one or more ‘‘b’’s, and a ‘‘c’’.

Regular expressions and the Edit menu

Regular expressions can also be used with KEDIT’s Edit Find, Edit Replace, and Edit Selective Editing dialog boxes. These dialog boxes use a Regular Expression check box to indicate the use of regular expressions, so you do not need to type in REGEXP, and the strings that you enter into these dialog boxes are not enclosed in delimiters. This discussion will use LOCATE commands to illustrate most of its points about regular expressions, but the principles discussed apply to other uses of regular expressions.

Text specifiers and operators

Regular expressions consist of *text specifiers* and *operators*. Text specifiers are elementary values that KEDIT attempts to match against text in your file. Operators affect the way that text specifiers are processed and are used to build more complex regular expressions.

For example,

```
locate reg /abc/
```

uses a regular expression consisting only of the text specifiers ‘‘a’’, ‘‘b’’, and ‘‘c’’. This regular expression matches the string “abc”. But the regular expression in

```
locate reg /ab+c/
```

uses the ‘‘+’’ operator, which matches one or more occurrences of whatever immediately precedes it. So this regular expression matches a string consisting of an ‘‘a’’, one or more ‘‘b’’s, and a ‘‘c’’.

Escape character

Suppose that you want to use a regular expression to match the string ‘‘ab+c’’. You will then need to specify the ‘‘+’’ character in the regular expression without having it interpreted as an operator. To make any of the characters that have special meanings within regular expressions instead serve as literal characters that match themselves, you precede them with a backslash (‘‘\’’). Backslash serves as the regular expression *escape character*. So while the regular expression in the last example matches ‘‘a’’, one or more ‘‘b’’s, and a ‘‘c’’, this example matches ‘‘ab+c’’:

```
locate reg /ab\+c/
```

In general, letters, numbers, and blanks can be used directly in regular expressions as text specifiers, but most special characters have special meanings, and must be preceded by a backslash if they are to be used as text specifiers that match themselves.

Wildcard character

Another character with a special meaning is the question mark (‘‘?’’), which is a text specifier that matches any single character. That is, ‘‘?’’ is the *wildcard character* for regular expressions. For example,

```
locate reg /a?c/
```

will look for an ‘‘a’’ followed by any other character and then by a ‘‘c’’, matching strings like ‘‘a5c’’ or ‘‘axc’’, but not ‘‘axyc’’.

Character classes

Another type of text specifier is the *character class*. Suppose you want to find strings containing the character "y" followed by a numeric digit. If KEDIT did not support regular expressions, you would need to use this lengthy combination of ordinary string targets:

```
locate /y0/|/y1/|/y2/|/y3/|/y4/|/y5/|/y6/|/y7/|/y8/|/y9/
```

But you can instead use this much shorter regular expression to match any of "y0", "y1", etc.:

```
locate reg /y[0123456789]/
```

To specify a character class, you enclose a set of characters in square brackets ("[" and "]") to form a text specifier that matches any one of the characters in the class. This regular expression can be further shortened because you can specify a range of consecutive characters within a character class by giving the first character in the range, a minus sign ("-"), and the last character. So another way to match the letter "y" followed by a digit would be:

```
locate reg /y[0-9]/
```

You can use a tilde ("~") immediately after the left bracket that introduces a character class to indicate that anything *except* the specified characters will be matched. For example,

```
locate reg /y[~0-9]/
```

matches a "y" followed by anything other than a digit. This regular expression would match "ya" or "y+", but not "y0" or "y1".

Predefined expressions

Some sequences, like [0-9] (which matches any single digit) and [a-zA-Z0-9] (which matches any single alphanumeric character), are used frequently enough in regular expressions that they are available in shorthand form. For example, ":a" is a *predefined expression* that can be used instead of [a-zA-Z0-9]. Predefined expressions consist of a colon and a lowercase letter. ":d" is the predefined expression used for [0-9], so one more way to look for a "y" followed by a digit would be:

```
locate reg /y:d/
```

Using operators

Suppose that you want to look for a string consisting of a "y" followed by one or more digits followed by a "z". The character class [0-9] can match only a single digit, and not a group of digits. But by combining this character class with an operator, you can do the job:

```
locate reg /y[0-9]+z/
```

This would match strings like "y12z" or "y343478z". The regular expression uses "+", which is known as the *minimal plus* operator. "+" operates on the immediately preceding expression (in this case the character class "[0-9]", which matches any single digit) and matches one or more occurrences of the expression (in this case it matches one or more digits).

Another frequently-used operator is ‘‘*’’, the *minimal closure* operator, which matches zero or more occurrences of an expression, as opposed to ‘‘+’’, which matches one or more occurrences of an expression. The preceding example will match ‘‘y1z’’ or ‘‘y12z’’, but not ‘‘yz’’, because no digits appear between the ‘‘y’’ and the ‘‘z’’ and

```
locate reg /y[0-9]*z/
```

will match ‘‘y1z’’ and ‘‘y12z’’, but also ‘‘yz’’.

CHANGE command

In addition to their role as string targets, regular expressions can also be used with the CHANGE and SCHANGE commands and with the Edit Replace dialog box. To indicate that you are using a regular expression, you need to follow the name of the CHANGE or SCHANGE commands with REGEXP, or a truncation like REG or R, or use Edit Replace’s Regular Expression check box. You can then use a regular expression to specify the string that is to be changed. For example,

```
change reg /y[0-9]*z/xxx/all *
```

will replace all occurrences in your file of ‘‘y’’ followed by zero or more digits and then by ‘‘z’’ with the string ‘‘xxx’’.

6.6.2 Regular Expression Text Specifiers

Here is a description of each of the text specifiers that can be used within KEDIT regular expressions.

Literal characters

Characters that have no special meaning as regular expression operators, etc. can be used directly in regular expressions as text specifiers that match themselves. Most special characters do have special meanings, and match themselves only if preceded by a backslash, as discussed next. But letters, numbers, and blanks do not have special meanings and can be used directly. For example,

```
locate reg /abc 123/
```

matches ‘‘abc 123’’.

\ Escape sequences

It is not possible to represent every individual character value as a simple character. Many characters have special meaning in regular expressions or simply cannot be input directly from the keyboard. When you want to include one of these characters in your regular expression, it must be preceded by the regular expression escape character, which is a backslash (“\”). For example,

```
locate reg /?/
```

matches any character, while

```
locate reg /\?/
```

matches the ‘‘?’’ character.

The characters with special meanings in regular expressions, all of which must be preceded by a backslash in at least some contexts if you want them to be taken literally, are the backslash itself; left and right parentheses, brackets, and braces; question mark; tilde; caret; dollar sign; asterisk; at sign; pound sign; split vertical bar; colon; ampersand; plus sign; and minus sign: \, (,), [,], {, }, ?, ~, ^, $, *, @, #, |, :, &, +, and -.

Special characters that cannot be input directly from the keyboard can be expressed with an escape sequence representing their hexadecimal or octal values:

```
\xnn or \Xnn - Hexadecimal value of character code
\nnn         - Octal value of character code
```

For example, tab characters (hexadecimal value 09) can be located with

```
locate reg /\x09/
```

You can also locate tab characters with

```
locate reg /\t/
```

This is because some control codes have been given symbolic names. They are:

\b - Backspace
\f - Formfeed
\r - Carriage return
\n - Linefeed
\t - Tab

\c Cursor positioning

The escape sequence ‘‘\c’’ is a special escape sequence that determines the cursor position after a successful regular expression search done via the Edit Find dialog. This can be useful when you want the cursor to be positioned at a particular spot in a character pattern, rather than in its normal location at the start of the pattern. For example, if you use the Edit Find dialog box to search for the regular expression ‘‘qwe\crty’’, the cursor will be positioned after the search at the ‘‘r’’ in the string ‘‘qwerty’’, and not at the ‘‘q’’. The ‘‘\c’’ escape sequence also affects CLOCATE commands, where it determines the column pointer location put into effect after a successful search.

? Wildcard character

The wildcard character, a question mark (“?”), matches any single character. For example,

```
locate reg /t?p/
```

matches “t”, followed by any other character, followed by ‘‘p’’.

[class] Character class

Character classes allow you to specify a set of characters, any one of which will be matched. The set of characters is enclosed in square brackets. For example,

```
locate reg /[abc]/
```

matches the strings “a”, “b”, or “c”. Note that character classes match only a single character, and not multi-character strings. To match multiple occurrences of characters in a character class, you need to combine the character class with an operator. For

example, to match one or more occurrences of characters from the character class [abc], you could use

```
locate reg /[abc]#/
```

which matches "a", "b", and "c", and also "abc", "cba", "abcabc", "bbbaaaccccc", etc.

A range of values can be represented in a character class by joining the first and last value in the range with a minus sign. For example, the class [0-9] matches any numeric digit. Ranges and simpler enumerations can be combined in a single class specification. For example, the class [a-zA-Z246] matches any letter and matches the digits 2, 4, and 6.

[~class] Character Unclass

Character unclass is a notational convenience that is useful when a class is complex enough that it is easier to list the characters that do not belong than to list the characters that do. You specify the characters that should not be matched, preceding the list with a tilde and enclosing it in square brackets. The character unclass will match any character *not* specified. For example,

```
locate reg /[~abc]/
```

matches any character except "a", "b", and "c".

^ Beginning of Line

The caret character ("^") matches the beginning of a line. For example,

```
locate reg /^Mary/
```

matches "Mary" in the string

```
Mary had a little lamb
```

since Mary appears at the beginning of the line; however, this command does not match "Mary" in the string

```
everywhere that Mary went
```

where ''Mary'' is not at the start of the line. If blanks optionally precede your pattern at the beginning of the line, you should include them in your regular expression. For example, you could use this command to look for a line whose first nonblank text is ''Mary'':

```
locate reg /^ *Mary/
```

The expression matches the start of a line, then uses the minimal closure operator to match zero or more blanks, and then matches ''Mary''.

Strictly speaking, the caret does not actually match the beginning of a line, but instead matches the beginning of the zone. Since the left zone is usually set to column 1, the beginning of the zone and the beginning of the line are normally the same thing. But if, for example, ZONE 6 20 is in effect,

```
locate reg /^fgh/
```

would succeed on the line

```
abcdefgh
```

while the following command would not:

```
locate reg /^abc/
```

Note that the caret character does double duty in regular expressions. If it is used at the start of a regular expression, it matches the beginning of the line. If it is used elsewhere in a regular expression, it is interpreted as the power operator, which is discussed below.

$
End of Line

The dollar sign ("$") matches the end of an individual line. For example,

```
locate reg /lamb$/
```

matches ''lamb'' when it appears at the end of this line:

```
Mary had a little lamb
```

but not when it appears in the middle of this line:

```
the lamb was sure to go
```

6.6.3 Regular Expression Operators

Regular expression operators and some related topics are discussed here. Operators are used in combination with the text specifiers discussed above to build more powerful regular expressions.

Processing order

Regular expressions are scanned from left to right. Text specifiers not followed by an operator are used to match some text in your file. When an operator is encountered, it usually applies to the text specifier that immediately precedes it. Consider these examples:

```
locate reg /abc/
```

This regular expression consists of three text specifiers: the characters ''a'', ''b'', and ''c''. This expression will match the string ''abc''.

```
locate reg /abc#/
```

In this example, the regular expression begins with the character ''a'' and the character ''b''. Then comes the maximal plus operator ''#'', applied to the character ''c''; this will match the longest possible string consisting of ''c''s. So the entire expression will match strings like ''abc'' and ''abcccc''. Note that it will not match strings like ''abcabc'', because the maximal plus operator applies only to the text specifier that immediately precedes it.

Grouping

What if you want an operator to apply to something more than a single text specifier? You can use parentheses to group together portions of a regular expression and have them treated as a unit. So the command

```
locate reg /(abc)#/
```

applies the maximal plus operator to the entire sequence ‘‘abc’’, and will match ‘‘abc’’, ‘‘abcabc’’, ‘‘abcabcabc’’, etc.

**X*
Minimal
Closure**

To match repeated occurrences of a single pattern, you use the closure and plus operators. Minimal closure matches zero or more occurrences of the pattern it is applied to, but only as many as absolutely necessary. For example,

```
locate reg /ab*/
```

only matches “a” in the string “abbbbc”, but

```
locate reg /ab*c/
```

matches the entire string because it must match “bbbb” in order to match both the “a” and the “c”.

**X+
Minimal Plus**

Minimal plus is similar to minimal closure except that it must always match at least one occurrence of an expression. For example,

```
locate reg /ab+/
```

matches “ab” in the string “abbbbc”

**X@
Maximal
Closure**

Maximal closure matches zero or more occurrences of an expression, but unlike minimal closure it matches as many occurrences of the expression as possible instead of as few as possible. For example,

```
locate reg /ab@/
```

matches “abbbb” in the string “abbbbc” and matches ‘‘a’’ in the string ‘‘ac’’.

**X#
Maximal Plus**

Maximal plus is similar to maximal closure, matching as many occurrences of an expression as possible, except that it must always match at least one occurrence of the preceding expression. So

```
locate reg /ab#/
```

matches ‘‘abbbb’’ in the string ‘‘abbbbc’’, but does not match anything in the string ‘‘ac’’.

**X^n
Power
Operator**

The power operator matches exactly *n* occurrences of X, where *n* is a number in the range 1 to 255. For example,

```
locate reg /ab^4/
```

matches the string “abbbb” in the string “abbbbc”

~X
Not Operator

The not operator doesn't really match anything. Instead, it checks to see if the immediately following item matches, and fails if it does. For example,

```
locate reg /book~s/
```

matches the string ''book'' whenever it is not immediately followed by an ''s''. So ''book'' would be matched in ''bookcase'', but not in ''bookshelf''.

```
locate reg /n~[0-9]^3/
```

This example looks for occurrences of the letter ''n'' that are not immediately followed by three digits. So it would match ''n'' in the strings ''n12x'' and ''nabc'', but not in the strings ''n123'' or ''n5678''.

(X1|X2|...|Xn)
Alternation

Alternation lets you match any one of a series of expressions. Alternation takes a sequence of expressions, enclosed in parentheses and separated by split vertical bars (character code 124), and processes each of the expressions in turn until one of them matches or they all fail.

For example,

```
locate reg /I saw (him|her) leave/
```

matches both ''I saw him leave" and ''I saw her leave".

Note that while the entire alternation is enclosed in parentheses, individual expressions in the alternation do not need to be included in parentheses. That is, the alternation in the preceding example does not need to be given as ''((him)|(her))'', although that would also be acceptable.

```
locate reg /a(b+|c+)d/
```

This looks for strings like ''abd'', ''abbbbd'', ''acd'', and ''accd''.

{X}
Tagged Expressions

Tags provide a way to ''remember'' the text that is matched by a portion of a regular expression. You can refer to this text later in the same regular expression. To use a tag, place braces around the portion of the regular expression whose matching text you want to remember. You can then refer to the matched text by using &1 later in the regular expression. For example, this command will look for a string of three identical letters (''aaa'', ''bbb'', etc.):

```
locate reg /{[a-zA-Z]}&1&1/
```

The character class [a-zA-Z] will match any letter, and since this character class is enclosed in braces, the letter that is matched can be referred to via &1, and the expression uses &1 twice to specify that whatever letter is originally matched should appear two more times.

You can tag multiple portions of the expression by using multiple pairs of braces, and then use &1 to refer to the text corresponding to the first pair, &2 for the text corresponding to the second pair, etc., through &9. For example, to look for a group of one or more letters followed by a pair of identical digits followed by the original group of letters (''abc55abc'', ''x99x'', etc.), you could use

```
locate reg /{[a-zA-Z]+}{[0-9]}&2&1/
```

The most common use of tags is in connection with the CHANGE and SCHANGE commands and with the Edit Replace dialog box, which let you use &1, &2, etc. not only in the first string (which is a regular expression specifying the string to be changed) but also in the second string (which specifies the replacement string). Note that, unlike the string to be changed, the replacement string is *not* a regular expression and none of the special characters used within regular expressions have any special meaning, except for the use of *&n* that is discussed here.

For example, suppose you want to change strings consisting of "y" followed by one or more digits followed by "z" into strings with "a" followed by the same set of digits followed by "b". "y123z" would change to "a123b", "y55552z" would change to "a55552b", etc. You could use this command to make the changes:

```
change reg /y{[0-9]+}z/a&1b/ all *
```

You can use &0 to refer to the entire string that is matched by a regular expression. For example, this command would put parentheses around all strings of one or more commas in a file:

```
change reg /,#/(&0)/all *
```

"*&n*" (an ampersand followed by a digit) in the replacement string of a change operation involving regular expressions has a special usage as a way of referring to tagged expressions. An ampersand followed by any other character has no special meaning and is taken literally. The only exception is "&&", an ampersand followed by another ampersand, which is taken as a single ampersand, so that you can use something like "&&1" in a replacement string as a way specifying the string "&1" without signalling a tagged expression. For example,

```
change reg /a{?}c/&1&x&&1/
```

would change "abc" to "b&x&1".

:letter Predefined Expressions

Several predefined expressions are provided for your convenience. They are referenced by a single lowercase letter preceded by a colon (":") For example, to search for a letter followed by a digit, you could use

```
locate reg /[a-zA-Z][0-9]/
```

or you could use predefined expressions:

```
locate reg /:c:d/
```

Here is a list of the predefined expressions:

Name	Definition	Description
:a	[a-zA-Z0-9]	Alphanumeric character
:b	([\t\x20]#)	White space (a group of blanks and tabs)
:c	[a-zA-Z]	Alphabetic character
:d	[0-9]	Digit
:h	(((0x)|)[0-9a-fA-F]#)	Hexadecimal numbers
:i	([a-zA-Z_\$][a-zA-Z0-9_\$]@)	C-language identifiers
:n	(([0-9]@.[0-9]#)|([0-9]#))	Numbers (possibly including a decimal point)
:q	(("[~"]@")|('[~']@'))	Quoted string (in single quotes or double quotes)
:w	([a-zA-Z]#)	String of alphabetic characters
:z	([0-9]#)	Integers (one or more digits)

6.6.4 Usage Notes

All of KEDIT's regular expression processing takes place on a line-at-a-time basis. This means that regular expressions cannot match text that extends from the end of one line onto the following line.

Regular expression processing is affected by SET CASE. For example, when you tell KEDIT to respect case differences, the command

```
locate reg /a[yz]/
```

will match only "ay" and "az", but when you tell KEDIT to ignore case differences, these additional strings will be matched: "aY", "aZ", "Ay", "Az", "AY", and "AZ".

Regular expression processing is affected by SET ZONE. Regular expression searches take place only with the current ZONE columns. If the left zone is set to some column greater than 1, a caret ("^") at the start of the expression, which normally matches the beginning of the line, will match the left zone column.

Regular expression processing is not affected by SET VARBLANK, SET HEX, or SET ARBCHAR. That is, during regular expression processing, KEDIT acts as if VARBLANK OFF, ARBCHAR OFF, and HEX OFF were in effect. Of course, regular expressions provide their own mechanisms for matching multiple blanks, matching wildcard characters, and specifying hexadecimal character codes.

When processing regular expressions, KEDIT considers the last nonblank character of a line to be the end of the line unless TRAILING ON is in effect, in which case trailing blanks are also considered to be part of the line. Since regular expression searches look only within the current zone columns, there will be no match for "$" (end-of-line) in lines that end before the left zone column or that extend beyond the right zone column.

There is no implied linefeed or carriage return at the end of each line. Linefeed and carriage return characters can be used within a regular expression, but will only match linefeed or carriage return characters that occur within a line. Use "$" to match the end of the line.

Because of the complex pattern matching that goes on, regular expression processing can be slower than normal target processing. The performance issues are usually not significant, but it is possible to construct regular expressions that force large amounts of backtracking within the pattern matcher and take a very long time to process.

6.6.5 Regular Expression Summary

Here is a summary of the components of regular expressions:

Item	Meaning
?	Wildcard character – matches any single character
^	Matches the beginning of a line
$	Matches the end of a line
[class]	Definition of a character class – matches any character in class
[~class]	Definition of a character unclass – matches characters not in class
(X)	Parenthetical expressions – groups expressions together for other operations
X*	Minimal closure – matches shortest possible string of zero or more occurrences of X
X+	Minimal plus – matches shortest possible string of one or more occurrences of X
X@	Maximal closure – matches longest possible string of zero or more occurrences of X
X#	Maximal plus – matches longest possible string of one or more occurrences of X
X^n	Power function – matches exactly n occurrences of X
~X	Not function – succeeds only if X isn't matched
(X1|X2|...)	Alternation – matches X1 or, if X1 doesn't match, matches X2, etc.
:letter	Predefined expression
\x	Escape sequence
\c	Set cursor position/current column after Edit Find or CLOCATE
{X}	Tagged expression – when X is matched the value is saved for later reference
&n	Reference to value of *n*th tagged expression

Chapter 7. The Prefix Area

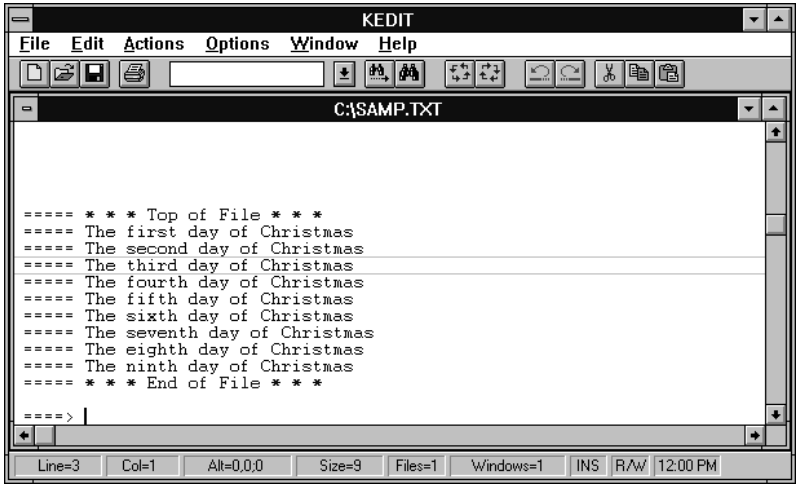
7.1 Prefix Commands

KEDIT's *prefix area* is an optional feature that allows you to enter special *prefix commands* that can add, move, copy, shift, exclude, show and delete lines of text. IBM's XEDIT text editor makes use of a prefix area, and the prefix area is included in KEDIT largely for compatibility with XEDIT. Almost everything that can be done with the prefix area can also be done, often in a more flexible manner, using key assignments discussed in Chapter 4, "Keyboard and Mouse", and summarized later in this chapter.

The prefix area is turned off by default. To turn the prefix area on, issue the command

```
SET PREFIX ON
```

With the prefix area enabled, your screen will look something like this:



The prefix area normally appears as five equal signs ("====="), but if you issue the command

```
SET NUMBER ON
```

then line numbers will appear instead of equal signs. (The width of the prefix area can be changed with the SET PREFIXWIDTH command if you need to edit large files with more than the 99999 lines whose line numbers will fit within five characters.)The equal signs and line numbers are not part of your text and are not written to disk when your file is saved.

The prefix area is usually placed to the left of your text, but you can have it displayed to the right of your text by issuing the command

```
SET PREFIX ON RIGHT
```

The prefix area is there so that you can enter prefix commands. For example, to use the prefix area to add a line after the fourth line in the window, you would put an "A" in the prefix area of that line and then press the F12 key (if you are using INTERFACE CUA) or the Home key (if you are using INTERFACE CLASSIC) to have the command carried out.

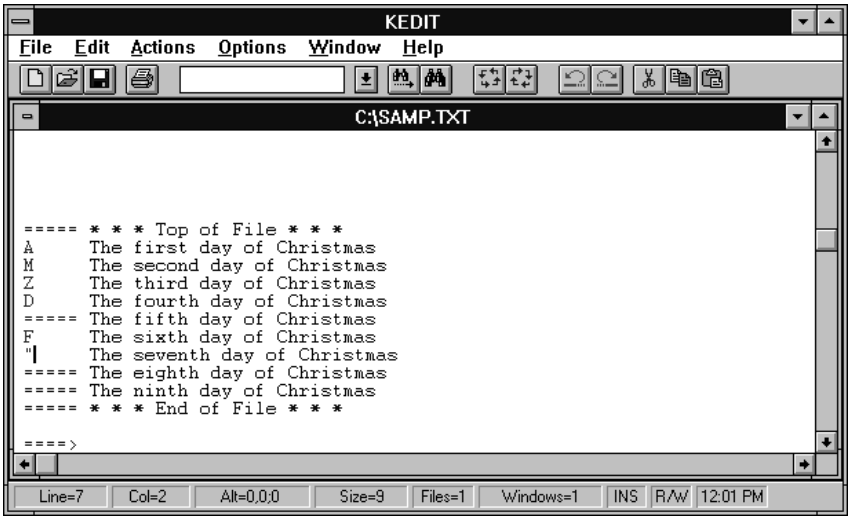
The basic prefix commands are:

A	Adds a blank line to your file.
C	Indicates a line that is to be Copied.
D	Deletes a line from your file.
F	Indicates the line following which text will be moved or copied. (Used in conjunction with "C" or "M".)
I	Inserts a blank line into your file—same as "A".
L	Lowercases a line.
M	Indicates a line that is to be Moved.
P	Indicates the line Preceding which text will be moved or copied. (Used in conjunction with "C" or "M".)
S	Shows excluded lines represented by a shadow line. The S prefix command is valid only if issued from the prefix area of a shadow line. See Chapter 8, "Selective Line Editing and Highlighting", for discussion of shadow lines, excluded lines, and related topics.
U	Uppercases a line.
X	eXcludes a line.
/	Indicates a line that is to become the new current line.
"	Indicates a line that is to be duplicated.
<	Indicates a line that is to be shifted left 1 column. Text from the left zone column through the truncation column will be shifted.
>	Indicates a line that is to be shifted right 1 column. Text from the left zone column through the truncation column will be shifted.
SCALE	Indicates that the scale line is to be displayed in this line. (Similar to the SET SCALE command.)
TABL	Indicates that the tab line is to be displayed in this line. (Similar to the SET TABLINE command.)
.name	Gives a line a name. (Similar to the SET POINT command.)

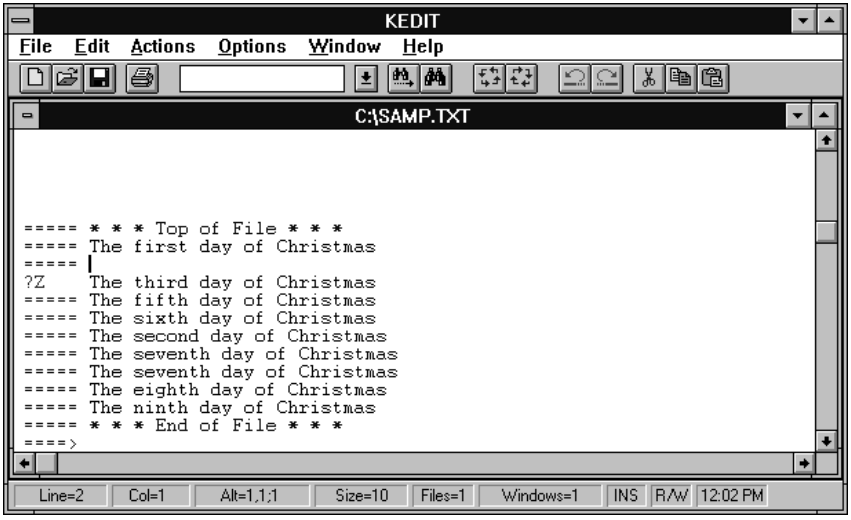
You can move the cursor into and out of the prefix area by using the cursor left and cursor right keys. When the cursor is in the prefix area, characters that you enter are interpreted not as text to be placed in your file, but as prefix commands to be executed. Prefix commands are not actually executed until you press the F12 key (or, with INTERFACE CLASSIC, the Home key). You can enter prefix commands into the prefix areas of several lines before pressing the key; all of the prefix commands will then be executed. The prefix commands are handled in sequential order, moving from the top of the file to the bottom of the file.

Invalid prefix commands do not cause an error message. Instead, they are redisplayed in the prefix area, preceded by a question mark ("?"). You can fix the invalid command by retyping it. You can also use the RESET PREFIX command, entered from the command line, which removes any commands from the prefix area.

As an example, assume that you enter the following prefix commands:



After pressing the F12 key (or, with INTERFACE CLASSIC, the Home key), you would see this:



What has happened is:

The A command caused a blank line to be added.

The Z command, which is invalid, was redisplayed with a question mark.

The D command caused the specified line to be deleted.

The line to be moved (M) was moved following (F) the specified line.

The " command caused a line to be duplicated.

Many prefix commands can be preceded or followed by some number *n*. For example,

```
3A
```

and

```
A3
```

are equivalent; both cause three lines to be added to your file. The prefix commands that allow this are:

n*A** or **A*n	Add *n* lines.
n*C** or **C*n	Copy *n* lines.
n*D** or **D*n	Delete *n* lines.
n*I** or **I*n	Insert *n* lines.
n*L** or **L*n	Lowercase *n* lines.
n*S** or **S*n	Show first *n* excluded lines represented by a shadow line.
S-*n*	Show last *n* lines represented by a shadow line.
n*U** or **U*n	Uppercase *n* lines.
n*X** or **X*n	eXclude *n* lines.
n*"** or **"*n	Duplicate a line *n* times.
n*<** or **<*n	Shift a line *n* columns to the left.
n*>** or **>*n	Shift a line *n* columns to the right.

Some prefix commands provide another way for you to indicate that they are to affect a group of lines. You do this by placing the name of the prefix command twice (for example, to copy a group of contiguous lines, use “CC”) in the prefix areas of the first and last lines involved. If, for example, you want to copy 15 lines, you could either put “15C” in the prefix area of the first line to be copied, or you could put “CC” in the prefix area of the first and the last lines to be copied, and avoid having to count the lines. The prefix commands that work this way are:

CC	Placed in the prefix area of the first and last lines to be Copied.
DD	Placed in the prefix area of the first and last lines to be Deleted.
LL	Placed in the prefix area of the first and last lines to be Lowercased.
MM	Placed in the prefix area of the first and last lines to be Moved.
UU	Placed in the prefix area of the first and last lines to be Uppercased.
XX	Placed in the prefix area of the first and last lines to be eXcluded.
<<	Placed in the prefix area of the first and last lines to be shifted left one column.

>>	Placed in the prefix area of the first and last lines to be shifted right one column.
""	Placed in the prefix area of the first and last lines to be duplicated.
n*<< or <<*n	Indicates that a group of lines is to be shifted *n* columns to the left.
n*>> or >>*n	Indicates that a group of lines is to be shifted *n* columns to the right.
n*""*or* ""*n	Indicates that a group of lines is to be duplicated *n* times.

For some actions, two or more prefix commands are required. For example, if you enter "CC" in the prefix area of a line, you must also enter "CC" in the prefix area of another line to fully indicate the group of lines to be copied, and either an "F" or "P" to indicate where the copied lines will be placed. If you press the F12 key (or, with INTERFACE CLASSIC, the Home key) to execute the commands in the prefix area and some actions cannot be carried out because some of the required pieces are missing (for example, a "CC" without a matching "CC"), KEDIT executes all of the prefix commands that it can. Prefix commands that can't be completed are left in the prefix area, with a message on the status line (temporarily replacing the alteration count) reminding you that prefix commands are still pending.

It is not necessarily an error, and is in fact often useful, to leave prefix commands pending. You might, for example, use MM and MM to indicate a group of lines to be moved, then move the cursor to the command line and issue some LOCATE commands to get to the location that the group of lines will be moved to.

7.2 Prefix Area Keyboard Considerations

Several keys act differently when PREFIX ON (or PREFIX NULLS) is in effect rather than PREFIX OFF. The keys act differently to allow you to conveniently move the cursor into and out of the prefix area and execute prefix commands. Keys involved with the prefix area include:

Cursor Right ( ) With PREFIX OFF, the Cursor Right key moves the cursor one character to the right, possibly causing horizontal scrolling if you attempt to move beyond the right edge of the window. It never moves the cursor to a different line.

With PREFIX ON, the Cursor Right key moves the cursor one character to the right. The cursor might cross the boundary between the file area and the prefix area and might wrap to the next line.

Cursor Left ( ) With PREFIX OFF, the Cursor Left key moves the cursor one character to the left, possibly causing automatic horizontal scrolling but never moving the cursor to a different line.

With PREFIX ON, Cursor Left moves the cursor one character to the left, possibly crossing the boundary between the file area and the prefix area and possibly wrapping to the previous line, without causing any horizontal scrolling.

Tab ( ) With PREFIX OFF, the Tab key moves the cursor to the next tab position.

With PREFIX ON, the Tab key instead moves the cursor to the beginning of the next *field*: if the cursor is in the prefix area, it moves to the beginning of the line of text; if the cursor is in the file area, it moves forward to the beginning of the next prefix area, without causing any horizontal scrolling.

You can still move to the next tab position with PREFIX ON. To do so, use function key F4.

Shift+Tab ( ) With PREFIX OFF, Shift+Tab moves the cursor backwards one tab position.

With PREFIX ON, Shift+Tab moves the cursor backwards one *field*, where each line of the file or the prefix area is considered one field.

F12 or **Home** With PREFIX OFF the F12 key (or, with INTERFACE CLASSIC, the Home key) simply moves the cursor to the command line.

With PREFIX ON, the cursor moves to the command line and then any pending prefix commands are executed. After the prefix commands have been executed, the cursor might be repositioned to the file area. For example, after executing the prefix command "A" to add a line, the cursor is positioned at the beginning of the newly added line.

Shift+Ctrl+Enter or **Ctrl+Numeric Pad Enter** Use these key combinations to toggle the cursor between a line's prefix area and file area. Additionally, any pending prefix commands are executed. With INTERFACE CLASSIC two additional keys, Ctrl+Enter and the "+" key on the numeric pad, perform the same function.

With PREFIX ON, several keys (Cursor Left, Cursor Right, Tab, and Shift+Tab) act quite a bit like their equivalents on an IBM 3270 terminal, the terminal most often used with XEDIT. However, you lose some of the benefits of KEDIT's normal definitions for these keys, which let the cursor keys cause automatic horizontal scrolling, and have the Tab and Shift+Tab keys move to tab columns within your data. If you don't like the way any KEDIT key is defined, you can change its definition. This is fully discussed in Chapter 10, "Using Macros". The following definitions, added to your profile, would change the Cursor Left, Cursor Right, Tab, and Shift+Tab keys to bypass their prefix-related actions and allow tabbing and horizontal scrolling.

```
"define curl 'cursor left'"
"define curr 'cursor right'"
"define tab 'sos tab'"
"define s+tab 'sos tabb'"
```

7.3 Prefix Command Equivalents

By default, the prefix area is turned off and other KEDIT facilities handle the functions of the prefix area. Here is a list of common sequences of prefix commands with their PREFIX OFF equivalents:

Prefix Commands	KEDIT Equivalent	Explanation of KEDIT Keys
c **f**	Alt+L Alt+C	Mark a block of 1 line with Alt+L. Then copy the line with Alt+C.
cc **cc** **f**	Alt+L Alt+L Alt+C	Mark the block to be copied by pressing Alt+L when the cursor is on the first and then the last line of the block. Copy the block under the line the cursor is on by pressing Alt+C.
mm **mm** **f**	Alt+L Alt+L Alt+M	Mark the block to be moved by pressing Alt+L when the cursor is on the first and then the last line of the block. Move the block under the line the cursor is on by pressing Alt+M.
"	F8	Duplicate the cursor line.
"" ""	Alt+L Alt+L Alt+C	Mark the block to be duplicated with Alt+L. Press Alt+C to duplicate the block.
n"" ""	Alt+L Alt+L DUP n BLOCK	Mark the block to be duplicated with Alt+L. Issue the command "DUP *n* BLOCK" to duplicate the block *n* times.
/	F5	Make the cursor line become the current line.
a	F2	Add a blank line below the cursor line.
***n*a**	ADD *n*	Use the command "ADD *n*"
d	Alt+D	Delete the cursor line.
dd **dd**	Alt+L Alt+L Alt+G	Mark block to be deleted with Alt+L. Press Alt+G to delete the block.
x		No direct equivalent
s		No direct equivalent

Chapter 8. Selective Line Editing and Highlighting

There is often a subset of your file that is of special interest to you. This subset could be made up of lines that are not right next to each other, but instead are in different parts of your file. For example, suppose you're writing a program that uses a variable named *count*. If you run your program and find that this variable is not being set correctly, you might want to concentrate on the lines in your file that use the variable *count*.

KEDIT provides two facilities that help you focus on a subset of your file. The first facility is the selective line editing facility, most commonly used with the ALL command and with the Edit Selective Editing dialog box. The selective line editing facility lets you pick out a subset of your file to view and work with on your screen, excluding the rest of your file from the display. KEDIT's selective line editing facility is discussed in the following section.

The second facility is the highlighting facility, which shows all of the lines of your file on the screen, but highlights the subset of lines that are of interest to you. You most often access the highlighting facility by using the SET HIGHLIGHTING command and the TAG command. KEDIT's highlighting facility is discussed in section Section 8.3, "Highlighting Facility".

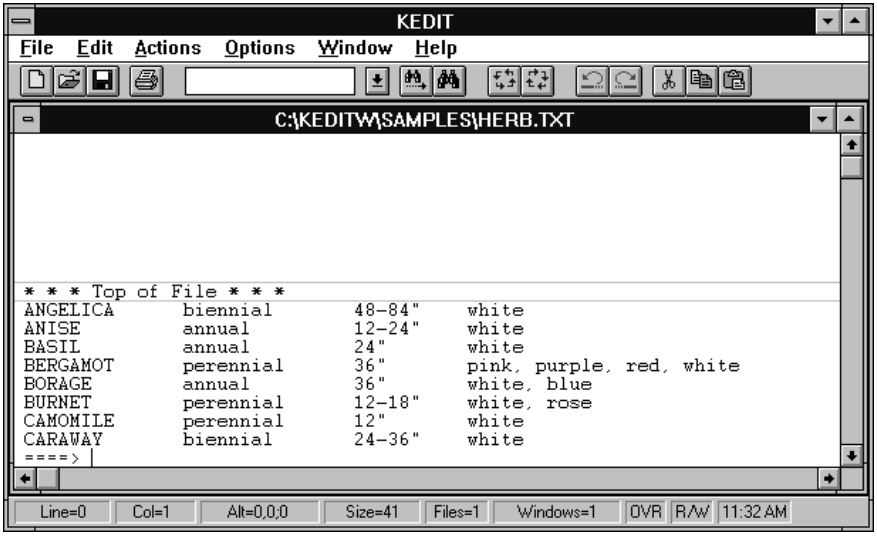
8.1 Selective Line Editing

8.1.1 General Discussion

To see how selective line editing works, let's go through some examples. The file we'll be using in these examples is called HERB.TXT. This file, which contains a list of forty-one garden herbs, is normally installed in your \KEDITW\SAMPLES directory by the KEDIT for Windows Setup program so that you can work through the examples if you'd like. If you do work through these examples on your PC, don't save any changes that you make to HERB.TXT. That way, the sample file will stay unchanged, so you can work through the examples again later on if you want to.

Suppose you have an ornamental herb garden. There are several things you'd want to know about each herb in your garden. You'd want to keep track of the plant type of each herb—whether it is an annual (which must be replanted each year), a biennial (which flowers in its second year), or a perennial (which stays alive for many years). You might also want to know the height of each herb, and perhaps the color of each one's flowers. Each entry in HERB.TXT contains this kind of information: the common name of the herb, its plant type (annual, biennial or perennial), the herb's height in inches, and the color of its flowers. The entries are arranged in alphabetical order.

When you first edit HERB.TXT, it looks like this:

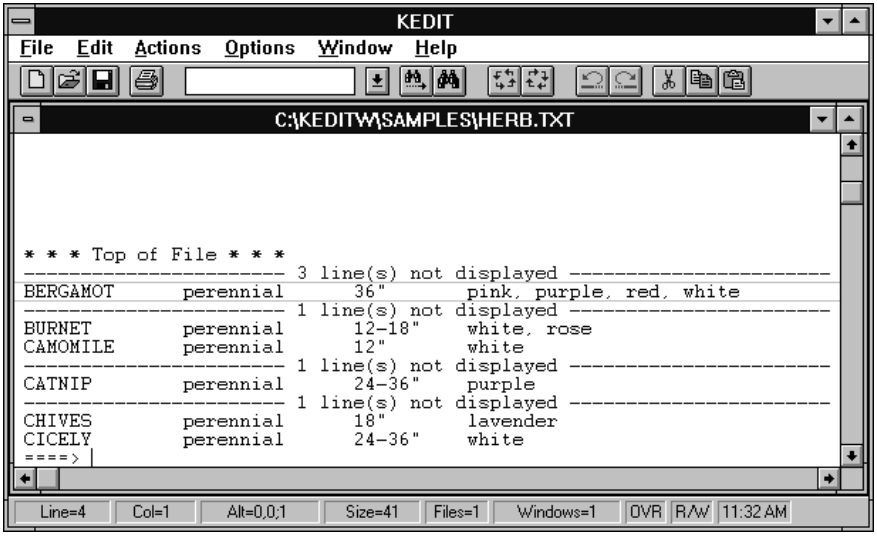


Screen 8.1: Sample file HERB.TXT

Suppose you want to look at only the herbs that are perennials. To do this you issue the command

```
all /perennial/
```

KEDIT then displays on the screen only the lines that contain the string ‘‘perennial’’, with the first line that is displayed becoming the new current line:



Screen 8.2: After issuing ‘all /perennial/’

Notice that in addition to displaying the lines containing the string ‘‘perennial’’, the screen displays lines that indicate where lines of the file are being excluded; these indicator lines are called *shadow lines*.

To toggle between viewing the subset of the file that you specified with the ALL command and viewing all lines of the file, use the Alt+Plus (that is, press Alt and the "+" key on the numeric pad). If you are viewing the subset of your file and you press Alt+Plus, KEDIT will display the entire file. If you then press Alt+Plus your display will return to the file's subset.

To completely reset your display and return to displaying all the lines of the file, just use the ALL command with no operands:

```
all
```

Edit Menu

Alternately, you can also use the Edit Selective Editing dialog box, type "perennial", and click on the Matching Lines button to view a subset of your file. You can then use the Edit Selective Editing dialog box's All Lines button to view the entire file again.

Toolbar

Once you have selected a subset of your file to work with (via command line or menu), you can use the Hide Excluded Lines and Show All Lines buttons on the default bottom toolbar to toggle your display between the full file and the subset you have selected.

8.1.2 The MORE and LESS Commands

KEDIT also provides commands called MORE and LESS to help you refine the selection of lines you made using the ALL command.

After you use the ALL command or the Edit Selective Editing dialog box to select a subset of lines in your file you can use the MORE command to add lines to that subset. For example, if in the above discussion you used

```
all /perennial/
```

you might then decide that you actually want to work with lines describing herbs that are either perennial or have white flowers. You could then use

```
more /white/
```

to add all the herbs with white flowers to the display. Similarly, the LESS command can be used to remove lines from the selected display. For example, if you used

```
all /perennial/
```

and then decided that you were not interested in perennial herbs with yellow flowers, you could use

```
less /yellow/
```

to remove perennials with yellow flowers from the display.

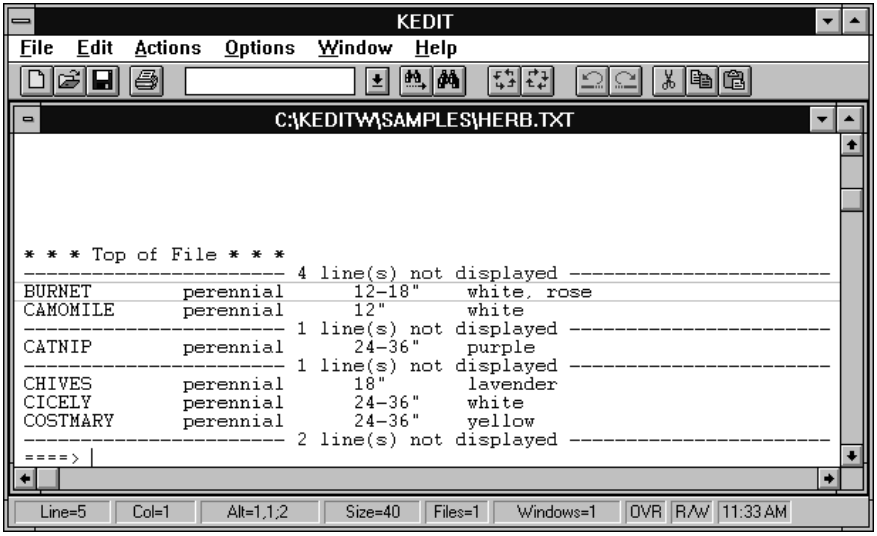
8.1.3 Editing Files that have Excluded Lines

When only a subset of the lines of your file are displayed, you can still edit the text just as you normally would. You can move the cursor in the file area, type over displayed text, etc. You can also perform block operations on your file.

Suppose again that you are interested in only the lines in HERB.TXT that contain the string "perennial". Use the command "ALL /perennial/" so that your screen looks like Screen 8.2. If you then issue the command

```
delete
```

then the current line, which is the entry containing the string "BERGAMOT", is deleted from your file. What is then displayed on the screen is this:



Screen 8.3: Afer issuing 'delete'

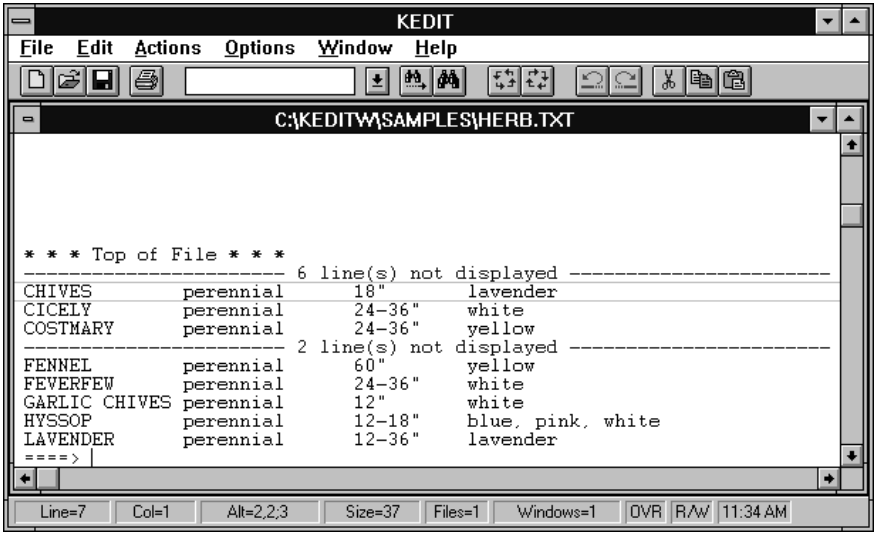
Notice that the line containing BERGAMOT is gone and that the pair of shadow lines on either side of it have become a single shadow line.

Note that a question comes up about what happens when you issue a command like DELETE 3—Which three lines are deleted? Does the DELETE command affect only the lines that are displayed, or are excluded lines affected as well? KEDIT gives you control over this.

Normally, after you issue the ALL command a SET option called SCOPE is automatically set to DISPLAY. This means that only lines that are displayed are operated on by most KEDIT commands. (The most important exceptions to this are File Save and related menu items and commands like FILE and SAVE. KEDIT will always save your entire file, including excluded lines.) So, in this case, the command

```
delete 3
```

will delete the current line and the next two displayed lines. For example, if your screen looks like Screen 8.3 and you issue the command DELETE 3, the resulting display will look like this:



Screen 8.4: After issuing 'delete 3'

Here, three displayed lines have been deleted: the line containing BURNET, the line containing CAMOMILE, and the line containing CATNIP. The line containing CHIVES becomes the new current line.

Notice that the shadow line above the current line indicates that there are now six excluded lines at the beginning of the file. These excluded lines include the four that had been above BURNET, the one above CATNIP, and the one above CHIVES. Since the displayed lines that separated these excluded lines have been deleted, all of these excluded lines are now adjacent to each other and so their position is indicated by a single shadow line.

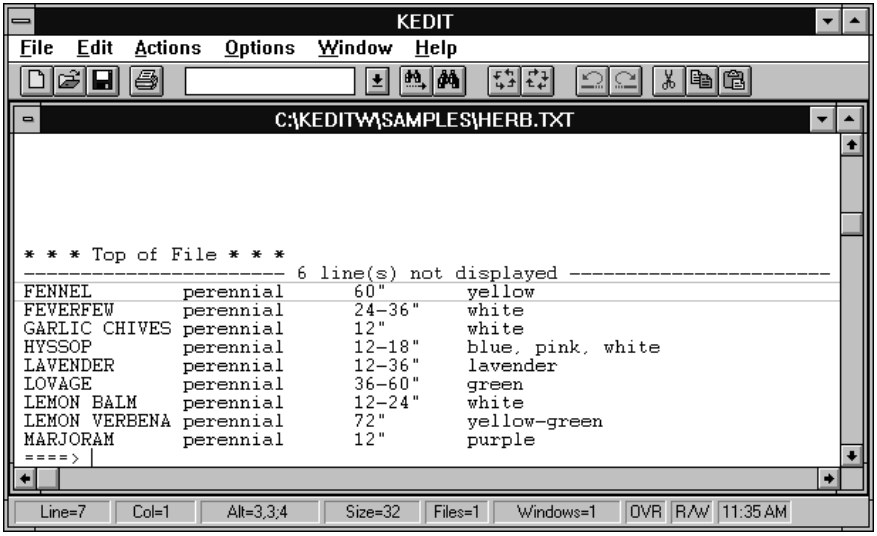
Now suppose that you want KEDIT commands to operate on all lines of your file, not just the lines that are displayed. In this case, after you issue the ALL command you will need to issue the command

```
set scope all
```

Entering the KEDIT command

```
delete 5
```

will then delete the current line and the next four lines below it, whether or not those lines are displayed. For example, if you have SCOPE ALL in effect and start out with the screen in Screen 8.4, after you issue the command DELETE 5 the screen will look like this:



Screen 8.5: After issuing 'scope all' then 'delete 5'

Here, the lines containing CHIVES, CICELY and COSTMARY have been deleted, along with the two excluded lines indicated by the shadow line below COSTMARY. The line containing FENNEL becomes the new current line.

Notes

You need to be careful when SCOPE ALL is in effect, since lines that are not displayed can inadvertently be changed or deleted—perhaps without your realizing it. For this reason SCOPE DISPLAY is the default setting, and is also automatically put into effect whenever you use the ALL command to select a subset of your file.

When SCOPE ALL is in effect and a (previously) excluded line becomes the new current line, that line will be displayed. If you then move the current line pointer, the line that was forced into the display will again be excluded from the display.

Examples

For the rest of this section we'll assume that you have not set SCOPE to ALL, but rather have the default, SCOPE DISPLAY. If you are working through these examples on your PC, you should now issue the command

```
set scope display
```

to reset SCOPE to its default setting.

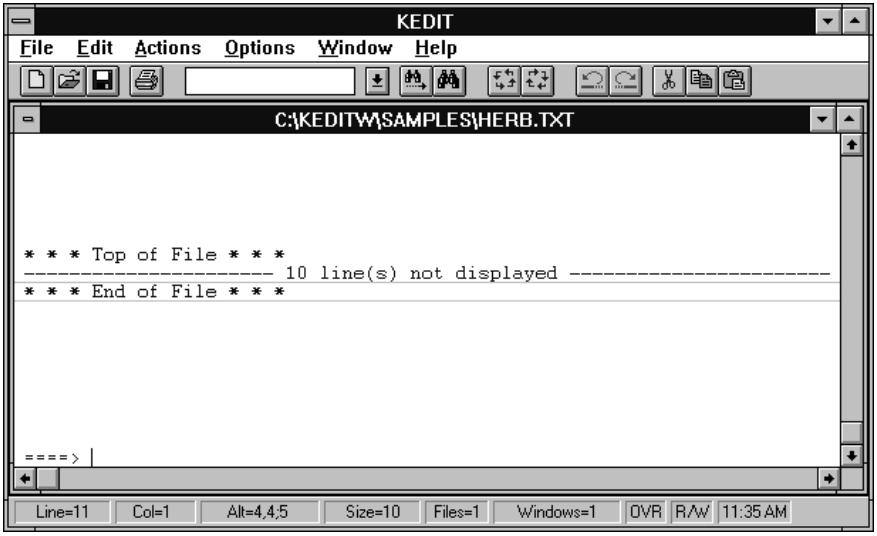
Let's look at a couple of situations where using the ALL command would be useful.

Suppose you now want to delete all of the entries for perennials in HERB.TXT. (Remember that only lines with "perennial" in them are currently displayed.) You can then issue the command

```
delete *
```

to delete all the displayed lines, from the current line down.

Notice that in our example the screen then looks like this:



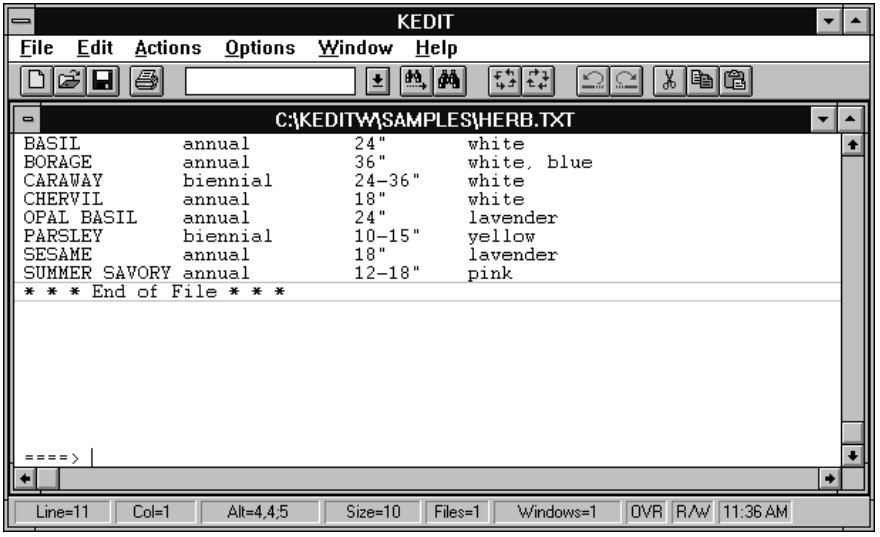
Screen 8.6: After issuing ‘delete *’ (with ‘scope display’)

The screen now indicates that the whole file is excluded, because the lines containing “perennial” have all been deleted and the rest of the lines are excluded from the display.

To redisplay the excluded lines, you issue the ALL command with no operands:

```
all
```

The screen will then look something like this:

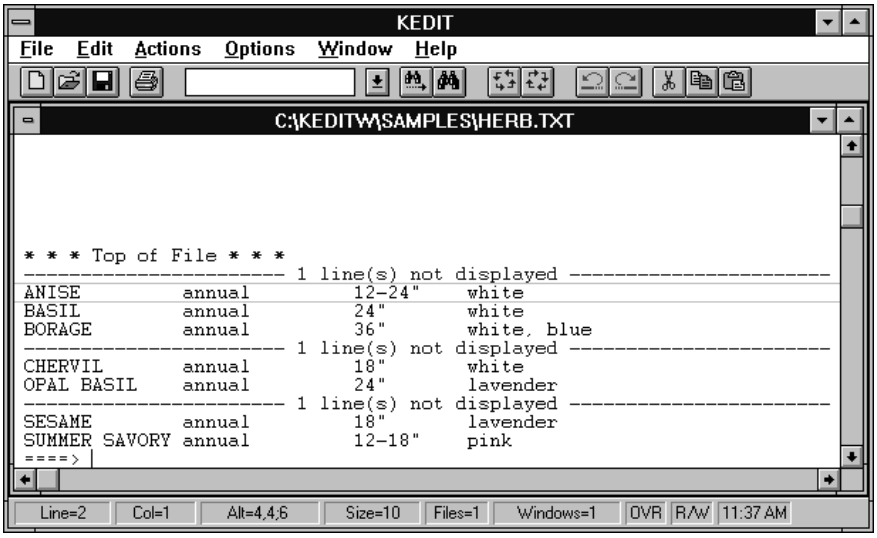


Screen 8.7: After issuing ‘all’ with no operands

As another example, suppose that you want to make a file containing all the entries for annuals still contained in HERB.TXT. You can have KEDIT display only the entries for annuals by issuing the command

```
all /annual/
```

If you start out with the screen shown in Screen 8.7, the resulting screen will look like this:

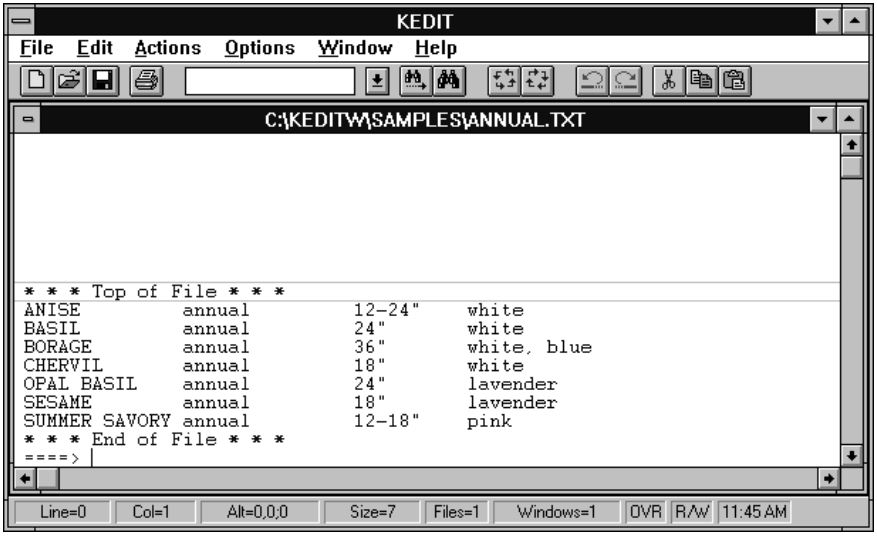


Screen 8.8: After issuing 'all /annual/'

Then you can issue the command

```
put * annual.txt
```

to PUT all the displayed lines, from the current line down to the last line of the file, in the file ANNUAL.TXT. If you edit this new file, it will look like this:



Screen 8.9: ANNUAL.TXT

The new file contains only the lines that were displayed in HERB.TXT.

If you are working through the examples on your PC, exit from KEDIT now without saving ANNUAL.TXT and without saving the version of HERB.TXT that the above examples have modified.

8.1.4 SET SHADOW

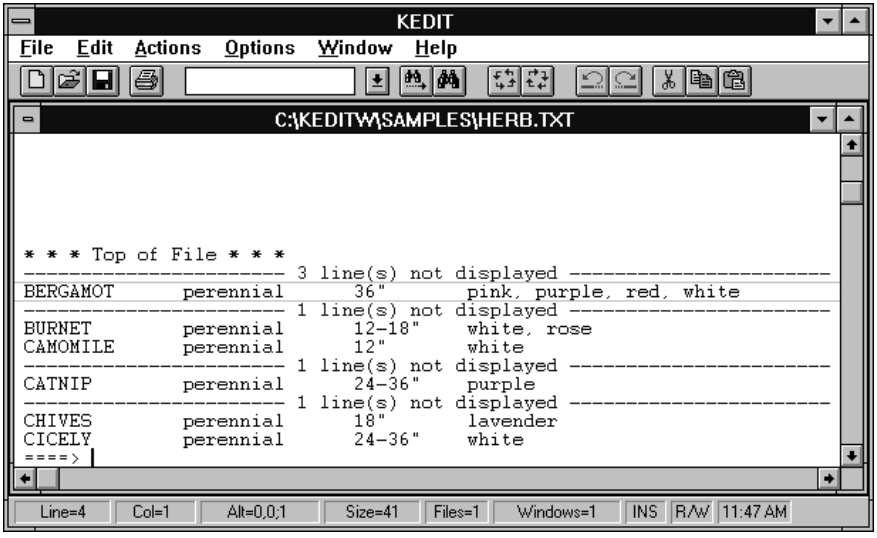
For many purposes, it's useful to see shadow lines to know how many lines are excluded from the display. For other purposes, you'd like to avoid the shadow lines and pretend that the excluded lines are not there. You can control whether or not shadow lines are displayed with SET SHADOW. The default is SHADOW ON.

If you are working through the examples on your PC, first edit the file HERB.TXT and then issue the command

```
all /perennial/
```

to display only the lines in the file that contain the string

"perennial". The screen then looks like this:

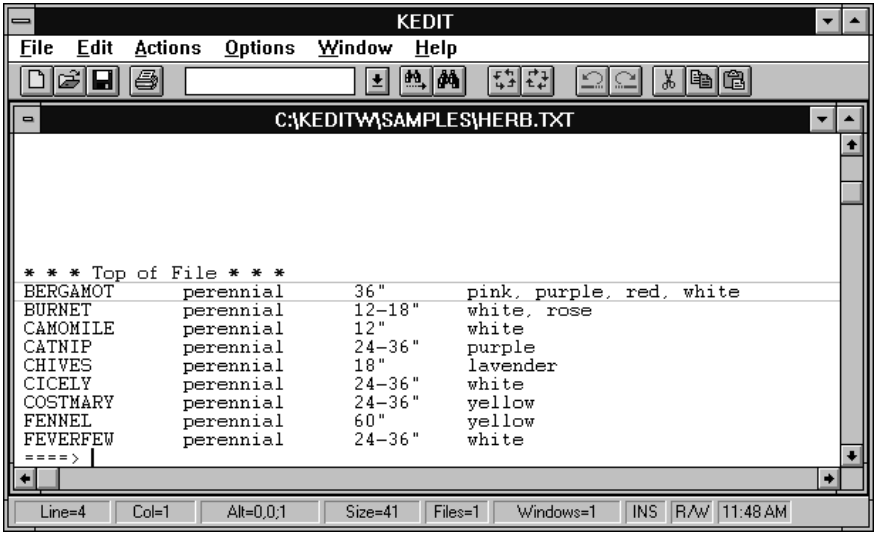


Screen 8.10: SHADOW ON is in effect

If you don't want to see the shadow lines, you can SET SHADOW OFF. Issuing the command

```
set shadow off
```

results in a display like this:



Screen 8.11: After issuing 'shadow off'

Note that the setting for SHADOW does not affect your ability to edit the displayed text. Also, SET SHADOW does not affect whether or not excluded lines are affected by KEDIT commands; this is controlled by SET SCOPE.

Excluded lines and the prefix area

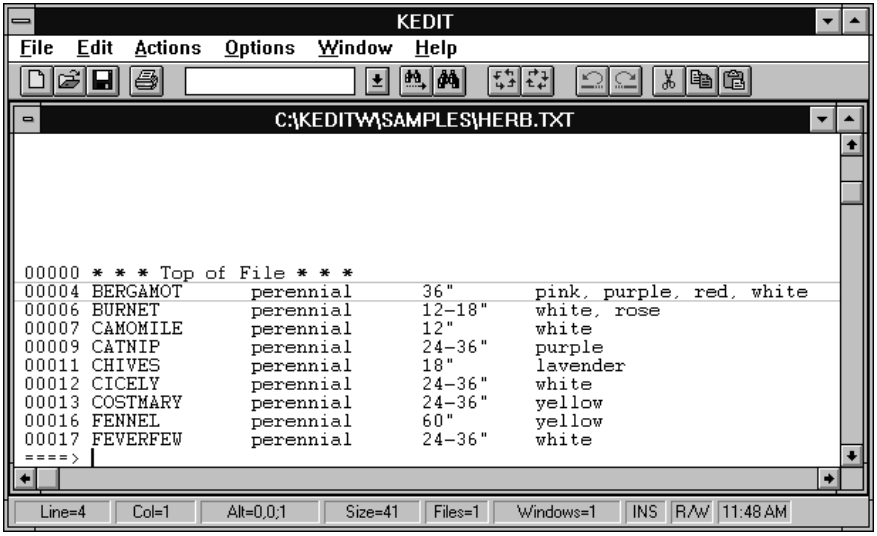
To keep track of excluded lines without displaying shadow lines, you can use the SET PREFIX and SET NUMBER commands. Issuing the commands

```
set prefix on
```

and

```
set number on
```

gives you a display like this:



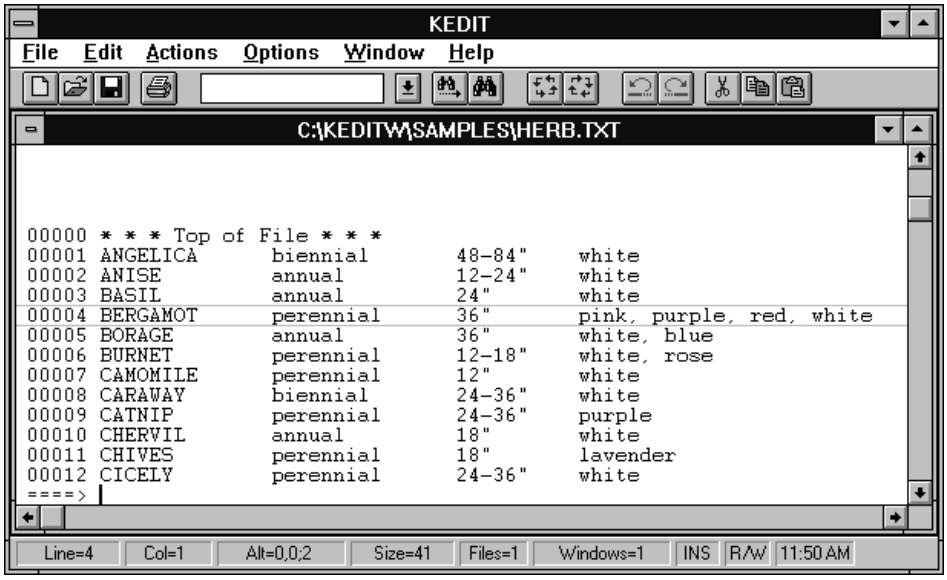
Screen 8.12: With SHADOW OFF, PREFIX ON, and NUMBER ON

Notice that the line numbers in the prefix area take into account excluded lines. So, for example, the prefix area shows that the line containing BERGAMOT is line 4. Even though this line is the first displayed line, it is still the fourth line of the file.

To redisplay all the lines of the file, issue the ALL command with no operands:

```
all
```

Your screen then looks like this:



Screen 8.13: HERB.TXT with PREFIX ON and NUMBER ON

If you are working through the examples on your PC, you should now enter the command

```
set shadow on
```

before continuing on to the next section, since the discussion there assumes that shadow lines are being displayed.

8.1.5 Prefix Commands Related to ALL

Related to ALL are KEDIT prefix commands that allow you to exclude specific lines from the display or to bring back to the display specific lines that have been previously excluded. Excluding and redisplaying specific lines is most commonly done using the prefix area, so the next few examples will show screens where PREFIX ON is in effect. To make clear where lines are excluded, these examples will also show screens with NUMBER ON in effect.

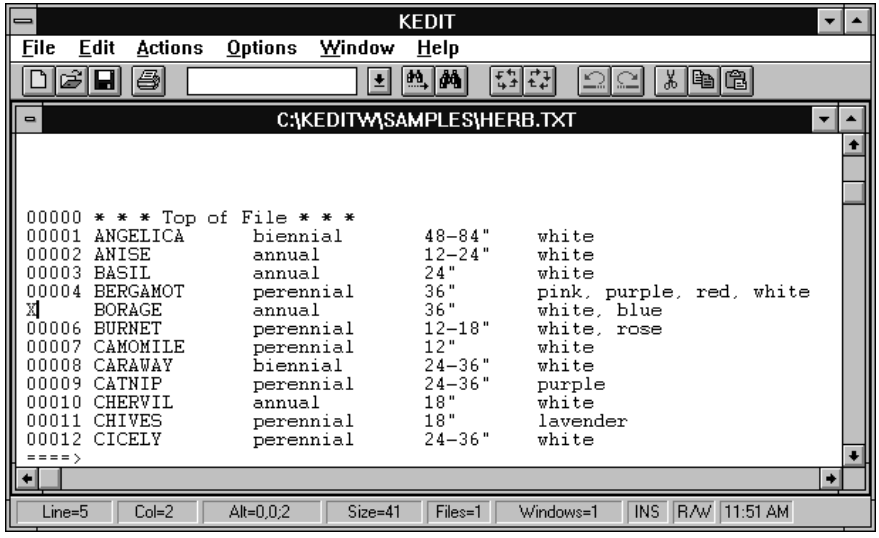
The prefix area is provided primarily for compatibility with XEDIT, the editor used on IBM mainframes running VM/CMS. If you are not familiar with XEDIT, then the X and S prefix commands and the line number display are probably the main reasons why you would use the prefix area. (See Chapter 7, “The Prefix Area”, for more information about the prefix area.)

If you are working through the examples on your PC and are not simply continuing from the last section, then edit the file HERB.TXT. Next, issue the commands

```
set prefix on
set number on
/BERGAMOT/
```

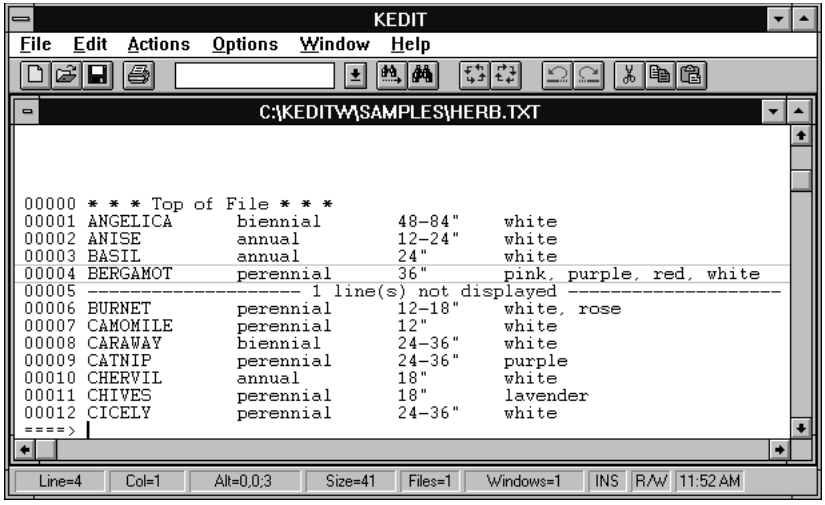
so that your screen will look like Screen 8.13.

The prefix command X can be used to exclude one line from the display (where X stands for "eXclude"). To exclude a line, enter an X in the prefix area next to the line you want to exclude and then execute this prefix command by pressing the F12 key (if you are using KEDIT's CUA interface) or the Home key (if you are using INTERFACE CLASSIC). For example, to exclude the line containing BORAGE, you put the prefix command X in the prefix area of line 5, as in the next screen, and then press F12 (for INTERFACE CUA) or Home (for INTERFACE CLASSIC):



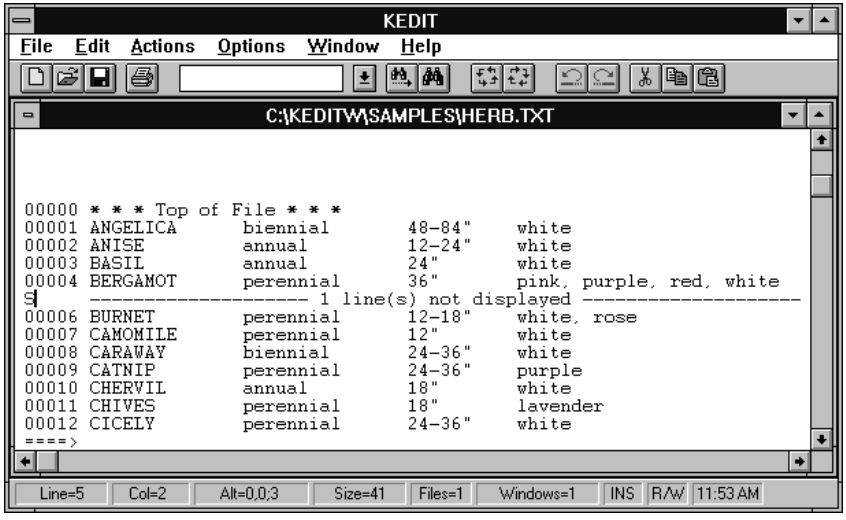
Screen 8.14: X prefix command

After you execute the X prefix command, the screen looks like this:



Screen 8.15: HERB.DOC after issuing the X prefix command

To bring an excluded line back into the display, you can use the prefix command S (where S stands for "Show"). Placing S in the prefix area of a shadow line and pressing the F12 (for INTERFACE CUA) or Home (for INTERFACE CLASSIC) will bring back to the display all the excluded lines corresponding to that shadow line. For example, to redisplay the line containing BORAGE, you type S in the prefix area of line 5, as in the next screen:



Screen 8.16: S prefix command

After you execute the S prefix command, the screen looks like Screen 8.13 again.

As with other prefix commands, you can enter *n*X or X*n* to exclude *n* lines, and *n*S or S*n* to show *n* lines. You can also enter XX in the prefix area of one line and XX in the prefix area of another line to exclude a block of lines.

For other variations of the X and S prefix commands, see Chapter 7, "The Prefix Area".

8.2 Selective Line Editing Details

This section gives more information on how KEDIT handles the ALL command and the X and S prefix commands. Most users, especially when first starting with these facilities, don't need to be concerned with these details.

8.2.1 Selection Levels

Associated with each line of a file is a number called the *selection level* of the line. In KEDIT, a selection level can be from 0 to 255. Initially, the selection level of all lines in the file is 0.

Selection levels, in conjunction with the SET DISPLAY command, are used to control what lines of your file KEDIT displays. The command

```
set display n1 n2
```

causes KEDIT to display only lines whose selection level is greater than or equal to *n1* and less than or equal to *n2*. Lines whose selection level are outside this range are not displayed by KEDIT. Instead, where there are lines excluded from the display, a shadow line is displayed if SHADOW is ON, or no lines are displayed if SHADOW is OFF.

To control the selection level of a line or group of lines, you can use the command SET SELECT. For example,

```
set select 5
```

sets the selection level of the current line to 5. Issuing the command

```
set select 2 *
```

sets the selection level of all lines to 2, from the current line down to the last line of the file.

Suppose you have a file with these five lines in it:

```
First line
Second line
Third line
Fourth line
Fifth line
```

and that the selection level of the first line is 10, the second line 20, the third line 30, the fourth line 40, and the fifth line 50. With DISPLAY 0 0, the default, none of the lines would be selected for display and the screen would show the following shadow line

```
——- 5 line(s) not displayed ——
```

If you then entered the command

```
display 30 30
```

the screen would display the third line, which has a selection level of 30.

Then, issuing the command

```
display 20 50
```

would result in the display of the second, third, fourth and fifth lines, whose selection levels range between 20 and 50. And you could enter

```
display 10 50
```

to display all the lines of the file.

8.2.2 How ALL Works

Here is an outline of how the ALL command works:

First, ALL puts SCOPE ALL into effect and sets the selection level of all the lines in the file to 0. (The selection level of the lines in the file would normally already be 0, unless previous ALL commands or X prefix commands had been issued or you had issued SET SELECT yourself.)

Then, for all lines that match the target specified with the ALL command, the selection level is set to 1.

Next, ALL sets DISPLAY to DISPLAY 1 1 so that only the lines that match the target will be displayed.

Regardless of the previous setting of SCOPE, the ALL command also resets SCOPE to SCOPE DISPLAY.

You reset the selection level of all lines in your file by using ALL with no operands. ALL with no operands sets the selection level of all lines in the file to 0, and sets DISPLAY to DISPLAY 0 0.

In KEDIT, ALL is implemented as a command to maximize performance, but ALL could have been written as a macro. (In fact, ALL is implemented as a macro in XEDIT.)

8.2.3 How X and S Work

Internally, the X prefix command sets the selection level of all lines that are to be excluded to 1 greater than the highest DISPLAY setting. Suppose you have DISPLAY 20 30 in effect. If you then issue the X prefix command, it sets the selection level of the affected lines to 31.

The S prefix command sets the selection level of affected lines to the higher of the two current DISPLAY settings. For example, if you have DISPLAY 20 30 and want to "show" some currently excluded lines, the S prefix command will set the selection level of those lines to 30.

Under XEDIT, X and S are macros. These prefix commands are built into KEDIT for performance reasons. Also, KEDIT does not currently support prefix macros.

8.3 Highlighting Facility

KEDIT's highlighting facility lets you highlight portions of your file. You can, for example, highlight lines that contain a certain string, or you can highlight altered lines.

The TAG command

To highlight lines that contain particular strings you will most often access the highlight facility by using the TAG command. The TAG command is in many ways similar to the ALL command in that it lets you specify a subset of your file that is of interest to you.

For example, where the command

```
all /Fred/
```

would select only the lines containing the string "Fred" for display,

```
tag /Fred/
```

would instead highlight the lines containing the string "Fred", but leave all of the lines in your file displayed.

And, just as you can use ALL with no operands to reset the status of selective editing and resume displaying all the lines in your file, you can use TAG with no operands to reset (turn off) highlighting.

Similarly, the MORE and LESS commands work in conjunction with the TAG command as well as with the ALL command. So, after specifying

```
tag /perennial/
```

you could use

```
more tag /white/
```

to highlight the entries in HERB.TXT that describe either perennials or herbs with white flowers.

HIGHLIGHT ALTERED

The other primary use of the highlighting facility is highlighting lines that have been altered in your current editing session. To do this you would use the command HIGHLIGHT ALTERED. Any lines in your file that are new or changed would be highlighted.

As you make changes to additional lines of your file, these lines are also highlighted. If you use the undo facility to undo all changes to a line, the line will no longer be highlighted.

You can use the SET COLOR command to control the colors used to highlight lines of your file; SET COLOR HIGHLIGHT controls the color used for highlighted lines.

See also

The highlighting facility also lets you highlight lines based on other flag bits and on selection levels. For additional information about the highlighting facility, see the description of SET HIGHLIGHT in the Reference Manual. For additional information about flag bits, see the description of SET LINEFLAG in the Reference Manual.

Target highlighting

Note that the highlighting facility discussed here is different from KEDIT's target highlighting. The target highlighting facility, controlled by SET THIGHLIGHT, highlights string targets that you find using the LOCATE, CLOCATE, and TFIND commands, and with the Edit Find and Edit Replace dialog boxes.

Chapter 9. Tailoring KEDIT

KEDIT is a highly configurable text editor. There are over 100 SET options that you can use to control various aspects of KEDIT, and you can further adjust KEDIT's keyboard and mouse behavior by defining your own macros in place of or in addition to the macros that are built into KEDIT.

Sometimes you will want to make a temporary change to KEDIT's behavior, affecting your editing of a single file or affecting only the current editing session. At other times you will want to make changes to KEDIT's behavior that remain in effect during future editing sessions.

This chapter discusses how you can use the SET command or the Options SET Command dialog box to make temporary changes to the values of KEDIT's SET options, how you can make these changes permanent by using the Options Save Settings dialog box, and how you can use KEDIT's profile facility to make further adjustments to your SET options and to define your own macros.

9.1 SET Options

The SET command

The SET command, covered in detail in Reference Manual Chapter 4, "The SET Command", provides options that you can use to control many details of KEDIT's behavior. Most SET options can also be controlled through the Options SET Command dialog box. A few SET options are either too specialized or too complex to be handled through the dialog box and are available only via the SET command.

You can control such things as the colors KEDIT uses on your display, whether the command line is placed at the top or bottom of your document window, and whether KEDIT's autosave facility is enabled. For example,

```
set color cmdline blue
set cmdline top
set autosave 25
```

With the exception of the rarely-used SET ALT and SET = options, the word SET is not required. When KEDIT sees a command that it does not recognize, it will automatically check for a SET option and handle it properly. The examples in this chapter will generally spell out the word SET, but it is optional. The preceding commands could have been entered as:

```
color cmdline blue
cmdline top
autosave 25
```

Levels

The documentation for each SET option, and the information displayed for each option within the Options SET Command dialog box, indicate the ''level'' at which the option takes effect. Some SET options are at the Global level, affecting your entire KEDIT session. Some options are at the File level, affecting only the current file. Other options

are at the View level, and can be different for each view you have of a file that is displayed in multiple windows.

At the Global level are options like STATUSLINE, which determines whether KEDIT displays a line of status information at the bottom of its frame window, and MACROPATH, which controls which directories KEDIT searches when looking for a macro. Most options at the File level affect how a file is read from or written to disk, such as LRECL and TABSOUT. This is because if you have several files in the ring, you might want them all to be written to disk with different record lengths. But if the same file is displayed in multiple windows, it is unlikely that you would want it to be written to disk with different record lengths depending on which window the save operation was initiated from. The largest number of options are at the View level, since you might well want to have, for example, different VERIFY settings in different views of the same file.

Saving your settings

Issuing a SET command, via the command line or via the Options SET Command dialog box, only affects the current editing session. To make a more permanent change to the value of a SET option, you must specifically tell KEDIT to save the new value for use in future editing sessions.

KEDIT uses the Windows registry to keep track of information that is saved from one editing session to the next. This includes the saved settings discussed here, lists of recently-edited files and recently-processed command lines, and various other status information. At the start of each new editing session, KEDIT processes the information saved in the registry and puts your saved settings back into effect.

There are three ways to save SET option values in the Windows registry for use in future editing sessions:

From within the Options SET Command dialog box you can use the Save Setting button to save the value of an individual SET option. The value of the SET option that is currently displayed within Options SET Command will be saved, and the saved values of other SET options will not be affected.

You can use the Options Save Settings dialog box to process the current values of all savable SET options. Options Save Settings displays a list of the SET options whose current values differ from the values previously saved in the registry. You can review this list and then either save these values to the Windows registry or cancel the save operation.

From the KEDIT command line, you can use the REGUTIL command, which has operands that let you save individual settings in the Windows registry, save all settings in the registry, and reset all settings saved in the registry to their default values.

Notes

Some notes on saving your settings:

The values of most, but not all, SET options can be saved in the Windows registry; the SET command documentation and the Options SET Command dialog box indicate which options are savable and which are not.

The options that cannot be saved are, in general, those that you might set on a one-time basis, or that you might use with a particular type of file, but would be unlikely to want automatically in effect for every file that you edit. For example, you can use the SET ZONE command to tell KEDIT to restrict string searches to certain columns of a file, but it is unlikely that you would want to automatically restrict the search columns in every file that you edit, so SET ZONE is not saved in the Windows registry. Additionally, a few of the more complex options, like SET TOOLBUTTON and SET TOOLSET, cannot be saved in the registry.

When you use Options Save Settings, not all option values are actually written to the Windows registry; to speed things up, KEDIT only writes out the options whose values differ from the built-in KEDIT default.

There is one special class of SET options whose values are automatically updated in the Windows registry whenever they are set. These are SET INSTANCE, SET INITIALDIR, SET INITIALDOCSIZE, SET INITIALFRAMESIZE, SET INITIALINSERT, and SET INITIALWIDTH, and what they have in common is that they have an effect only during KEDIT initialization. They are automatically saved because there is no point in setting these options unless the changes are reflected in registry so that they can affect future KEDIT sessions. Setting these options has no effect on the current KEDIT session, because you don't get a chance to set them until KEDIT has already been initialized. But whenever you set one of these options, the new value is automatically saved in the registry, and it will affect future KEDIT sessions.

9.2 KEDIT Profiles

9.2.1 Overview of KEDIT Profiles

For many users, customizing KEDIT can simply involve using the Options SET Command dialog box to choose preferred settings, and then using the Options Save Settings dialog box to save these settings for use in future editing sessions. There are, however, some limitations to this approach:

The values of some KEDIT options, like the SET TOOLBUTTON and SET TOOLSET options used to customize the toolbar, cannot be handled by Options Save Settings.

Your saved settings are put into effect in all future editing sessions, regardless of the type of file you are editing. Some other mechanism is needed if you want to put different options into effect depending on the type of file you are editing.

Macro definitions, which are controlled by KEDIT's DEFINE command, are not handled by Options Save Settings.

To address these considerations, you can create a KEDIT *profile*.

What is a profile?

Your profile is a special macro file, normally called WINPROF.KEX, that KEDIT processes at the start of each editing session. Any KEDIT commands that you want to have executed at the start of each editing session, including SET commands for options not processed by Options Save Settings, and DEFINE commands for KEDIT macros, can be included in WINPROF.KEX. You can also use IF-THEN-ELSE logic within your profile to issue different commands depending on the type of file that you are editing.

You do not need to know all about KEDIT macros to set up a useful KEDIT profile, and this chapter covers only the bare minimum. Chapter 10, "Using Macros", has much more information about using KEDIT macros.

To create a profile, use KEDIT to create a file called WINPROF.KEX. As discussed in Section 10.2.3, "Storing Your Macros", we recommend that WINPROF.KEX be kept in the "KEDIT Macros" subdirectory of your Windows Documents folder (which is sometimes known as the My Documents folder). In WINPROF.KEX, put the KEDIT commands that you want to issue whenever you run KEDIT. For reasons discussed below, you should put quotes around the commands. An example:

```
'set synonym search 1 locate'
'set cmdline top'
'set autosave 25'
```

Whenever you start KEDIT, it looks for the file WINPROF.KEX. If WINPROF.KEX is found, KEDIT processes it, issuing all the commands that it contains.

Note that the SET SYNONYM command in the above example is not handled by Options Save Settings, and would therefore need to be put into your profile if you want it processed at the start of every KEDIT session. Options Save Settings does handle most options, including SET CMDLINE and SET AUTOSAVE, so these settings could be put into effect via your profile or via Options Save Settings. SET commands issued from your profile take precedence over saved settings loaded from the Windows registry. See Section 9.2.2, "Order of Processing", for more on this point.

Using quotes

Enclose commands issued from your profile in quotes. This causes the commands to be passed directly to KEDIT for execution. Commands that are not in quotes will often be executed properly, but are sometimes interpreted by KEDIT's macro processor as macro instructions, expressions, or variables, leading to unexpected and possibly confusing errors. To avoid these problems, always use quotes.

You can use either single or double quotes. Single quotes are most common:

```
'set wrap on'
'set arbchar on'
```

An exception comes when the command that you are issuing itself contains single quotes, as DEFINE commands often do. In this case, you would normally enclose the command in double quotes. So the command

```
define f5 'cursor down'
```

should appear in your profile as

```
"define f5 'cursor down'"
```

Comments

You can use comments to document the contents of your profile. Any line in your profile that begins with an asterisk (“*”) is a comment line and is ignored by KEDIT when it processes your profile. Note that comment lines should not be enclosed in quotes. Blank lines in your profile are also ignored.

```
* set options for string searches
'set wrap on'
'set arbchar on'
'set varblank on'

* make end key toggle between beginning and end of line
"def end if after() then 'sos firstchar'; else 'sos endchar'"
```

Making decisions

You can use IF instructions in your profile to test conditions and act differently based on the results. The most common use of this in profiles is to issue different SET commands based on the extension of the file you are editing. FEXT.1() is an implied EXTRACT function that returns the extension of the file you are editing. We will not discuss implied EXTRACT functions in detail here, but the usage of FEXT.1() should be clear from the following examples. Implied EXTRACT functions are discussed in Chapter 10, “Using Macros”, and fully documented in Reference Manual Chapter 5, “QUERY and EXTRACT”.

The next example sets WORDWRAP ON if you are editing a file with an extension of TXT. Note that the IF instruction is not enclosed in quotes. It is part of KEDIT’s macro language and is not a KEDIT command. However, constant strings used in macro expressions, such as the word TXT in this example, should be given in quotes.

```
if fext.1() = 'TXT' then
   'set wordwrap on'
```

The next example sets MARGINS to columns 1 and 75 if you are editing a file with an extension of TXT, and to columns 1 and 80 otherwise:

```
if fext.1() = 'TXT' then
   'set margins 1 75'
else
   'set margins 1 80'
```

The third example shows how you can use a DO—END pair to issue several commands when some condition is true:

```
if fext.1() = 'TXT' then do
   'set margins 1 75'
   'set wordwrap on'
   'set format justify'
   end
```

The DEFINE command

Your profile can contain any KEDIT commands, but the commands most commonly used in profiles are SET commands and DEFINE commands. SET commands, used to control the values of KEDIT SET options, have been discussed already.

Your profile can use the DEFINE command to setup other KEDIT macros, most often macros that are assigned to keys on your keyboard. Whenever you press a key on your keyboard, KEDIT executes the macro assigned to that key. So by using the DEFINE command, you can reconfigure your keyboard, defining the actions you want each key to control.

Short, one-line macros can be given directly in DEFINE commands included in your profile. For example, including the following line in your profile would cause function key F5 to move the cursor down one line:

```
"define f5 'cursor down'"
```

If you have a number of macros to define, and they are not simple one-line macros, you can put the macros into a file with an extension of KML (for ''KEDIT Macro Library'') and load all of the definitions at once with a single DEFINE command in your profile. For example,

```
'define mykeys.kml'
```

To actually write the macros handled by DEFINE commands, you'll need to know more about macros than is covered in this chapter. Macros are discussed in more detail in Chapter 10, "Using Macros".

SET REPROFILE

By default, your profile is executed only once, at the start of each KEDIT session. If your profile issues the command

```
'set reprofile on'
```

your profile will be executed at the start of each KEDIT session and, if you use KEDIT to edit multiple files, it will be re-executed whenever you begin to edit an additional file. You can then use IF instructions to make decisions based on information such as the file type of the file you are editing. The following sample profile will help show how this can be useful.

```
'set reprofile on'
if fext.1() = 'TXT' then
   'set wordwrap on'
```

If you start KEDIT by issuing, from the File Manager's File Run dialog box, the command

```
keditw program.c
```

this profile will turn REPROFILE ON, but will not set WORDWRAP ON for PROGRAM.C, since it does not have an extension of TXT. If, from the KEDIT command line, you start to edit a second file by issuing the command

```
kedit second.txt
```

your profile will be re-executed (since REPROFILE ON is in effect), and WORDWRAP ON will be put into effect for SECOND.TXT, since it has an extension of TXT.

If you had left the default of REPROFILE OFF in effect, KEDIT would not have re-executed your profile when you began to edit SECOND.TXT and WORDWRAP ON would not have been put into effect.

Note that when you are editing multiple files, as in this example, it is possible to have WORDWRAP ON in effect for one file while WORDWRAP OFF is in effect for another file. A number of SET options work this way, and can be set on a per-file basis. Many options can even be set on a per-view basis, where, if you are viewing the same file through two different KEDIT windows, SET options can have different values in each view. For more on this topic, see the discussion at the beginning of Reference Manual Chapter 4, "The SET Command".

INITIAL()

With REPROFILE ON, KEDIT will re-execute your profile whenever a new file is added to the ring. But some portions of your profile might only need to execute once, at the start of a KEDIT session. DEFINE commands are a common example of this.

Because macro definitions are global to KEDIT and, unlike most SET options, are not managed on a per-file or per-view basis, many KEDIT users have one set of macro definitions for their keys. These definitions can be stored in a .KML file and loaded only once, at the start of a KEDIT session, to avoid a time-consuming reload whenever a new file is added to the ring.

To make this possible, an IF instruction in your profile can test the Boolean function INITIAL(), which returns 1 (meaning True) if it is called from a profile executed at the start of a KEDIT session and otherwise returns 0 (meaning False). Here is a typical usage:

```
if initial() then
   'define mykeys.kml'
```

Loading your file

If your profile issues any command that depends on your file being loaded (such as a LOCATE or CHANGE command), KEDIT loads your file before processing the command. Otherwise, KEDIT loads your file after completion of your profile. You can force KEDIT to load your file at any point in your profile by issuing a command like LOCATE 0. (LOCATE 0 is useful because it does not change the current line location, which is usually set to the top-of-file line when a file is loaded but can be set elsewhere if the LINE initialization option is used.)

Most of what is typically done in profiles does *not* force the loading of your file. This includes use of the SET, EXTRACT, and DEFINE commands, and of KEXX Implied Extract, Boolean, and built-in functions. This is useful because it gives you a chance to issue commands like SET TABSIN from your profile before your file is loaded in, so that the command can have an effect on how the file gets loaded. (See Reference Manual Section 2.3 for a full list of commands that do not force the loading of your file from within the profile.)

9.2.2 Order of Processing

During KEDIT initialization, SET options are processed in the following order:

KEDIT's built-in default values for the SET options are put into effect.

KEDIT processes its section of the Windows registry, overriding the default SET option values with any values previously saved to the registry via the Save Settings dialog box.

KEDIT processes any initialization options specified via the KEDITW environment variable or the command line used to invoke KEDIT.

KEDIT processes your profile, which can make additional changes to the SET option values. Note that SET commands issued from your profile therefore override the effect of settings saved in the registry.

After all of this, you can make further changes by issuing the SET command from the KEDIT command line, or by using the Options SET Command dialog box, accessible through the Options menu.

Note in particular that since your profile is processed after your saved settings are processed, SET commands issued from your profile take precedence over saved settings. If you use Options Save Settings, but some of the saved settings do not take effect in future editing sessions, this is often the cause.

For example, assume that you have used the Options SET Command dialog box to put AUTOSAVE 40 into effect, and that you used the Save Setting button to save this value for use in future sessions. At the start of your next KEDIT session, AUTOSAVE 40 will be put into effect when KEDIT processes your saved settings. KEDIT will then execute your profile. If your profile contains the command SET AUTOSAVE 25, AUTOSAVE 25 will be put into effect, overriding the effect of the AUTOSAVE 40 from your saved settings.

For more information on KEDIT's processing of your saved settings and your profile, see Reference Manual Section 2.3.

9.2.3 Initialization Options

There are a number of special options, known as initialization options, that you can specify on the command line used to invoke KEDIT or to add additional files to the ring. (The KEDITW32 command is used to start KEDIT for Windows. Once a KEDIT session has begun, you can use the KEDIT command to add additional files to the ring.) These options let you specify that special processing is to take place for a particular KEDIT session, or when a particular file is added to the ring. For example, you can specify that an alternate profile, other than the normal WINPROF.KEX, is to be executed for a file that you are adding to the ring. To execute ALTPROF.KEX instead of WINPROF.KEX when starting KEDIT:

```
KEDITW32 SPECIAL.FIL (PROFILE ALTPROF
```

Initialization options follow the fileid on the command line. They are preceded by a left parenthesis, and are optionally followed by a right parenthesis.

For a full discussion of KEDIT's initialization options, see Reference Manual Chapter 2, "Invoking KEDIT".

9.2.4 A Sample Profile

Here is a very basic sample profile that you might use as a starting point for your own profile. This file is in KEDIT's SAMPLES subdirectory as PROFILEA.KEX. To use it, you would want to copy it to a file called WINPROF.KEX. We recommend that you keep WINPROF.KEX and any other macros that you create in the "KEDIT Macros" subdirectory of your Windows Documents folder (which is sometimes known as the My Documents folder).

PROFILEA.KEX

```
* a sample KEDIT profile

* turn on backup and autosave facilities
'set autosave 25'
'set backup keep'

* set some useful options
'set wrap on'
'set defsort date'
'set hexdisplay on'

* un-comment the next line to get an xedit-style prefix area
* 'set prefix on'

* make end key toggle between start and end of line
"def end if after() then 'sos firstchar';else 'sos endchar'"
```

Description of PROFILEA.KEX

Here is what this sample profile does:

```
* turn on backup and autosave facilities
'set autosave 25'
'set backup keep'
```

The profile turns on KEDIT's autosave facility, telling it to save your file after every 25 changes to the file. Your file is saved under its own name, with an extension of .AUS. This can help you recover your work in the event of a system crash or power failure. The autosave file is erased by KEDIT whenever the file is successfully saved via File Save or a related menu item, or via the FILE or SAVE commands.

The profile also turns on KEDIT's backup facility, which causes KEDIT to keep the previous copy of your file on disk (with an extension of .BAK) whenever a FILE or SAVE command is issued. If you make changes to a file with KEDIT and then have second thoughts about those changes, it is often useful to have access to the previous

version of the file. You can periodically erase any .BAK files that KEDIT leaves on your disk to free up the disk space involved.

Both SET AUTOSAVE and SET BACKUP are highly recommended for all KEDIT users who are not extremely short on disk space.

Note that another method for getting AUTOSAVE, BACKUP, and the other options set through this sample profile into effect at the start of each editing session would be to set the options through the Options SET Command dialog box and then save the settings with Options Save Settings. This would save the settings in KEDIT's section of the Windows registry, and you would not need to set them through your profile.

```
* set some useful options
'set wrap on'
'set defsort date'
'set hexdisplay on'
```

The next commands set options that most KEDIT users will find useful:

SET WRAP ON means that, instead of giving up when it reaches the bottom of your file, the LOCATE command will wrap around to the top and search your entire file for string targets.

SET DEFSORT DATE means that the DIR command will sort files by date, with the most recent files, which are the ones most likely to be of interest, listed first; the default is to sort files by name.

SET HEXDISPLAY ON tells KEDIT to display on the status line the decimal and hexadecimal values of the ASCII code for the character at the cursor position. This is useful to anyone who works with files containing special characters.

```
* un-comment the next line to get an xedit-style prefix area
* 'set prefix on'
```

If you are a user of IBM's mainframe XEDIT editor and you want to use XEDIT-style prefix commands, you will want to turn PREFIX ON in your profile. You should also look at VMPROF.KEX on the KEDIT distribution disk, which is a sample profile that turns on more of KEDIT's XEDIT-compatible features. See also Appendix A, "XEDIT Compatibility".

```
* make end key toggle between start and end of line
"def end if after() then 'sos firstchar';else 'sos endchar'"
```

This DEFINE command sets the END key to toggle the cursor between the beginning and end of a line. By default, the END always moves the cursor to the end of a line.

Other sample profiles

Two other sample profiles are on the KEDIT distribution disk:

PROFILEB.KEX is a bit more sophisticated than the sample described here. It uses IF instructions to tailor its behavior to the type of file being edited, and loads a set of macros from a .KML file.

VMPROF.KEX, mentioned above in the discussion of PREFIX ON, is a sample profile that makes KEDIT act more like IBM's XEDIT mainframe editor.

Chapter 10. Using Macros

You can use KEDIT macros to tailor KEDIT to your needs and automate repetitive editing tasks. A KEDIT *macro* is a program that controls a set of actions that you want KEDIT to perform.

KEDIT macros are written in a language called KEXX. KEXX is a subset of the REXX language, a widely-used macro language originally developed by IBM.

This chapter first discusses how to run KEDIT macros. Next it discusses the DEFINE command, which lets you load macros into memory and redefine the macros assigned to keys. Finally, it discusses the KEXX language itself, which you need to be familiar with if you will write your own macros.

Several other parts of the KEDIT for Windows documentation also discuss macros:

Chapter 9, "Tailoring KEDIT", covers profile macros, which let you adjust KEDIT's default option settings and key definitions.

Chapter 11, "Sample Macros", shows you how macros work in practice, discussing in detail several of the sample macros supplied with KEDIT.

Reference Manual Chapter 6, "Macro Reference", covers the KEXX language in more detail than does the overview given in this chapter. It also includes a summary of the differences between KEXX and REXX.

10.1 Running Macros

There are several ways to run a macro:

Pressing a key

Issuing the MACRO command

Running an implied macro

Using the mouse

Automatically executing your profile

Issuing the IMMEDIATE command

Issuing the DEBUG command

Pressing a key

Whenever you press a key in KEDIT, you are actually telling KEDIT to run a macro. KEDIT takes the name of the key that you press and runs a macro with that name. When you press function key F2, KEDIT runs a macro called F2. When you press the A key,

KEDIT runs a macro called A. When you press Ctrl+L, KEDIT runs a macro called Ctrl+L.

Default definitions for all keys are built into KEDIT. Many are very simple. For example, the macro assigned to the A key simply inserts the character ''a'' into your file, and the macro assigned to the PgDn (Page Down) key simply issues the command FORWARD. But you can use the DEFINE command, discussed in the next section, to assign your own macros to any KEDIT key.

MACRO command

The MACRO command is used to run a single in-memory macro (a built-in macro or a macro previously loaded by the DEFINE command) or a macro stored in a .KEX file.

For example,

```
macro test
```

runs a macro called TEST. KEDIT looks in memory for a macro called TEST and runs it. If no macro called TEST is in memory, KEDIT looks on disk for TEST.KEX.

Implied macros

If IMPMACRO ON is in effect, as it is by default, and you type in a command that KEDIT does not recognize, KEDIT automatically tries to execute a macro with that name. (This feature works only if the name of the macro is entirely alphabetic, with no numeric or special characters.) For example, with IMPMACRO ON, if you simply enter

```
test
```

KEDIT will look for the macro TEST, just as if you had entered

```
macro test
```

Using the mouse

Just as whenever you press a key you are actually running a KEDIT macro, whenever you click the mouse within a document window, press a toolbar button, or select an item from one of KEDIT's menus, you are also running a KEDIT macro, and these macros can be redefined. One way to do this is by adding your own buttons to the toolbar and defining the macros to be run when they are selected. Mouse macros are discussed in Reference Manual Chapter 7, "Built-in Macro Handling".

Profile macros

A special macro called the *profile* is automatically executed when you start up KEDIT. A profile is most often used to tailor KEDIT to your needs by changing KEDIT settings and by defining other macros useful in your daily work or associated with the type of file you are editing. Profiles are discussed in Chapter 9, "Tailoring KEDIT".

IMMEDIATE command

The IMMEDIATE command can be used to run a one-time-only macro from the command line, without assigning it to an in-memory macro. Once you are familiar with the KEXX language, you might find this useful. For example,

```
immediate do n = 1 to 4; 'input Part number' n; end
```

will insert these four lines into your file, below the current line:

```
Part number 1
Part number 2
Part number 3
Part number 4
```

IMMEDIATE is also useful for making a quick check to see whether a macro instruction exists or works the way you think it should.

DEBUG command

The DEBUG command is like the MACRO command, except that it runs your macro with KEDIT's macro debugging facility active. Debugging is discussed below in Section 10.4, "Debugging KEXX Macros".

10.2 Defining Macros

This section covers the DEFINE command, which is used to load macros into memory. It also gives examples of some very simple KEXX macros and discusses how they work.

10.2.1 One-line Macros

The simplest way to define a KEDIT macro and load it into memory is to enter the definition on the command line using the DEFINE command. This method is limited in that it only allows you to define macros that fit on the command line. Later we will look at defining longer macros.

define *macroname macrodefinition*

The macro name can be any name, but if it is the name of a key (or key combination) then the macro will be bound to that key. (For a list of the key names used by KEDIT, see Reference Manual Chapter 7, "Built-in Macro Handling".) When a macro is bound to a key, then pressing the key causes the macro to be run. For example, if you define a macro with the name F1 and later press the F1 key, the F1 macro will be run.

For this reason, you should avoid using one-character names for macros that you don't want to have assigned to keys. For example, if you define a macro named A, it will be executed whenever you try to use the keyboard to enter the letter ''a''.

The simplest macro would issue a single KEDIT command. For example,

```
define f1 'top'
```

This binds the macro to the F1 key. So in this case pressing the F1 key executes the KEDIT command TOP.

Using quotes

A KEDIT command issued from a macro should be specified as a literal string, and so should be enclosed in single quotes (') or double quotes ("). The command TOP is enclosed in quotes to distinguish it from a variable name. For example, a KEDIT command such as ''put * temp'' should be entered in a macro as

```
'put * temp'
```

to distinguish it from a macro expression where the value of a variable named *put* is multiplied by the value of a variable named *temp*. KEDIT commands issued from a macro can include variable information. If you include variables in the specification of a KEDIT command, any literal part of the command should be enclosed in quotes and any variable part of the command should appear outside of the quotes. An example is given and discussed in Section 10.3.5, "Commands".

Note that if a DEFINE command is issued from a macro (and DEFINE commands are often issued from KEDIT profile macros), the DEFINE command itself should be enclosed in quotes. Since the DEFINE command normally includes single quote characters, you would usually put it in double quotes in a macro. So the DEFINE command in the macro above would appear in a KEDIT profile as

```
"define f1 'top'"
```


Issuing multiple commands

A slightly more complicated macro issues a series of KEDIT commands. For example,

```
define f1 ':1';'delete';'bottom';'delete'
```

This macro deletes the first and last lines of a file. Pressing F1 in this case will first issue the KEDIT command :1, making the first line of the file the focus line. Then the KEDIT command DELETE will be issued, deleting that line. Next the command BOTTOM will be issued, making the last line of your file the focus line. Finally, the DELETE command will again be issued, deleting that line.

Notice that this macro contains four "clauses", each clause made up of a separate KEDIT command. You can put multiple clauses on one line if they are separated by semicolons (";").

So far we've looked at macros that issue KEDIT commands, but we haven't seen any macros that take advantage of the control structures provided by the macro facility. The next macro makes use of a loop with a control variable.

```
define f1 do i = 1 to 5;'input' i i*i i**3; end
```

The loop in this example is executed five times. When the DO clause is first executed, the control variable *i* is set to 1. The variable *i* is incremented after each iteration of the loop, and looping continues until the value of *i* becomes greater than 5.

So the clause

```
'input' i*i i**3
```

is evaluated five times, causing these five KEDIT INPUT commands to be issued:

```
input 1 1 1
input 2 4 8
input 3 9 27
input 4 16 64
input 5 25 125
```

Five lines are inserted in the file, each new line containing a number (from 1 to 5) along with its square and cube. Where these lines are inserted depends on how you run the F1 macro. If you issue the command

```
macro f1
```

from the command line, then the inserted lines will be added below the current line. Since F1 is the name of a key, you could also run the F1 macro by pressing the F1 key. If you press F1 while the cursor is in the file area, then the inserted lines will be added below the line that the cursor is on.

Focus line

Macros act relative to the *focus line*. If you use the MACRO command to issue a macro from the command line, or if the cursor is on the command line when you press the key that invokes a macro, then the focus line is the current line. If the cursor is in the file area when you press the key that invokes a macro, then the focus line is the line the cursor is on, and the macro acts relative to the cursor line.

10.2.2 Multi-line Macros

Multi-line macro definitions can be stored in disk files. There are two ways to create multi-line macros:

.KEX files

A file with an extension of .KEX can contain a single multi-line macro.

A macro in a .KEX file doesn't have to be loaded into memory before you run it. You can run a macro that resides in a .KEX file directly from disk by issuing the command

```
macro fname
```

where *fname* is the name of a .KEX file. Remember, too, that if IMPMACRO ON is in effect, as it is by default, then in most cases you can enter just the macro name on the command line without prefixing it with MACRO.

You can also load a .KEX file into memory for the duration of a KEDIT session. This will cost you some storage, but will make the macro run a bit faster, because it doesn't have to be read from disk every time you execute it. To load into memory a macro defined in a .KEX file, you enter the command

```
define fname.kex
```

where *fname* is the name of a .KEX file.

For example, consider the macro from the previous section which inserted into your file lines containing the integers from 1 to 5 along with their squares and cubes. Suppose you want to rewrite this macro so that each clause appears on a separate line. To do this, you can create a file named CALC.KEX that contains the following lines:

```
* enter table of five squares and cubes
do i = 1 to 5
   'input' i i*i i**3
   end
```

This macro has the same clauses as the earlier one-line version, but by adding a comment and by using separate lines for each clause, the multi-line version is much easier to read.

To load this macro into memory, you enter the KEDIT command

```
define calc.kex
```

Macros whose definitions have been loaded into memory via the DEFINE command are known as *in-memory* macros.

.KML files

Putting each multi-line macro in its own .KEX file is fine if you only have a few macros to define. However, you might have dozens of macros that you want loaded into memory every time you use KEDIT. It would be inconvenient to have separate .KEX files and DEFINEs for each of these macros. Instead, you can enter a set of macro definitions in a file having .KML as its extension (where .KML stands for "KEDIT Macro Library") and then load all of those macros into memory in one step.

To load into memory the macro definitions contained in a .KML file, you enter the KEDIT command

```
define fname.kml
```

Each separate macro definition in a .KML file begins with a *header*, which identifies the macro being defined.

```
:macroname
```

A header must begin in column 1 with a colon (":") immediately followed by the name of the macro you're about to define.

```
:f1
```

or

```
:calc
```

Following the header is the definition of the macro. For example,

```
:f1
'top'
'add 2'
```

Here again, single clauses can appear on separate lines. The definition ends when either end-of-file or the next header is reached.

Example

Suppose you want to include three macros in a file called ABC.KML. This file might contain the following lines:

```
:f1
'top'
:alt+r
':1'
'delete'
'bottom'
'delete'
::* Here is the CALC macro:
:calc
* enter table of five squares and cubes
do i = 1 to 5
'input'  i  i*i  i**3
end
```

The first macro is named F1, and so is bound to the F1 key. The macro issues the KEDIT command TOP.

The second macro is named Alt+R, and so is bound to the Alt+R key. The macro issues a series of KEDIT commands (:1, DELETE, BOTTOM, and DELETE).

The third macro is named CALC.

The third macro is preceded by a KML comment line. You can use KML comments, which are indicated by "::*" beginning in column 1, anywhere within a KML file. Unlike KEXX comments, which start with an asterisk ("*") or slash-asterisk ("/*"), KEDIT completely ignores KML comment lines and does not consider them part of a macro definition to be loaded into memory.

To load into memory the macros in the file ABC.KML, you would enter the KEDIT command

```
define abc.kml
```

File extensions

There is one other rarely-used aspect of .KML files: A macro name in a .KML file can optionally be followed by a list of one or more file extensions. For example,

```
:indent .c
```

or

```
:margins .txt .lst
```

If you don't give a list of extensions, then the definition that follows the header is always assigned to the macro named in the header. If you do give a list of extensions, then the definition is assigned to the macro only if one of the extensions listed matches the extension of the file that is the current file when the DEFINE command is issued. Extensions can be given with or without a leading period. A period alone matches a file with no extension. Once defined, the macros can be executed regardless of the extension of the current file.

10.2.3 Storing Your Macros

KEDIT normally looks in the following places for .KEX and .KML files: in the current directory, in the directories listed in your PATH environment variable (or in a different environment variable specified via SET MACROPATH). Then it looks in the "KEDIT Macros" subdirectory of your Windows Documents or My Documents folder, in the directory from which KEDIT was loaded, and in the USER and SAMPLES subdirectories of that directory.

We recommend that you keep any macros that you create in the "KEDIT Macros" subdirectory of your Documents or My Documents folder. (Documents is the usual name for this Windows folder under Windows Vista; in earlier versions of Windows, My Documents was the usual name.) The KEDIT Macros subdirectory of Documents/My Documents is created by KEDIT's install program and is a convenient place to store macros because KEDIT automatically looks in this directory when searching for macros.

As mentioned above, KEDIT also looks for macros in the main KEDIT program directory (usually C:\Program Files\KEDITW) and in its USER and SAMPLES subdirectories. However, we recommend that you not put your own files into these subdirectories, reserving them for files installed by KEDIT's install program, and using the KEDIT Macros subdirectory of Documents/My Documents for your own macros. Under Windows Vista there is another reason to avoid these directories: these directories normally cannot be written to under Windows Vista because of Vista's User Account Control security facility.

There is one exception to our recommendation that you use the KEDIT Macros subdirectory of Documents/My Documents for your own macros. If you are accessing a copy of KEDIT for Windows that is stored on a network server, or if your computer is shared by multiple users who access KEDIT for Windows from different Windows accounts, the USER subdirectory of the KEDIT program directory is a good place to put shared macros that all of the users who access that same copy of KEDIT want to use. (See the KEDIT License Agreement if you have any questions regarding licensing requirements for installing KEDIT on a network server.)

10.3 Features of KEXX

This section is an informal discussion of the KEXX language. Later you will want to refer to Reference Manual Chapter 6, "Macro Reference", where the KEXX language is summarized more formally and aspects of KEXX not covered here are documented.

KEXX has features common to most programming languages, such as control statements (called *instructions*), variables and assignments, and built-in functions.

Macros can directly issue KEDIT commands. Besides KEDIT commands that you normally issue from the command line, there are several KEDIT commands especially designed to be issued from macros. These special commands handle such tasks as cursor movement, getting information about the file being edited, and displaying dialog boxes.

10.3.1 Comments

You can use comments to document your macros. Comment lines are optional and can appear as any line of a macro. A line that begins with an asterisk (“*”) is a KEXX comment. For example,

```
* this is a KEXX comment
```

Since these comments occupy entire lines, they cannot appear on the same line as a clause of your KEXX program and cannot be used in one-line KEXX programs.

You can also use REXX style comments, enclosing comment text between slash-asterisk (“/*”) asterisk-slash (“*/”) pairs. This type of comment must begin and end on the same line, but need not occupy the entire line.

```
/* this is a comment */
X = 17   /* this comment follows a clause */
/* and this comment precedes one */ N = 19
```

10.3.2 Variables and Assignments

KEXX variable names are made up of any combination of letters and digits, as long as the first character is non-numeric. (The characters “!”, “?”, and “_” can also appear in variable names.)

The value of any KEXX variable is a character string. When you assign a numeric value to a variable, it is converted by KEXX into a character string. Values used in arithmetic expressions are converted back to numbers; an error occurs if a value is not numeric and cannot be converted. Since all KEXX variables have the same character string type, you do not need to declare variables before using them. Each variable starts out with a default value equal to its own name, translated to uppercase. For example, the initial value of the variable *name* is

```
'NAME'
```

The value of variables can be changed by assignment clauses. Assignments have the form

variable = expression

where the value of *expression* is assigned to *variable*. (Expressions are discussed in the next section.)

Here are some examples of assignments:

```
sum = 12
```

sets the value of the variable *sum* to 12.

```
calc = sum**2 * 3
```

assigns to the variable *calc* the current value of the variable *sum* squared and multiplied by 3.

```
name = 'Sara'
```

assigns the string ‘‘Sara’’ to the variable *name*.

```
parm = ''
```

sets the value of the variable *parm* to a null string.

```
fname = upper('Fred')
```

sets the value of the variable *fname* to ‘‘FRED’’, the value returned by the function UPPER('Fred').

In addition to the variables we’ve seen so far, KEXX also has compound variables, which are KEXX’s closest equivalent to arrays. A compound variable is made up of a stem and a tail, with no intervening blanks. The stem is simply a variable name, with the same restrictions as simple variable names, immediately followed by a period. The tail is a number or a variable that evaluates to a number. For example,

```
fname.3
```

is a compound variable, and if the variable *i* has the value 3, then

```
fname.i
```

is another way of referring to *fname.3*.

Compound variables are actually more flexible than we’ve described here. You can use multiple variables in the tail, giving you the equivalent of multidimensional arrays, and the variables in the tail need not have numeric values. See Reference Manual Chapter 6, “Macro Reference”, for more information about compound variables.

10.3.3 Expressions and Operators

Expressions

Here are some examples of typical KEXX expressions:

```
'Sara ' || lname
(2 + 9) * ((5**3 - 4) % sum)
12 + length('ADAM')
15.26 / 4
```

Expressions are built up from literal strings, variables, compound variables, numbers, function calls, and operators.

Literal strings

Literal strings are strings of characters enclosed in single quotes (') or double quotes ("). Here are some examples of literal strings:

```
'down 5'
"address.txt"
'today''s date'
"today’s date"
```

Notice that the last two examples illustrate how to enter a literal string that contains a single quote, in this case the string ‘‘Today’s date’’.

Variables

Variable names are any combination of alphabetic and numeric characters, with the restriction that the first character must be non-numeric. Examples of variable names are:

```
j
name1
file
```

Variable names can also contain the special characters "!", "?", and "_".

See the previous section for a discussion of variables.

Numbers

Numbers in KEXX can be integers (such as 12, 1234, 999999999, or -4321), can involve decimal points (such as 12.2, .0005, or -13.368), or can involve exponential notation (such as 12.34E5, which is equal to 1234000).

Note that in KEXX there is no real distinction between numbers and character strings. All variables are stored as character strings, but values are converted to numeric form as necessary during the processing of arithmetic expressions, and the results of expression evaluation are converted back to character string form. For example, 12 will be treated as a character string in the expression

```
12 || ' days'
```

since the operands of the concatenation operator are character strings. The value of the expression is

```
'12 days'
```

Similarly, the expression '12' will be treated as a number in the expression

```
'12' + 3
```

since the operands of arithmetic operators are numbers. The value of the expression is "15".

Function calls

Function calls are made by giving a function name immediately followed by a pair of parentheses enclosing the arguments, if any, of the function. The opening left parenthesis must immediately follow the function name, with no intervening blanks. If a function has multiple arguments, each argument is separated from the next by a comma. For example,

```
arg(1)
upper('fred')
substr(name,3,2)
min(length(name),10)
```

are all KEXX function calls. We'll discuss KEXX functions later in this chapter.

Operators

The kinds of operators that are standard in most programming languages are also available in KEXX: concatenation, arithmetic, comparison, and Boolean operators. In general, these operators work as you would expect, but the following items cover a few things about KEXX operators that you should be aware of:

Arithmetic operators

KEXX normally uses up to nine significant digits for arithmetic results. If necessary, you can use the NUMERIC DIGITS instruction, described in Reference Manual Chapter 6, "Macro Reference", to specify that KEXX use up to 1000 significant digits.

Comparison operators

Comparison operations are Boolean: expressions involving comparison operators evaluate to 1 if the expression is true and 0 if false. There are two types of comparisons in KEXX: normal comparison and strict comparison.

Examples of normal comparison operators are = (equal), > (greater than), and < (less than). Under normal comparison, a numeric comparison is made if both operands of the comparison are numeric. Otherwise, a character-by-character comparison is made. In either case, leading and trailing blanks are ignored. For example, under normal comparison

```
'hello' = 23                  0
'hello' = ' hello'            1
005 = ' 5'                    1
```

Examples of strict comparison operators are == (strictly equal), >> (strictly greater than), and << (strictly less than). Strict comparison is always a character-by-character comparison rather than a numeric comparison, and strict comparison does not ignore leading and trailing blanks. So, for example, although the numbers 005 and 5 will be treated as equal under normal comparison, the strict comparison

```
005 == 5
```

evaluates to 0 (false), since 005 and 5 do not match character-for-character.

Concatenation operators

Concatenation is the joining together of character strings. You can concatenate two strings either with a blank between the strings or without a blank.

To concatenate with a blank, you simply use a blank. For example,

```
fn = 'Adam'
ln = 'Smith'
name = fn ln
```

sets the value of the variable *name* to "Adam Smith".

To concatenate without a blank, you use the concatenation operator, ||. (The concatenation operator consists of two occurrences of ASCII character 124, which appears on most U.S. keyboards as a split vertical bar, located on the backslash key.) For example,

```
type = 'straw'
kind = 'berry'
fruit = type || kind
```

sets the value of the variable *fruit* to "strawberry".

In some cases you can concatenate without a blank simply by abutting two strings (that is, by entering them one after the other with no intervening blank). For example,

```
type = 'straw'
fruit = type'berry'
```

also sets the value of the variable *fruit* to ''strawberry''. However, this option is not always available to you. For example, KEXX would treat

```
fruit = 'straw''berry'
```

as a single literal string that contains a quote, so here the value of the variable *fruit* is set to

```
straw'berry
```

For a list of all KEXX operators, along with further examples, see Reference Manual Chapter 6, "Macro Reference".

10.3.4 Instructions

KEXX has a number of instructions, each beginning with a keyword like IF, DO, or SAY, that are used to control the flow of execution and handle tasks like displaying output messages and controlling debugging output.

Several instructions are discussed in this section:

SAY, used to display output messages;

IF, THEN, and ELSE, which handle conditional execution;

DO, END, LEAVE, and ITERATE, used to group instructions together and control loops;

EXIT, which terminates execution of a macro.

The other keyword instructions are:

PARSE, an important instruction that takes strings of data and breaks them into smaller pieces based on pattern specifications that you supply. This is useful if you need to do pattern matching and string manipulation on arguments passed to your macros or data in the files you are editing. See Reference Manual Chapter 6, "Macro Reference", for a discussion of PARSE and the related ARG and PULL instructions;

The TRACE instruction, which helps you debug KEXX macros by causing debugging output to be displayed in a special debugging window on your screen. Debugging is discussed below in Section 10.4;

CALL, PROCEDURE, and RETURN (used mainly in connection with the subroutines that you can write), and INTERPRET, DROP, NOP, SIGNAL, SELECT, and NUMERIC. All are documented in Reference Manual Chapter 6, "Macro Reference".

SAY instruction

The SAY instruction is used to output messages to the user of your macro. The output is displayed in the message area of the document window. For example,

```
do 5
   say 'Hello'
end
```

will display ‘‘Hello’’ on your screen, five times.

Conditional instructions

You will often want KEDIT to perform a set of actions only if some condition is met. In such a case, you can use the instructions IF and THEN. For example, suppose you had a macro containing the clauses

```
if pswd = 'mada'
then say 'Hello, Adam'
```

Here the message ‘‘Hello, Adam’’ will be displayed only if the value of the variable *pswd* is ‘‘mada’’.

The keywords IF and THEN each introduce a separate clause. THEN is special, though, in that it can appear on the same line as IF without a preceding semicolon. So, an alternative to the last example is

```
if pswd = 'mada' then say 'Hello, Adam'
```

Sometimes you want KEDIT to perform one set of actions if a certain condition is met, and another set of actions otherwise. In this case, you can use the IF, THEN, and ELSE instructions. For example, in a macro containing the clauses

```
if pswd = 'mada' then say 'Hello, Adam'
else say 'Incorrect password'
```

the message ‘‘Hello, Adam’’ will be displayed if the value of *pswd* is ‘‘mada’’; otherwise, the message ‘‘Incorrect password’’ will be displayed.

DO groups

In the examples we’ve looked at so far, the set of actions that are conditionally performed have been very simple—made up of a single clause. You can also group clauses together using a DO group. Any group of clauses that appears between a DO and a matching END is treated as a unit by KEXX.

This grouping together of clauses is most useful when used with conditionals. For example, suppose that you want to go to the top-of-file and insert ‘‘hello there’’ only if the value of the variable *j* is ‘‘ON’’. Your macro might then contain the lines:

```
if j = 'ON' then do
   'top'
   'i hello there'
end
```

DO loops

Besides the simple DO group we’ve just looked at, you can also use DO to control looping:

```
do name = start to finish
```

where *name* is the control variable of the loop, *start* is the initial value of that variable, and the loop terminates when the value of *name* exceeds *finish*. For example,

```
do i = 1 to 3
   'input Part number' i
end
```

will insert these three lines in your file:

```
Part number 1
Part number 2
Part number 3
```

With each iteration of the loop, the control variable is normally incremented by 1. You can modify the stepping of the loop by using BY. For example,

```
do i = 1 to 9 by 3
'input Part number' i
end
```

inserts these three lines in your file:

```
Part number 1
Part number 4
Part number 7
```

Another type of loop available in KEXX uses DO ***n***, where *n* is a number (or an expression with a numeric value). For example,

```
do 2
 'down'
 'input *****'
 'center'
end
```

issues a set of KEDIT commands—DOWN, INPUT, and CENTER—twice.

DO WHILE loops (which execute for as long as some condition is true) and DO UNTIL loops (which execute until some condition becomes true) are also supported. For example,

```
* input all powers of two < 500000
j = 1
do while j < 500000
   'input' j
   j = j * 2
   end
```

A final type of loop available in KEXX has the form is the DO FOREVER loop. For example,

```
do forever
 'down'
 'input *****'
 'center'
end
```

executes again and again, without any predetermined limit. In this example the loop will in fact go on forever—or at least until you press Ctrl+Break or Alt+Ctrl+Shift or run out of memory. You normally use the LEAVE or EXIT instructions to break out of DO FOREVER loops.

To break out of a loop early, you use the LEAVE instruction. For example, if your macro contains the lines

```
do n = 1 to 5
 if n = 3 then leave
 else 'input Part number' n
end
'down'
```

then processing of the loop stops when the value of *n* is 3, and then the clause following END is processed. So in this example, the lines

```
Part number 1
Part number 2
```

are inserted in the file, and then the KEDIT command DOWN is issued.

If the loop containing LEAVE is embedded in one or more other loops, only the loop immediately containing LEAVE is exited.

Another instruction that alters the normal processing of loops is ITERATE. The ITERATE instruction interrupts processing of a DO loop and passes control back up to the DO clause. For example, in this loop

```
do n = 1 to 5
 if n = 3 then iterate
 else 'input Part number' n
end
```

processing of the loop is interrupted when the value of the variable *n* is 3, and then *n* is incremented to 4 and processing continues. So in this case the lines

```
Part number 1
Part number 2
Part number 4
Part number 5
```

are inserted in the file.

Leaving a macro

There are two ways to exit a macro. One is simply to "run off the end" of your macro by executing the last clause of the macro. The other is to use the EXIT instruction. When KEXX processes an EXIT instruction, the macro ends immediately.

EXIT can appear anywhere in a macro, including inside a loop.

10.3.5 Commands

When KEXX comes upon a clause that isn't an assignment or an instruction, KEXX treats the clause as an expression whose value is passed to KEDIT as a command. After the KEDIT command is processed, a return code is set and control is returned to the macro. The return code is stored in a variable called *RC*. This variable can be examined to determine the success or failure of the command. The return code is set to 0 if the command completed successfully. A nonzero return code is set if the KEDIT command failed or encountered something unusual. See Reference Manual Chapter 9, "Error Messages and Return Codes", for complete details on return codes set by KEDIT commands.

It is a good idea to enclose each command in quotes, to indicate that the command is a literal string, and not a variable name. For example, suppose your macro contained a variable named *file* and this variable had been assigned the value "D" to indicate a data file. Suppose you want your macro to issue the KEDIT command FILE, and so you used

```
file
```

as a clause in your macro. The expression *file* is evaluated, giving "D", the value of the variable *file*. This is passed to KEDIT, and the command D—not the command FILE—is processed by KEDIT. Instead of FILEing the file, the KEDIT command DOWN, whose minimal truncation is D, makes the line one line below the focus line become the new focus line.

Putting quotes around KEDIT commands, as in

```
'file'
```

avoids this problem. In this case, the expression has the value "file". This string is passed to KEDIT and the KEDIT command FILE is issued, as desired.

Sometimes you will really want to include variables in KEDIT commands issued from macros. In this case, you should quote any part of the command that you want to be taken literally, and leave any variables outside of the quotes. For example, suppose you have a macro that contains these lines:

```
targ = 'block'
'copy' targ
```

The first clause assigns the value "block" to the variable *targ*. The second clause issues a KEDIT COPY command. Notice that the part of the command to be taken literally is enclosed in quotes, and the part that is a variable appears outside he quotes. Since the value of *targ* is "block", the command

```
copy block
```

is issued.

Any KEDIT command can be issued from a macro, but several KEDIT commands are used only within macros, or are particularly useful within macros. Here are brief

descriptions of some of these commands; all are fully documented in the Reference Manual.

EXTRACT command

The EXTRACT command retrieves information about the status of your KEDIT session or the contents of your file, placing the results into variables that your macro can use. For example,

```
'extract /zone/'
```

returns information to your macro about the current zone columns. With ZONE 1 255 in effect, this command would set the variable *zone.1* to 1, and the variable *zone.2* to 255.

Depending on what information you extract, the amount of information returned by EXTRACT can vary, so the EXTRACT command always sets another variable to indicate the amount of information returned. In this case, *zone.0* would be set to 2, because two pieces of information, the left zone column and the right zone column, were returned.

Frequently used EXTRACT operands include:

```
'extract /curline/'
```

This sets several variables, the most useful being *curline.3*, which is set equal to the contents of the focus line.

```
'extract /line/'
```

Sets *line.1* equal to the line number within your file of the focus line.

```
'extract /size/'
```

Sets *size.1* equal to the number of lines in your file.

Reference Manual Chapter 5, "QUERY and EXTRACT", has further discussion of the EXTRACT command, including information on the variables set for every possible EXTRACT operand. See also the discussion on page 229 of implied EXTRACT functions, which are often a convenient alternative to the EXTRACT command.

READV command

You can use the READV command to obtain information from the user of your macro. With READV CMDLINE, the user of your macro enters a line of text in the command line, and this text is then passed back to your macro as the value of the variable *readv.1*. (The PARSE PULL instruction is another way to read a line of user input from the command line.) With READV KEY, KEDIT reads a key from the keyboard and passes back to your macro information about the key that was pressed.

The following example displays a message asking for the name of the user of the macro, reads the name from the command line, and then enters the name into the current file five times.

```
say 'Enter your name:'
'readv cmdline'
do 5;'input' readv.1;end
```

DIALOG command

Another approach is provided by the DIALOG command, which reads its input from a dialog box, and which gives your macro control over the contents of the dialog box's title bar, prompt text, and icons.

Here is the DIALOG equivalent of the example just given for the READV command:

```
'dialog /Enter your name/ editfield'
do 5;'input' dialog.1;end
```

TEXT command

The TEXT command enters text at the cursor position, just as if you had typed the text on the keyboard. For example, to enter the sentence "Hello there." at the cursor position, a macro could use:

```
'text Hello there.'
```

CURSOR, SOS commands

The CURSOR command lets you reposition the cursor from within a macro. You can move the cursor to a particular line and column of the window, to a particular line and column of your file, etc. The SOS ("Screen Operation Simulation") command handles additional cursor positioning operations (such as tabbing to the next word or next window) along with numerous miscellaneous editing functions (such as deleting the word at the cursor position).

Many of the macros assigned by default to KEDIT's keys do nothing more than issue CURSOR and SOS commands.

EDITV command

Variables and variable values used within a macro are discarded when the macro finishes. In some cases, you might need to save information obtained during execution of one macro for use in a macro that executes later. KEDIT's EDITV command is available for these situations. With the EDITV command, you can access special variables that retain their values either throughout your KEDIT session or for as long as the current file remains in the ring.

NOMSG command

Sometimes commands that you issue from within a macro generate messages that you don't want the user of your macro to see. For example, you might want a macro to issue a LOCATE command and then test the variable *RC* to see if it succeeded, but you might want to avoid display of any possible error message. The NOMSG command lets you execute a command, but bypass display of any messages resulting from the command.

The following macro uppercases any "a"s, "b"s, and "c"s in your file, but skips the messages normally produced by the change command:

```
'nomsg c/a/A/ all *'
'nomsg c/b/B/ all *'
'nomsg c/c/C/ all *'
```

Focus line

It is important to remember that commands issued from macros act with respect to the focus line. When the cursor is on the command line, the current line is the focus line; when the cursor is in the file area, the line the cursor is on is the focus line. If, for example, the cursor is in the file area when a macro issues a DELETE command, the cursor line, and not the current line, will be deleted. If you are not familiar with the focus line concept, you should read Section 6.5, "The Focus Line".

10.3.6 Functions

Four types of functions can be used in KEXX macros: internal routines, built-in functions, implied EXTRACT functions, and Boolean functions.

Internal routines

You can write your own function and include it in a macro as an internal routine. This is discussed in Reference Manual Chapter 6, "Macro Reference".

Built-in functions

Many useful functions are built directly into KEXX, most having to do with character string manipulation. There are several dozen built-in functions; all are described, with examples, in Reference Manual Chapter 6, "Macro Reference". Here are some of the most frequently used built-in functions:

ARG([n[,option]])

ARG() returns information about arguments passed to macros and internal routines. ARG(1) (that is, ARG() with a parameter of 1) is the most common usage; it returns the value of the argument string passed to a macro, or returns the null string if the macro was invoked with no argument string.

For example, assume that the macro TEST is invoked via the following KEDIT command:

```
macro test I think, therefore, I am.

arg(1)          'I think, therefore, I am.'
```

D2C(n)

When given a decimal value *n* in the range 0 to 255, returns the character that has *n* as its character code.

```
d2c(97)         'a'
d2c(50)         '2'
```

LENGTH(string)

Returns the length of *string*.

```
length('314159')              6
length('a . b')               5
length('')                    0
```

LOWER(string)

Returns the value of *string* with any uppercase letters translated to lowercase.

```
lower('ABCDEF')               'abcdef'
lower('1F3De5')               '1f3de5'
```

POS(string1,string2,[start])

Returns the position of the first occurrence of *string1* in *string2*. The search starts at position *start*, which defaults to the beginning of *string2*. One frequent use of POS() is to simply test whether one string is present in another, in which case POS() will return a nonzero result.

```
pos('heat','in the heat of the night')          8
pos('heat','in the heat of the night',10)       0
pos('sleet','in the heat of the night')         0
pos('heat of','in the heat of the night')       8
```

SUBSTR(string,start[,[length][,pad]])

Returns the substring of *string* beginning at the *start* position for a length of *length* characters. If necessary, the value is padded with the *pad* character, which defaults to a blank. The default for *length* is the remaining length of *string* beginning at the *start* position.

```
substr('in the heat of',4)                  'the heat of'
substr('intheheatof',3,7)                   'theheat'
substr('intheheatof',15,2)                  '  '
substr('intheheatof',15,2,'-')              '—'
```

UPPER(string)

Returns the value of *string* with any lowercase letters translated to uppercase.

```
upper('abcdwxyz')             'ABCDWXYZ'
upper('90a3ff')               '90A3FF'
```

Implied EXTRACT functions

The KEDIT command EXTRACT is used to set a special set of compound variables associated with QUERY options. An alternative way of accessing the same information is to use implied EXTRACT functions. Implied EXTRACT functions all have the form

```
extractvar.n()
```

That is, the name of an implied EXTRACT function is the name of a variable that would have been set by the corresponding EXTRACT command. So, for example,

```
a = zone.1()
```

can be used in place of

```
'extract /zone/'
a = zone.1
```

In both cases, the variable *a* will be set to the value of the variable *zone.1* (that is, the left zone column). It is often more convenient to use implied EXTRACT functions within a macro, and an implied EXTRACT is often more efficient than the corresponding EXTRACT command.

See Reference Manual Chapter 5, "QUERY and EXTRACT", for a complete list of the information available through the EXTRACT command and implied EXTRACT functions.

Boolean Functions

A Boolean function is a special function that tests some condition within KEDIT and returns a 1 if the condition is true, or 0 if it is false.

Boolean functions are used most frequently in conditionals. For example,

```
if blank() then exit; else 'sos addline'
```

Here, the Boolean function BLANK() tests whether the cursor is on a blank line. If it is, then the EXIT instruction is processed; otherwise, the KEDIT command SOS ADDLINE is issued.

Some commonly used Boolean functions are listed in the table below:

Function	Tests whether
`AFTER()`	the cursor is after the last nonblank character of the cursor line
`BLANK()`	the cursor is on a blank line
`BLOCK()`	a block is marked
`COMMAND()`	the cursor is on the command line
`CURRENT()`	the cursor is on the current line
`CUA()`	INTERFACE CUA is in effect
`EOF()`	the cursor is on the end-of-file line

PREFIX() the prefix area is on

TOF() the cursor is on the top-of-file line

For a complete list, see Reference Manual Chapter 6, “Macro Reference”.

10.3.7 Passing an Argument to a Macro

You can pass an argument string to a macro that the macro can examine using the ARG(1) function. (The PARSE ARG instruction can also be used.) For example, suppose you have a macro named ID that displays the value of its argument string on the message line. This macro might be defined as

```
parm = arg(1)
say 'ID given was' parm
```

Entering the KEDIT command

```
macro id Adam
```

displays

```
ID given was Adam
```

and entering the command

```
macro id Adam Smith III
```

displays

```
ID given was Adam Smith III
```

The value of ARG(1) is any text following the macro name, passed as a single string to the macro.

See the notes at the end of Reference Manual Section 2.2, “KEDIT Initialization Options”, for information on passing arguments to your profile.

10.4 Debugging KEXX Macros

KEDIT includes a debugging facility that you can use to help track down problems in macros that you are developing.

When you debug a macro, KEDIT’s frame window and document windows are displayed as usual, but a special debugging window is also displayed, with output from KEDIT’s macro debugger. As your macro executes, its progress is traced, with trace output sent to the debugging window. You can use the KEXX TRACE instruction to control how much information is traced (all commands issued, results of all expressions evaluated, etc.) and whether tracing is interactive (with pauses for your input after every traced clause) or noninteractive.

Included in KEDIT to support the debugging facility are the SET DEBUGGING command, which controls whether the debugging window is active, how large it is, and the default tracing level, the DEBUG command, which controls which macros are executed with tracing in effect, and the KEXX TRACE instruction, which modifies tracing levels while a macro is executing.

10.4.1 Using the Debugger

To begin using the debugger, issue the command

```
set debugging on
```

This tells KEDIT to begin displaying the special debugging window, with a default debugging window size and with a default tracing level that interactively traces all clauses executed and shows the results of all expressions evaluated. (If you use the KEXX TRACE instruction while the debugging window is off, you will not receive any error messages, but no trace output will be displayed.)

Then issue the command

```
debug macroname
```

where *macroname* is the macro you want to debug.

KEDIT will then execute the macro, displaying trace output in the debugging window, and pausing after each clause is executed for interactive trace input.

When you are finished with the debugger, you can turn off the debugging window with the command

```
set debugging off
```

or you can use the mouse to close the debugging window by double-clicking on the debugging windows's system menu.

Interactive trace input

You can enter interactive trace input in the edit field at the bottom of the debugging window.

If you want to continue with execution of the next clause of your macro, just press Enter.

If you want to enter interactive trace input, the text you enter must be something that would be valid as a line of a KEXX macro.

KEXX takes your input and executes it as if it were part of your KEXX macro. You therefore have full access to the KEXX language from within the debugger. You can

set variables within the macro being debugged,

display variable values,

issue KEDIT commands, and

change tracing levels.

Examples

```
say total
```

This would display the value of the KEXX variable *total*. Output from the SAY instruction is normally displayed in the KEDIT message area, but when the SAY instruction is entered as interactive trace input, the output appears in the debugging window.

```
total = total + size.1()
```

This would set the value of the KEXX variable *total* to the sum of *total* and the number of lines in the current file.

```
'change /a/b/ 4 *'
```

This would cause KEXX to pass the CHANGE command to KEDIT. KEDIT would then execute this command and update its display to reflect the change that was made.

```
exit
```

This would cause KEXX to immediately end execution of your macro.

```
trace off
```

This would end interactive tracing of your macro, and the macro would finish executing without displaying additional trace output.

Issuing KEDIT commands

Two important points should be noted about issuing KEDIT commands as interactive trace input:

KEDIT commands are not issued directly from the interactive trace prompt. Your input is passed to KEXX, which interprets the input according to the normal KEXX rules before passing it to KEDIT for execution. So the quotes around the CHANGE command in the example above are necessary, just as they would be if the command were issued from within a KEXX macro, to prevent KEXX from interpreting the command as an expression involving division or multiplication.

You should use caution when issuing KEDIT commands via interactive trace that change the contents of the current file, your position in the file, or that bring a new file into the ring. You might affect the macro being debugged, since it could be in the middle of performing some operation on what was the focus line, and it might have variables whose values were set depending on the current file, or your position in the file.

10.4.2 The TRACE Instruction and the DEBUG Command

The level of trace output produced by the debugger is controlled by the KEXX TRACE instruction. The TRACE instruction can appear within a KEXX macro and it can be entered when the debugger pauses for interactive trace input. Here are the tracing levels that you can use. Note that only the first character of the TRACE setting is significant:

TRACE Off TRACE Off turns off all tracing that is in effect.

TRACE Error Any command passed to KEDIT that yields a nonzero return code is traced, as is the return code.

TRACE Command All clauses that cause commands to be issued to KEDIT are traced, as well as the commands themselves and any nonzero return codes.

TRACE All All clauses are traced as they are executed, as well as all commands issued to KEDIT and any nonzero return codes.

TRACE Results Same as TRACE All, except that the final results of all expressions evaluated are also traced.

TRACE Intermediates Same as TRACE All, except that both the intermediate and final results of all expressions evaluated are also traced.

TRACE Labels Traces labels in the macro as they are encountered during execution of the macro.

In addition to controlling the level of trace output, you can also use the TRACE instruction to turn interactive tracing on or off. When interactive tracing is in effect, the debugger will pause after execution of most traced clauses in your macro to let you enter interactive trace input, as described in the preceding section.

TRACE + turns interactive tracing on.

TRACE - turns interactive tracing off.

TRACE ? toggles the interactive tracing on if it is off, or off if it is on.

You can also use +, -, or ? in combination with one of the trace settings discussed above. For example,

```
trace +r
```

turns on interactive tracing of results, while

```
trace -c
```

causes noninteractive tracing of commands.

Notes on interactive tracing

Three points to note about interactive tracing:

TRACE is a KEXX instruction, and is not a KEDIT command. So it can only be issued from within a KEXX macro or from the interactive trace prompt, and should not be enclosed in quotes.

When interactive tracing is in effect, TRACE instructions in your macro are ignored. This is to prevent an unexpected change to tracing status while you are controlling the debugging session via interactive trace input.

TRACE Off turns off all tracing output, and it also turns off interactive tracing, whether it is or isn't prefixed with +, -, or ?.

You can also use the TRACE instruction to tell the debugger to temporarily stop pausing for interactive input, or to temporarily stop displaying trace output. To do this, specify a numeric value with the TRACE instruction:

TRACE *nnnn* A positive number tells the debugger to continue executing your macro, and to continue displaying trace output, but that at the next *nnnn* places where it would ordinarily pause for interactive input, it should instead continue without a pause.

TRACE -*nnnn* A negative number works like a positive number, in that the next *nnnn* pauses for interactive input are skipped. The difference is that during this period the display of trace output is also suppressed.

Tracing with the DEBUG command

When a macro begins execution, TRACE OFF is normally in effect. Unless the macro contains a TRACE instruction that changes this initial TRACE setting, the KEXX debugger will not be active while the macro is running. To let you debug macros without the need to edit the source of the macro and insert TRACE instructions, KEDIT provides the DEBUG command. The are three variations on the DEBUG command:

DEBUG [/*tracesetting*] *macroname*

The first form of the DEBUG command works just like the MACRO command, except that, if the debugging window is on, tracing is turned on when the macro is started. You can specify the trace setting to use via an optional DEBUG command switch, or use a default setting controlled by SET DEBUGGING (normally +R). For example, to run the macro XYZ with TRACE ALL in effect:

```
debug /a xyz
```

DEBUG START *macroname*

DEBUG START causes the in-memory macro specified to run with tracing on (using the SET DEBUGGING trace setting) whenever the macro is executed.

DEBUG STOP *macroname*

DEBUG STOP cancels the effect of a previous DEBUG START.

PROFDEBUG initialization option

To help debug problems with your profile, the KEDIT command supports the PROFDEBUG initialization option. Using this option causes KEDIT to automatically turn on the debugging window and execute your profile with debugging turned on. For example, you could start KEDIT for Windows with the command

```
keditw sample.fil (profdebug
```

10.4.3 Trace Output

KEDIT displays several different types of trace output in the debugging window. To help you identify the type of output involved, KEDIT precedes each type of trace output with a three character prefix and, on a color display, KEDIT uses different colors for different types of output. Here are the prefixes and different types of trace output:

+++ Used for messages from the debugger. Shown in blue.

- Used to display source code from your macro. Shown in black. Each clause to be traced is displayed as it is executed. The line number within the macro of the traced clause is also displayed, unless multiple clauses from the line are being executed and the line number has already been displayed.

>>> Used for the result of evaluating an expression. If the result is passed to KEDIT as a command, it is shown in green. Other results are shown in red.

The following types of trace output are displayed only when TRACE Intermediates is in effect. The output is shown in magenta:

>C> Used for derived names of compound variables in which substitution has occurred.

>F> Used for values returned from functions.

>L> Used for constant symbols, literal strings, and variables that have their default values.

>O> Used for results of operations with two operands, such as comparison and concatenation.

>P> Used for results of prefix operations, such as negation.

>V> Used for variables with non-default values.

Chapter 11. Sample Macros

Chapter 10, "Using Macros", went over the rules for defining KEDIT macros and gave an overview of the KEXX language and its debugging facilities. This chapter takes another approach, giving you a detailed look at a number of ''real life'' KEDIT macros.

Most of these macros are simple; all are useful. They have been selected primarily because they illustrate a particular feature of KEDIT's macro environment that is important to understand. They also demonstrate just how much more powerful KEDIT can become when you are able to write even the simplest of macros.

For each of the macros, the following information is presented:

a description of what the macro does;

the source code for the macro;

a line-by-line explanation of how the macro works;

a discussion of how the macro is used, including the commands used to define the macro to KEDIT and to run the macro.

11.1 Counting the Words in a File

This first example will count the number of words in a file and display the result on the KEDIT message line.

WC.KEX

This macro demonstrates how to perform an action on the contents of every line in a file using a DO WHILE loop.

```
total = 0
'TOP'
'DOWN 1'

do while rc = 0
   total = total + words(curline.3())
   'DOWN 1'
   end

say 'Word count =' total
```

Description of WC.KEX

The instructions in the macro do the following:

```
total = 0
```

Total is a macro variable whose value will contain the total number of words counted. Since the number of new words on each line is added to the existing total for the entire file, *total* must be initialized to 0 so that the first addition will work properly.

```
'TOP'
'DOWN 1'
```

These two KEDIT commands set up to start counting with the first line of the file. Another way to position yourself at the beginning of the file is with the KEDIT LOCATE command:

```
'LOCATE :1'
```

Either will work fine under most circumstances, but the combination of commands selected for this macro will work even if line 1 of the file is outside of the current RANGE boundaries or has been excluded from display by the selective line editing facility.

```
do while rc = 0
```

The bulk of the work in the macro gets done in a loop that counts all the words in the focus line and then advances to the next line in the file. The loop will terminate when the special variable *rc* has a value other than zero.

The variable *rc* always contains the return code from the last KEDIT command a macro issued. Testing it is an excellent way to determine if a command executed properly. Most commands give a zero return code if no problems are encountered and a nonzero return code if an error or an unusual situation is encountered. See Reference Manual Chapter 9, “Error Messages and Return Codes”, for a discussion of return codes.

In this case, one of the two DOWN commands will always have been the last command to execute before *rc* is tested. If its value is nonzero, that means that a DOWN command has reached the end-of-file line and there are no more lines left to count.

```
   total = total + words(curline.3())
```

This statement does the actual counting. The new value of the *total* variable is set to the old value plus the number of words in the focus line. The number of words is determined by the built-in KEXX function WORDS(), and the contents of the focus line are supplied by the implied EXTRACT function CURLINE.3().

The KEDIT EXTRACT command sets the values of certain variables in your macro to reflect information about the current state of your editing session. For example, the command

```
EXTRACT /curline/
```

would set the variables *curline.0* through *curline.6* with values that represent a number of different characteristics about the current and focus line. One of those values, *curline.3*, would be the contents of the focus line in mixed case.

CURLINE.3() is an example of a KEDIT implied EXTRACT function. The name of each implied EXTRACT function corresponds to the name of a variable that would be set by the EXTRACT command. An implied EXTRACT function returns as its value the same information that the EXTRACT function would place in the corresponding variable. It is used here because only the contents of the focus line are needed, and there’s no point to setting all those other variables.

You will find a description of all the available EXTRACT operands in Reference Manual Chapter 5, "QUERY and EXTRACT".

```
'DOWN 1'
end
```

After each line has been processed, the loop asks KEDIT to move to the next line by issuing another DOWN command. After DOWN is executed, the macro goes back to the top of the loop, and checks *rc* to see if the command was successful.

```
say 'Word count =' total
```

The macro "falls" out of the loop after all the lines in the file have been counted. The only task remaining is to report the results. KEXX will interpret the value of the *total* variable, concatenate it to the end of the string "Word count =", and SAY will put the resulting string into the KEDIT message area.

Using WC.KEX

WC.KEX is an example of a macro definition that is stored in a disk file. It can be run from the KEDIT command line by typing:

```
macro wc
```

or, with the default of IMPMACRO ON in effect, WC.KEX can be run from the command line just by typing the filename:

```
wc
```

KEDIT will search for the macro in your current directory. Next it searches the directories in your PATH; this can be changed via the SET MACROPATH command. If the macro still hasn't been found KEDIT looks in the "KEDIT Macros" subdirectory of your Windows Documents or My Documents folder, in the directory from which KEDIT was loaded, and in the USER and SAMPLES subdirectories of that directory. This search procedure will locate WC.KEX, which is installed in the SAMPLES subdirectory by KEDIT's setup program.

Once the macro is found, KEDIT will load it into memory and run it. After it has finished executing, KEDIT frees up the memory the macro occupied so that it can be used for other purposes.

If you use a macro frequently, you can load the macro into memory and keep it there for the duration of your KEDIT session. This uses more memory, but eliminates the need for KEDIT to search your disk every time you want to use the macro.

To load WC.KEX into memory you would type

```
define wc.kex
```

at the KEDIT command line, or you can put the line

```
'DEFINE wc.kex'
```

into your WINPROF.KEX file if you want the macro to be loaded every time you invoke KEDIT.

11.2 KEDIT Key Definitions

Every functional key on the keyboard is assigned a KEDIT macro; the name of the macro is the same as the name of the key. Pressing a key is equivalent to asking KEDIT to run the associated macro with the MACRO command. For example, the A key runs a macro that types a lowercase ‘‘a’’ at the current cursor location, and the F2 key normally corresponds to a macro that adds a line to your file.

Several of the keys on your keyboard are predefined to run a macro that simply does nothing. They can be identified by using the DEFINE command to display their associated macro in the KEDIT message area.

For example, DEFINE CTRL+F3 displays the definition for the CTRL+F3 key. If the response is ‘‘nop’’, the KEXX no-op statement that does nothing, then you’ve located a do-nothing key. These keys are good candidates for customized macros that make it easier for you to work with KEDIT.

This next example is a simple macro that you might find useful enough to assign to one of those do-nothing keys on your keyboard.

CTRL+F3, a pop-up command line

If you’re going to write a lot of macros, it’s important to grasp the concept of KEDIT’s focus line. The focus line is the line on which the cursor resides, unless the cursor is on the command line, in which case the current line is the focus line. There is a corresponding focus column concept that works exactly the same way.

If you run a KEDIT command from the command line, it will always act with respect to the current line—which is what you would expect. If you run a command from a macro, and the cursor is in the file area, the command acts with respect to the line that contains the cursor, which might or might not be the current line. This is often a source of confusion to new KEDIT macro writers.

This example can help demonstrate the difference. Once assigned to a key, it can be used to pop up a dialog box from which you can run any KEDIT command. Wherever the cursor happens to be when you press the key is the focus line, and the command you enter through the dialog box will act with respect to that line.

For example, if you use this macro to issue the DELETE command, it will delete the line of the file on which the cursor is located, unless the cursor is on the command line, in which case the current line is deleted.

```
* prompt for and issue a command.
'DIALOG /Enter Command/ EDITFIELD'
if rc \= 0 | dialog.2 \= 'OK' then exit
if dialog.1 \= '' then dialog.1
```

Description of CTRL+F3

The instructions in the macro do the following:

```
'DIALOG /Enter Command/ EDITFIELD'
```

This KEDIT command displays a dialog box on the screen with an editable field in it where the user can enter a command.

```
if rc \= 0 | dialog.2 \= 'OK' then exit
```

The dialog box is displayed with two buttons, OK and CANCEL. The name of the button the user activated to end the dialog box is returned in the KEXX variable *dialog.2*. If the user doesn't select OK, or the DIALOG command fails in some other respect, the macro terminates without doing anything more.

```
if dialog.1 \= '' then dialog.1
```

The KEDIT command that you typed into the edit field of the dialog box is returned in the variable *dialog.1*. The IF instruction checks to make sure something was entered and, if so, executes the statement *dialog.1*. Since *dialog.1* is not a KEXX keyword, it is considered by KEXX to be an expression whose value must be determined and then passed to KEDIT as a command. The expression *dialog.1* has as its value the contents of the variable *dialog.1*, which is the command that you entered in the dialog box. So the command that you entered is executed by KEDIT.

Defining CTRL+F3

Keyboard macros must be predefined before they can be used. KEDIT will not search your disk to find them as it would for a .KEX file.

Simple one-line macros can be directly established with the DEFINE command, but not all macros are simple ones. Fortunately, KEDIT allows you to place one or more multi-line macros into a special file known as a KEDIT Macro Library or .KML file.

Preceding each macro in a .KML file is a special label, beginning with a colon (":"), that names the macro that follows. This allows you to put several different macros in a single file one after another, and permits you to keep related macros together. It can even help improve performance when it comes time to load your macros into memory.

Using KEDIT, create a file containing the macro for Ctrl+F3, and save it in a file with an extension of .KML. As discussed in Section 10.2.3, "Storing Your Macros", we suggestthat you store .KEX and .KML files that you create in the "KEDIT Macros" subdirectory of your Windows Documents folder (which is sometimes known as the My Documents folder). You should wind up with a file that looks like this:

```
:CTRL+F3
 * prompt for and issue a command.
 'DIALOG /Enter Command/ EDITFIELD'
 if rc \= 0 | dialog.2 \= 'OK' then exit
 if dialog.1 \= '' then dialog.1
```

Even though you've created a .KML file, you're not quite finished. You can't execute a .KML file the way you can a file with an extension of .KEX. You've got to load the contents of the .KML file into memory with the DEFINE command. For example, assuming you named the file MYMACROS.KML, you would issue the command

```
define mymacros.kml
```

The define command understands the .KML format and reads the file and loads all the macros into memory. Once in memory they can be invoked like any other KEDIT macro. CTRL+F3 is special because it has the same name as one of the keys on the keyboard. You can invoke it by pressing the Ctrl+F3 key.

If you wanted to load the macros in MYMACROS.KML file every time you ran KEDIT, you could place the following line into your WINPROF.KEX file:

```
'DEFINE mymacros.kml'
```

KEDIT will search for the .KML file in the same way that it searches for .KEX files.

11.3 Working with KEDIT's Default Key Definitions

The default definition of the macro assigned to the Alt+X key is set up to help you work with DIR.DIR files—the files created by KEDIT's DIR command that contain a list of files and subdirectories.

If the current file is DIR.DIR, then pressing Alt+X will cause KEDIT to either bring a new file into the ring, reissue the DIR command for a subdirectory, or do nothing, depending on where the cursor happens to be when the key is pressed.

Alt+X can also be used to edit the file named at the cursor position in non-DIR.DIR files.

The default Alt+X definition

This example demonstrates an easy way to determine if the current file is DIR.DIR, how to find out the fileid corresponding to a line in a DIR.DIR file, and how to extend or replace KEDIT's predefined key definitions.

First, consider the default definition for Alt+X that is built into KEDIT:

```
if \dir() then
    'macro edit_cursor_file'
else if dirfileid.1() = '' then
    exit
else if pos('<dir>',curline.3()) = 0 then
    'K "'dirfileid.1()'" (nodefext'
else
    'DIR "'dirfileid.1()'"'
```

Description of Alt+X

The instructions in the macro do the following:

```
if \dir() then
    'macro edit_cursor_file'
```

This statement makes use of the Boolean function DIR() to check to see whether the current file is a DIR.DIR file. (In fact, it tests for any file with a .DIR extension.) If it isn't, then the macro calls a predefined helper macro called EDIT_CURSOR_FILE to edit the file named at the cursor position of a non-DIR.DIR file. We will focus here on the DIR.DIR handling done by Alt+X and will not discuss EDIT_CURSOR_FILE.

```
else if dirfileid.1() = '' then
    exit
```

The name of the file or subdirectory represented by the focus line in a DIR.DIR file is returned by the implied EXTRACT function DIRFILEID.1(). If the current file is a

DIR.DIR file, but DIRFILEID.1() returns a null string, then the focus line does not describe a file or a subdirectory. (This normally happens if the focus line is the top-of-file or end-of-file line.) In this case it makes sense for the macro to end without anything more being done.

```
else if pos('<dir>',curline.3()) = 0 then
   'K "'dirfileid.1()'" (nodefext'
else
   'DIR "'dirfileid.1()'"'
```

At this point the macro has verified that it is in a DIR.DIR file and that the focus line represents either a subdirectory or a file. This IF statement uses the POS() built-in function to test whether the focus line contains the string ‘‘<dir>’’, which indicates that the focus line describes a subdirectory.

If ‘‘<dir>’’ is not found, the line represents a file that is then brought into the ring for editing. Otherwise a DIR command is issued for the subdirectory in question.

Note that the macro encloses the value returned by DIRFILEID.1() in double quotes when constructing either the KEDIT or DIR command string. Windows allows fileids with embedded blanks, and any fileid with embedded blanks must be surrounded by double quotes so that KEDIT can properly determine where the fileid begins and ends.

You should also notice the use of the NODEFEXT option on the end of the KEDIT command. DIRFILEID.1() returns the value of a fileid. If that fileid doesn’t happen to have an extension and DEFEXT ON is in effect, the KEDIT command will tack the extension of the current file (.DIR) onto the end. In the context of the Alt+X macro, this behavior is undesirable, and it is overridden by the NODEFEXT option.

Working with the Alt+X macro

Since Alt+X is predefined it is available immediately as part of any editing session. Simply issue the KEDIT DIR command, move the cursor to the line in the resulting DIR.DIR file that represents a file you want to edit or a subdirectory you’d like to look over, and press Alt+X.

For the sake of this discussion let’s suppose you aren’t quite happy with the existing Alt+X macro and want it to behave a little differently. If the new macro is simple enough, you can use the DEFINE command to change it. For example, suppose you wanted Alt+X to always move you to the next file in the ring. You could change its definition by typing in the following on the KEDIT command line:

```
define alt+x 'KEDIT'
```

If you wanted the ALT+X key to work like this in every KEDIT session, you could place the following into your WINPROF.KEX file:

```
"DEFINE alt+x 'KEDIT'"
```

Because the DEFINE command is being used in a macro (WINPROF.KEX) it should be given in quotes. And since it is being used to define a macro that contains another command string, that other command string should also be in quotes. The easiest way to embed a quoted string inside another quoted string is to use double quotes for one and single quotes for the other.

If you want to make changes to macros that are more complicated, you use the .KML files that were introduced in the last example.

Any existing in-memory macro can be extracted into .KML format by the MACROS command. The MACROS command is invaluable for looking over existing macros and/or making slight adjustments that are more to your liking. With it you can look at any of KEDIT's predefined macros. Just give the MACROS command a specific macro name and that macro's definition will be placed in a file in the ring called MACROS.KML.

For example, suppose you wanted to change Alt+X to do the same things it does by default if the current file is a DIR.DIR file, but to move to the next file in the ring if it were in any other file. You might start by getting the default definition into a MACROS.KML file by typing:

```
macros alt+x
```

at the KEDIT command line. The resulting MACROS.KML file will look like this:

```
:alt+x
 if \dir() then
    'macro edit_cursor_file'
 else if dirfileid.1() = '' then
    exit
 else if pos('<dir>',curline.3()) = 0 then
    'K "'dirfileid.1()'" (nodefext'
 else
    'DIR "'dirfileid.1()'"'
```

Using KEDIT, make the changes you desire. You might come up with the following:

```
:alt+x
 if \dir() then
    'K'
 else if dirfileid.1() = '' then
    exit
 else if pos('<dir>',curline.3()) = 0 then
    'K "'dirfileid.1()'" (nodefext'
 else
    'DIR "'dirfileid.1()'"'
```

The second line of the original macro:

```
'macro edit_cursor_file'
```

has been changed to move to the next file in the ring

```
'K'
```

The rest of the macro is essentially the same as the original default definition.

As with CTRL+F3, you can file this new macro away in a .KML file with any name you choose, though it's a good idea to avoid the name MACROS.KML. KEDIT uses it automatically, and you might accidentally replace your own one day and lose any macros stored in it.

11.4 Saving Your Place in a File

It's not unusual during an editing session to want to save your place so that you can wander off, look at other parts of a file, and still find your way back quickly. You can use the Set Bookmark 1 and Go to Bookmark 1 buttons on the optional bottom toolbar to do this for a single location in your file, or you can use the Actions Bookmark dialog box to work with multiple bookmarks. Another method is to use the sample macro discussed here. Once assigned to a key, this macro will not only save your place at the touch of that key, but it will also find its way back later on if you press the key twice in succession.

XPOINT.KEX

This macro demonstrates the READV KEY command and the concept of named lines. It also makes interesting use of the fact that every key on the KEDIT keyboard is assigned a macro. Pressing a key is equivalent to asking KEDIT to run the associated macro with the MACRO command.

```
'EXTRACT /lastkey/'
'READV KEY NOIGNOREMOUSE'
if rc = 0 then do
   if lastkey.1 \= readv.1 then do
      'SET POINT .'lastkey.1
      'MACRO' readv.1
      end
   else
      'LOCATE .'readv.1
   end
else if rc = 2 then
   'SET POINT .'lastkey.1
```

Description of XPOINT.KEX

The instructions in the macro do the following:

```
'EXTRACT /lastkey/'
```

The macro needs to know the name of the key that was pressed to invoke this macro if it is going to be able to check to see if the key was pressed twice in succession. EXTRACT will store this keyname in the variable *lastkey.1*.

```
'READV KEY NOIGNOREMOUSE'
```

Next, the READV command waits for another key press or a mouse event; either will terminate the command. If a mouse event was detected, the return code from the command will be nonzero. If a key was pressed, *rc* will be 0 and the name of the key will be stored in the variable *readv.1*.

```
if rc = 0 then do
```

This IF statement is used to separate out mouse events. A key press gets handled by the THEN clause; mouse events are handled by the corresponding ELSE at the end of the macro.

```
if lastkey.1 \= readv.1 then do
    'SET POINT .'lastkey.1
    'MACRO' readv.1
    end
```

The next IF statement knows it has the name of the key just pressed in *readv.1* and checks to see if it is the same as the one that EXTRACT stored in *lastkey.1*.

If the keys don't match, then two different jobs have to be attended to. First, the line must be named so that it can be found later on. The SET POINT command can be used to name a line, and a good name to use is the name of the key the macro is assigned to since it will be unique for each key this macro is associated with.

The macro must also process the second key that was pressed, since it was read by the READV command instead of KEDIT's normal keyboard handler. Each and every key on the keyboard has a macro assigned to it. KEDIT's keyboard handler executes the macros automatically every time it handles a key press. XPOINT.KEX needs to be a little more explicit and runs the macro that is bound to that key with the MACRO command.

```
else
    'LOCATE .'readv.1
```

If the keys match, then a double key press must be handled and the line named earlier when the key was pressed only once must be located.

```
else if rc = 2 then
   'SET POINT .'lastkey.1
```

This is the ELSE clause that belongs to the IF statement that checks the return code from the READV command. If it was nonzero, then it needs to be checked again to see if it was 2 which indicates that a mouse event was responsible for terminating READV. The mouse event itself is automatically queued up by KEDIT and is processed after the macro terminates.

Since a mouse event means that two identical key presses didn't happen in succession, a place saving is called for and SET POINT is used to name the current location.

Using XPOINT.KEX

KEDIT's setup program normally installs XPOINT.KEX in the SAMPLES subdirectory of the main KEDITW directory. XPOINT.KEX needs to be assigned to a key for it to work properly. This can be done by typing the following command on the KEDIT command line:

```
define alt+1 'macro xpoint'
```

This associates a macro that runs the XPOINT.KEX macro with the ALT+1 key. When the ALT+1 key is pressed, the MACRO XPOINT command is executed which will go through the process, discussed earlier, of looking for XPOINT in memory and then on disk in a .KEX file.

To set up three keys, Alt+1, Alt+2, and Alt+3 to run XPOINT, letting you maintain three separate bookmarks, you could place the following lines into your WINPROF.KEX. Since XPOINT is a short macro that you might use often, a DEFINE command loading XPOINT into memory is also included.

```
"DEFINE alt+1 'macro xpoint'"
"DEFINE alt+2 'macro xpoint'"
"DEFINE alt+3 'macro xpoint'"
'DEFINE xpoint.kex'
```

11.5 Saving the Contents of All Changed Files

This macro checks each file in the ring to see if it has been altered. If a file has been changed, a dialog box is displayed asking the user whether or not it should be saved to disk.

A NOPROMPT option is available to save all of the altered files without first asking for the user's permission.

SAVEALL.KEX

This example demonstrates how it is possible to move through the ring one file at a time, features the use of dialog boxes for displaying messages that allow the user to control the actions taken by the macro, and uses PARSE UPPER ARG to handle any supplied parameter. It also shows how to display multiple lines of output in a dialog box, and the use of the DELIMIT() built-in function.

```
parse upper arg prompt_option

do nbfile.1()
   'KEDIT'
   if \alt() then iterate

   * skip prompting for DIR.DIR
   if dir() then iterate

   if prompt_option \= "NOPROMPT" then do
    'DIALOG /Save changes to' Fileid.1()'/ YESNOCANCEL'
    if dialog.2 = "CANCEL" then exit 1
    if dialog.2 = "NO" then iterate
   end

   'NOMSG SAVE'
   if rc \= 0 then do
      lf = d2c(10)
      msg = 'Unable to save changes to' fileid.1()
      msg = msg || lf || lastmsg.1()
      'ALERT' delimit(msg) 'TITLE /Save Error/ OKCANCEL'
      if alert.2 = "CANCEL" then exit 1
   end

end
```

Description of SAVEALL.KEX

The instructions in the macro do the following:

```
parse upper arg prompt_option
```

The first line of the macro uses a PARSE UPPER ARG statement to save the uppercased value of any argument string passed to the macro in the variable *prompt_option*. Uppercasing the value makes it easier to test for a specific value later

on. See Reference Manual Chapter 6, “Macro Reference”, for a full discussion of PARSE, one of KEXX’s most useful and powerful instructions.

```
do nbfile.1()
```

NBFILE.1() is an implied EXTRACT function that returns the number of files in the ring. This DO loop is going to be executed once for each of those files.

```
   'KEDIT'
```

The KEDIT command without any operands moves to the next file in the ring. Each time through the DO loop this command sets up a different current file for the macro to work with.

```
   if \alt() then iterate
```

ALT() is a Boolean function that only returns 1 (TRUE) if the current file has been altered somehow. If the file hasn’t been changed, then there is nothing to save.

The ITERATE statement instructs KEDIT to skip the rest of the current DO loop. Control returns to the top of the loop, and starts with the next iteration if appropriate. If there aren’t any more files left, control will fall out of the loop to the end of the macro.

```
   * skip prompting for DIR.DIR
    if dir() then iterate
```

If the current file is a DIR.DIR file, even though it has been changed, it’s unlikely that those changes need to be saved to disk; it is ignored.

```
   if prompt_option \= "NOPROMPT" then do
```

Because it is usually better to err on the side of caution, the macro interprets any value passed to the macro other than ‘‘NOPROMPT’’ as meaning ‘‘PROMPT’’. If the value is ‘‘NOPROMPT’’, the logic asking for permission is skipped and any changed files always get saved.

```
      'DIALOG /Save changes to' fileid.1()'/ YESNOCANCEL'
       if dialog.2 = "CANCEL" then exit 1
       if dialog.2 = "NO" then iterate
     end
```

The DIALOG command prompts the user for instructions about each changed file. The dialog box displayed will have three buttons: YES, NO, and CANCEL. The name of the button the user selects is returned in the variable *dialog.2*.

If the user selects CANCEL, the entire process is aborted and the EXIT statement stops the macro before it can go any further. A return code of 1 is specified, indicating that the macro didn’t succeed. This return code can be checked by any other macros that ‘‘call’’ SAVEALL.KEX with the MACRO command.

If NO was selected, then the remainder of the loop is skipped, and the current file doesn’t get saved.

```
'NOMSG SAVE'
```

If execution gets this far, the file has been changed, it isn't a DIR.DIR file, and the user requested that the file be saved either by selecting the YES button in the dialog box or specifying "NOPROMPT" as an argument to the macro.

The NOMSG command tells KEDIT to issue the command that is passed as an argument without displaying any resulting messages on the screen. (There will be more to say about this shortly.) In this particular case, it is used to issue the SAVE command that will attempt to save the contents of the altered file.

```
if rc \= 0 then do
```

The IF statement checks the value of *rc* after the SAVE command is issued to determine if SAVE was able to do what was expected of it. If *rc* is nonzero, something went wrong, and the user should be notified. When a command fails, KEDIT usually displays a message relating something that the user might need to know.

Under most circumstances, KEDIT does not update the screen when a macro is running. Any messages that were "displayed" during a macro's execution usually show up only when the screen is refreshed after the macro finishes.

If a message results when the SAVE command is issued for one of the files in the ring, it might not be visible at the end of the macro because switching to another file in the ring will clear the KEDIT message area. If there are more files to work with, the message will be gone before the end of the macro and won't be shown when the screen is finally refreshed. Even if the messages were to remain on the screen, it wouldn't always be clear to which file a particular message corresponded.

Instead of relying on KEDIT to display the error messages, the SAVE command is issued through the NOMSG command, and the macro displays any resulting messages itself in an alert dialog box. This has the added advantage of allowing the user the opportunity to decide whether the error can be ignored, or to end the macro before it goes any further.

```
lf = d2c(10)
```

This statement sets the value of the variable *lf* to the linefeed character (character code 10), which is needed later in the macro. Use of the built-in function D2C(10) is a technique that allows a macro to use linefeeds without embedding the actual character directly into the source, which would be a problem since KEDIT would treat it as an end-of-line character when reading in your macro.

Of course, not everyone can easily interpret D2C(10) as a linefeed character. Saving the value in a variable called *lf* helps to make the reference less obscure.

```
msg = 'Unable to save changes to' fileid.1()
msg = msg || lf || lastmsg.1()
```

LASTMSG.1() is an implied EXTRACT function that returns the value of the last message issued by KEDIT. Even when a message doesn't get displayed on the screen, it is still kept internally and can be retrieved by LASTMSG.1(). The message to be displayed by the macro will contain the name of the current file, which is returned by

FILEID.1(), and the error message issued by the SAVE command. A linefeed character is used to split the two lines at an appropriate place.

```
    'ALERT' delimit(msg) 'TITLE /Save Error/ OKCANCEL'
     if alert.2 = "CANCEL" then exit 1
end
```

The format of the ALERT command requires that you put delimiter characters around the text to be displayed. The DELIMIT() built-in function is used to find a valid KEDIT delimiter character that is not a part of the value of *msg*. DELIMIT(MSG) returns a string that consists of the value of *msg* surrounded by appropriate delimiter characters.

Note that a single text string passed to the ALERT command contains text that ALERT will actually display on different lines of your screen. Linefeed characters have been added to the text. Wherever the ALERT (or DIALOG) command sees a linefeed character in the string that it displays, it puts the text that follows on a new line of the dialog box.

An ALERT dialog box containing this text is issued with two buttons: OK and CANCEL. The name of the button the user selects is returned in the variable *alert.2*. CANCEL is interpreted as being a request to end the macro without going any further. OK means that the user has decided to live with the error and let the macro proceed.

Using SAVEALL.KEX

KEDIT's setup program normally installs SAVEALL.KEX into the SAMPLES subdirectory of the main KEDITW directory. SAVEALL.KEX is another example of a disk-based macro and is run in much the same way as WC.KEX. The primary difference with SAVEALL.KEX is that it accepts a parameter.

To use SAVEALL.KEX to save all the files in the ring without prompting for permission, type

```
saveall noprompt
```

on the KEDIT command line. The string ''noprompt'' is passed to the macro as an argument. It is uppercased and saved for later use in the variable *prompt_option* by the PARSE UPPER ARG statement.

If you want the macro to prompt for permission, no parameters are necessary. Simply type

```
saveall
```

11.6 Batch Macro Operations

It is sometimes necessary to make the same changes in several different files. Loading each file into KEDIT, making the changes, and saving the file back to disk can get tedious and is subject to mistakes. This section discusses a macro that can help automate the process, to save time and reduce the likelihood of errors.

BATCH.KEX

This example macro can help automate the process of making repetitive changes in your files. You first use the DIR command to create a DIR.DIR file that lists the files to be changed. Then you run BATCH.KEX, which steps through each of the files and, for each file, calls a second, task-specific macro to actually make the changes.

```
if \dir() then do
   'EMSG Current file must be DIR.DIR file'
   exit 1
   end

parse arg macroname
if macroname = '' then do
   'EMSG Macro name not specified'
   exit 2
   end

'TOP'
'DOWN 1'
do while rc = 0
   'MSG Processing line' line.1()
   'REFRESH'
   'KEDIT "'dirfileid.1()'" (nodefext profile' macroname
   if rc \= 0 | \dir() then do
       'EMSG File not properly processed'
       exit 3
       end
   'DOWN 1'
end
```

Description of BATCH.KEX

The instructions in the macro do the following:

```
if \dir() then do
   'EMSG Current file must be DIR.DIR file'
   exit 1
   end
```

The Boolean function DIR() is used to make sure that this macro is being run from a DIR.DIR file. If it isn't, the macro exits with an error message because continuing doesn't make much sense.

```
parse arg macroname
if macroname = '' then do
   'EMSG Macro name not specified'
   exit 2
   end
```

The name of the ''batched'' macro is supplied as an argument. The PARSE instruction saves that name in the variable *macroname*. *Macroname* is then checked to be certain that an argument was specified. If not, the macro exits with an error message.

```
'TOP'
'DOWN 1'
```

Much like WC.KEX, BATCH.KEX will loop through all the lines in a file, though in this case the file is a DIR.DIR file containing the names of the files to be processed. These commands set up the loop that follows by moving to the first line of the file.

```
do while rc = 0
   'MSG Processing line' line.1()
   'REFRESH'
```

Next the macro uses a DO WHILE loop to step through each line in the DIR.DIR file. The loop terminates when one of the DOWN commands fails to set *rc* to 0, indicating that the macro has reached the bottom of the DIR.DIR file. Each individual line in the DIR.DIR file represents a file that you want to process. Before processing a file, the macro displays a progress message. Messages generated while macros execute are normally not displayed until the macros complete or KEDIT pauses for user input. The REFRESH command forces KEDIT to update the display, so that you can see the progress message immediately.

```
   'KEDIT "'dirfileid.1()'" (nodefext profile' macroname
   if rc \= 0 | \dir() then do
       'EMSG File not properly processed'
       exit 3
       end
   'DOWN 1'
end
```

Next, the macro edits the file described in the DIR.DIR file's focus line, specifying the ''batched'' macro as an alternate profile, instead of the normal WINPROF.KEX. The macro uses DIRFILEID.1() to get the name of the file listed at the focus line, and the NODEFEXT option to be sure that a file with no extension is edited properly; both of these were discussed above in connection with the Alt+X macro. It is the job of the batched macro to make any necessary changes to the file, and then to remove it from the ring. At that point, the DIR.DIR file should once again be the current file; if it isn't, or if the KEDIT command had a non-zero return code, something has gone wrong and BATCH.KEX exits with an error message. Otherwise, BATCH.KEX moves down to the next line of DIR.DIR, in preparation for the next iteration through the loop.

Using BATCH.KEX

BATCH.KEX isn't useful all by itself; the ''batched'' macro must be supplied and has a few responsibilities of its own. This ''batched'' macro is entirely responsible for making all the desired changes and seeing to it that they are saved and that the file is removed from the ring when it is finished.

For example, the following macro might be used to change all occurrences of ''Superman'' to ''Clark Kent'' and then save the results:

```
'CHANGE /Superman/Clark Kent/ all *'
if alt() then 'FILE'
         else 'QUIT'
```

The first line of this macro issues the appropriate CHANGE command after the file has been loaded into memory by BATCH.KEX. Since there's no sense saving the file if nothing has changed, the second line uses the Boolean function ALT() to see if the contents of the file were changed. If changes were made they are filed away; otherwise the file is simply removed from the ring by the QUIT command.

Suppose that you want to change all of the references to Superman in all the .TXT files in your C:\COMICS subdirectory. First you type the macro above and save it in a file; let's call the file TOKENT.KEX.

Next you create the DIR.DIR file with the names of all the files that need to be changed by typing, for example:

```
dir c:\comics\*.txt
```

Finally, you would run BATCH.KEX from the command line of the DIR.DIR file, specifying TOKENT as the batched macro:

```
batch tokent
```

KEDIT's setup program normally installs BATCH.KEX and TOKENT.KEX in KEDITW's SAMPLES subdirectory.

11.7 Putting Sequence Numbers into a File

Sequence numbers originated in the days of punched cards; by putting the number of the card in a specific place, cards could be sorted into order after they were carelessly dropped or otherwise mishandled. Today, sequence numbers are often used on mainframe systems as a way to keep track of particular lines in files.

This macro will put sequence numbers into the lines of a file. For example, all lines of a file might have numbers in columns 73—80, growing in increments of 1000.

SERIAL.KEX

SERIAL.KEX brings together some of the things that were introduced elsewhere in this chapter. Much like WC.KEX, it uses a DO WHILE loop to work with each line in a file. Furthermore, the macro uses PARSE ARG to save the values of any arguments supplied when it was invoked.

It also demonstrates a technique for supplying default values for arguments that weren't specified, addresses some ALT and AUTOSAVE considerations for macros that make a large number of changes to a file, and makes use of a user-defined subroutine to check the validity of any supplied parameters.

```
parse arg incr startcol width .

if incr = '' then incr = 1000
if startcol = '' then startcol = 73
if width = '' then width = 8

call argcheck "increment", incr
call argcheck "starting column", startcol
call argcheck "width", width

'EXTRACT /alt/autosave/'
'SET autosave off'

'CLOCATE :'startcol
seq = incr
'TOP'
'DOWN 1'
do while rc = 0
   'COVERLAY' right(seq, width, '0')
   seq = seq + incr
   'DOWN 1'
   end

'SET alt' alt.1 + 1 alt.2 + 1
'SET autosave' autosave.1
exit

argcheck:
parse arg type, integer
if \datatype(integer, "N") then do
   emsg "Invalid" type":" integer
   exit 1
   end
return
```

Description of SERIAL.KEX

The instructions in the macro do the following:

```
parse arg incr startcol width .
```

This PARSE statement processes any arguments to the macro and saves their values in the variables *incr*, *startcol*, and *width*. Because there is a ".." at the end of the instruction, any arguments beyond the first three will be ignored.

```
if incr = '' then incr = 1000
if startcol = '' then startcol = 73
if width = '' then width = 8
```

These three instructions set default values for any of the arguments that weren't specified when the macro was invoked. This is a commonly used technique, and it is a good idea to put this sort of thing at the beginning of the macro where it is easier to find and change.

The default values specified here represent sequence numbers that are commonly found in files on IBM mainframe systems.

```
call argcheck "increment", incr
call argcheck "starting column", startcol
call argcheck "width", width
```

If it's possible, checking user supplied arguments is always a good idea. It will often prevent macros from failing in unexpected and confusing ways.

Each of these subroutine calls checks an argument to see if it represents a valid value. The first parameter in each statement is informational, and will be used to construct a message that will notify the user of any errors. The second parameter is the value to be checked.

Note that the CALL statement is used to invoke the subroutine as opposed to the function reference. CALL is used here because ARGCHECK doesn't return a value to the caller.

There are a couple of common syntax mistakes people make when using the CALL instruction. Note that there are not any enclosing parentheses around the arguments, and that the parameters are separated from one another by commas.

```
'EXTRACT /alt/autosave/'
'SET autosave off'
```

This macro will be changing each line in the file, one line at a time. Even so, the changes will be made to look like a single alteration. This will require some direct manipulation of the ALT counter that KEDIT displays on the status line. The EXTRACT command is used to save the current alteration counts in *alt.1* and *alt.2*. These values will be used later on to set the ALT count so that it looks as if only a single change was made to the file by the macro.

To further enhance the pretense of a single global file change, AUTOSAVE will need to be turned off while the changes are being made to the file. EXTRACT will save the current AUTOSAVE setting in *autosave.1*, so that it can be properly restored at the end of the macro.

```
'CLOCATE :'startcol
seq = incr
'TOP'
'DOWN 1'
```

The next group of statements sets things up so that when the loop starts, it will have the proper set of conditions in place for the first iteration. The CLOCATE command sets the current column to be the one in which the sequence numbers are to start. The variable *seq* will always contain the sequence number that is to be put on the next line. It is initially set to the value appropriate to the first line in the file. Finally, the TOP and DOWN commands are used to move to the first line in the file.

```
do while rc = 0
   'COVERLAY' right(seq, width, '0')
   seq = seq + incr
   'DOWN 1'
   end
```

These statements constitute the DO WHILE loop; you should notice that it is very much like to the ones used in both WC.KEX and BATCH.KEX. *RC* will reflect the

success or failure of the last DOWN command, and the loop will terminate when there are no longer any lines left to process.

The COVERLAY command is used to put the sequence number into the line. The RIGHT() built-in function is used to pad the value stored in *seq* with leading zeros until the number is *width* characters wide.

After the sequence number has been added, *seq* is incremented in anticipation of the next iteration of the loop. Finally, the next DOWN command is issued, and if there's another line left, the macro does it all over again.

```
'SET alt' alt.1 + 1 alt.2 + 1
'SET autosave' autosave.1
```

After the loop finishes making changes to the file, all that remains is to set the ALT count to reflect a single change and to restore the AUTOSAVE setting.

```
exit
```

An EXIT statement immediately ends the execution of a macro. In this case one is necessary to keep the macro from falling through to the statements that make up the ARGCHECK subroutine.

```
argcheck:
```

This is the KEXX label that identifies the beginning of a subroutine called ARGCHECK.

Since the process of checking parameters for validity is needed in several different places in this macro and since it isn't completely trivial, the ARGCHECK subroutine is used to accomplish the task. This helps reduce the complexity of the macro.

```
parse arg type, integer
```

The first line of the function uses a PARSE ARG statement to save the values passed as arguments into the variables *type* and *integer*.

Since two strings are passed with each function call, *type* and *integer* must be separated by a comma. Otherwise, *type* would be assigned the first word of the first string, *integer* would be assigned whatever was left of the first string, and the second parameter, which contains the number to be verified, wouldn't be assigned to anything at all.

```
if \datatype(integer, "N") then do
   emsg "Invalid" type":" integer
   exit 1
   end
return
```

The rest of the ARGCHECK subroutine is fairly simple. The *integer* argument is checked to be certain it represents a numeric value with the DATATYPE() built-in function. If it checks out, control is returned to the caller by the return statement. If the argument isn't numeric, then it is presumed bad, so the *type* argument is used to help construct an informative error message and the macro terminates.

Using SERIAL.KEX

KEDIT's setup program normally installs SERIAL.KEX in the SAMPLES subdirectory of the main KEDITW directory. You can run it by, for example, entering the following on the command line to put sequential line numbers in columns 1 through 5 of your file:

```
SERIAL 1 1 5
```

11.8 Macros and KEDIT's Toolbar

The toolbar makes it easy to access many KEDIT facilities. When you click on a toolbar button with the mouse KEDIT runs a corresponding macro, and you can customize the toolbar by supplying your own buttons and macros.

A Lock button

This example demonstrates how to add new buttons to the toolbar and how to define their corresponding macros. The example puts a Lock button on the toolbar, which will toggle the file locking status of a file, issuing the LOCK command to lock a file that is not locked, or issuing the UNLOCK command to unlock a file that is locked. See Chapter 12, "File Processing", for a discussion of KEDIT's file locking facility.

Adding Lock to the toolbar will let you lock or unlock files you are editing with the mouse instead of requiring you to use the keyboard. Many other KEDIT commands might be useful to have on the toolbar as well. It is completely customizable, and you're encouraged to experiment.

Three main steps are involved in adding a button to the toolbar:

Use the SET TOOLBUTTON command to define the button's name, visual appearance, and help text.

Use the DEFINE command to define the macro to be executed when the button is pressed.

Use the SET TOOLSET command to position the button on the toolbar.

Adding the following macro line to your WINPROF.KEX file will create a new toolbutton:

```
'set toolbutton lock lock.bmp /Lock or unlock file/'
```

The SET TOOLBUTTON command is used to create a toolbutton. In this case, a button named "Lock" is being created. The bitmap used to paint the button on the screen is located on disk in a file called LOCK.BMP. When the mouse pointer is over the button, "Lock or unlock file" is displayed in a pop-up help box and on the status line.

In addition to defining the toolbar button with SET TOOLBUTTON, you need to supply the associated macro. The macro should be given the name TOOL_LOCK, because when you click on a toolbar button, KEDIT executes a macro called TOOL_*name*, where *name* is the name you used as the first operand of the SET TOOLBUTTON command. Here is the macro:

```
if filestatus.1() \= 'NONE' then 'UNLOCK';else 'LOCK'
```

It is short enough to define as a one-line macro directly in your WINPROF.KEX file:

```
"DEFINE tool_lock if filestatus.1() \= 'NONE' then 'UNLOCK';else 'LOCK'"
```

The macro is very simple. If the file is currently unlocked, the LOCK command is issued, otherwise the macro will issue the UNLOCK command.

Finally, the toolbutton must be added to the current toolbar. This requires another line in your WINPROF.KEX file:

```
'set toolset top add Lock'
```

The SET TOOLSET command is used to construct the list of toolbuttons that are displayed on the toolbars at the top and the bottom of KEDIT's main window. Here, the ADD operand specifies that the Lock button is added to the end of toolset that displays at the top of the window.

So you could add the Lock button to your toolbar by including the following lines in your WINPROF.KEX file:

```
if initial() then do
  'set toolbutton lock lock.bmp /Lock or unlock file/'
  "DEFINE tool_lock if filestatus.1() \= 'NONE' then 'UNLOCK';else 'LOCK'"
  end
'set toolset top add Lock'
```

When REPROFILE ON is in effect, your profile is re-executed whenever a file is added to the ring. Since the SET TOOLBUTTON and DEFINE commands need only be processed once in each editing session, they are executed in this example only if the Boolean function INITIAL() returns 1, which it will do only for the first execution of your profile, at the start of an editing session. The SET TOOLSET command, on the other hand, is not dependent on the value of INITIAL(), so that if REPROFILE ON is in effect it will update the toolbar displayed for each file that is added to the ring.

KEDIT's setup program normally installs the bitmap used in this example, LOCK.BMP, in the SAMPLES directory of the main KEDITW directory. If you want to create your own toolbar bitmaps, you can do so with most paint programs and resource editing programs. For additional information on defining your own toolbar buttons, including additional examples, see the descriptions of SET TOOLBUTTON and SET TOOLSET in Reference Manual Chapter 4, "The SET Command".

Even more examples

Your entire KEDIT package is filled with examples of how to write macros. One good place to look is in KEDITW's SAMPLES subdirectory, where most of the macros discussed in this chapter, as well as a number of other sample macros, are usually installed by KEDIT's setup program. Reference Manual Chapter 7, "Built-in Macro Handling", has a discussion of the different types of macros built into KEDIT, and the file BUILTIN.KML in KEDITW's SAMPLES subdirectory gives the definitions of all of KEDIT's built-in macros. Additional macros can be downloaded via our Internet site at http://www.kedit.com.

Chapter 12. File Processing

This chapter discusses three important aspects of how KEDIT processes the files that you edit.

First, it covers KEDIT's file locking facility and the related timestamp checking capability. These features help you work with files that other users or other programs might try to access while you are using KEDIT to edit them.

Next, it discusses the file formats that KEDIT can process and the options you can use to control KEDIT's handling of tab characters, end-of-file characters, end-of-line sequences, etc.

Finally, it gives information about KEDIT's handling of long filenames.

12.1 File Locking

When a program like KEDIT wants to begin using a disk file, it tells the operating system to open the file. When it has finished using the file, it tells the operating system to close the file. When a program opens a file, it can ask the operating system to restrict the access that other programs have to the file; these restrictions are only in effect for as long as the file remains open.

KEDIT normally has a file open only when it is reading the file into memory at the start of an editing session and when it is writing it to disk during a save operation. Since all of the editing that you do is actually done on the copy of the file that KEDIT keeps in memory, KEDIT has no need to use the disk file, and hence no need to have the file open, during most of your editing session. This can cause problems on a network, where two or more users might try to access the same file at the same time, or even on an individual PC, where one user might try to access the same file from two different programs at the same time. Here is an example of the problem:

Sharon begins to edit a file with KEDIT.

Unaware of what Sharon is doing, Robert begins to edit the same file.

Sharon finishes with the file, and uses File Close to write the file to disk and remove it from the ring.

Robert finishes with the file, and also uses File Close to write his modified copy to disk. Both users think that their changes are safely on the disk, but Sharon's version of the file has been completely replaced by Robert's.

KEDIT's file locking facility, controlled by the SET LOCKING command, provides a solution to this problem. With the default of LOCKING OFF in effect, KEDIT opens files only briefly, as described above. But with LOCKING ON, KEDIT keeps files open for as long as you are editing them and prevents access to the files by other users.

In the example above, if Sharon had set LOCKING ON before editing the file, Robert would have gotten an error message as soon as he tried to begin editing it.

LOCKING ON is a global setting, affecting each new file as it is added to the ring. The status of files already in the ring is not affected when you issue the SET LOCKING command. If you want to lock a file that is already in the ring but is not already locked you can use the LOCK command, which locks the current file. If the current file is locked and you want to unlock it, you can use the UNLOCK command.

You can tell whether a file you are editing is locked by looking at the status line at the bottom of the frame window. If a file is locked, KEDIT displays ‘‘Lock’’ in the box to the right of the Insert/Overtype Mode indicator.

You can override the status of LOCKING when you begin editing a particular file by using the LOCK and NOLOCK options of the KEDIT command. For example, if LOCKING ON is in effect because you want to lock most of the files that you edit, but you want to edit the file ABC.TXT without locking, you could issue the command

```
KEDIT ABC.TXT (NOLOCK
```

Special considerations

There are several special considerations and special cases that you should be aware of if you plan to use file locking with KEDIT:

The advantage of KEDIT’s file locking facility is that it can prevent access to files you are editing by other users on a network and by other processes on your own computer. There are, however, situations where this can be a disadvantage. For example, if you are using KEDIT to look at a file, and do not intend to change the file, the file locking facility can needlessly prevent other users from doing the same thing. See the description in the Reference Manual of the SET SHARING command, which provides some degree of control over the sharing modes used by KEDIT and can sometimes help with problems like this.

File locking can also cause problems with compilers. For example, assume you are using file locking while you are editing the source code of a C program and you use File Save to copy your changes to disk while leaving the file locked and in the ring. If you then attempt to process the file with your C compiler, the C compiler might be unable to access your file because it is locked.

The SET BACKUP command asks KEDIT to keep a backup copy of any existing file that you are editing before saving a modified version. KEDIT does this by renaming the existing copy, giving it an extension of .BAK, and then creating a new file to hold the modified version of the contents. It is not possible to rename a locked file, so when LOCKING ON is in effect, KEDIT must unlock your file, rename it to have the .BAK extension, and then create a new locked file to hold the modified contents of the file. During the brief period between the unlocking of the old file and the creation of a new file, it is possible that another user on the network could access the file, bypassing the protection normally provided by SET LOCKING. The chances of this happening are very slight, but you should be aware that it is a possibility.

When you edit a new file with KEDIT and LOCKING ON is in effect, there is a potential problem. The file does not yet exist, so it cannot be opened and locked, but if another user were to create a file with the same name before the new file is saved to disk, the other user's file could be overwritten. KEDIT resolves this problem by creating a new file on disk, zero bytes in size, when you edit a new file with LOCKING ON. If you later close the file without ever writing your version of the file to disk, KEDIT then erases the zero byte disk file that it created.

LOCKING ON does not cause files on your A: and B: drives to be locked. This is to reduce the possibility of your switching disks in a floppy drive while files are open on that drive. If you need to lock a file on your A: or B: drive, you can still use the LOCK initialization option or the LOCK command.

It is not possible for KEDIT to lock a disk file that has its Read Only attribute set. But since it is also not possible for KEDIT or any other program (except for utility programs that change the Read Only attribute) to overwrite a Read Only file and cause unexpected changes to the file, there is generally no need to lock such a file. Therefore, even if LOCKING ON is in effect, KEDIT does not attempt to lock Read Only files that you edit, and KEDIT does not consider this to be an error. To let you know that you are editing a Read Only file, KEDIT displays ''R/O'' on the status line, in the box to the right of the Insert/Overtype Mode indicator.

When you use the DIR command to create a DIR.DIR file and when you use the MACROS command to create a MACROS.KML file, the LOCKING ON processing is bypassed, since these are normally used as temporary in-memory files.

KEDIT also bypasses lock processing for UNTITLED files.

Timestamp checking

KEDIT has another facility that can help prevent inadvertent changes to a file by multiple users. Whenever you start editing a file, KEDIT makes a record of the timestamp of the file. This is the date and time that the file was last written to, the same date and time displayed by the DIR command. Whenever you attempt to save a file, KEDIT compares its internal record of the timestamp of the file with the actual timestamp on the disk file. If the two timestamps disagree, it is likely that some other user or program has changed the disk file since you began editing it, and you might be about to inadvertently overwrite these changes.

SET TIMECHECK controls KEDIT's timestamp checking. If TIMECHECK ON, the default, is in effect, KEDIT gives you a warning when this situation occurs. If you decide that you want to go ahead with your changes to the file, you can tell the dialog box displayed by File Save to save the file anyway. You can also use the FFILE or SSAVE commands to bypass the warning and overwrite the existing file. With TIMECHECK OFF, this warning is never issued.

12.2 File Formats

To understand the file formats that KEDIT can process and how KEDIT handles special characters (for example, tab characters and end-of-file characters), you should be

aware of how KEDIT processes your file in each of three phases: reading a file from disk into memory, editing a file in memory, and writing a file from memory to disk.

12.2.1 Reading a File from Disk

Here are the rules followed by KEDIT when it reads a file to be edited into memory:

Files must consist of a series of lines. SET EOLIN controls what characters signal the end of a line. By default, KEDIT treats either a carriage return (character code 13) optionally followed by a linefeed (character code 10) or a linefeed character alone as signalling the end of a line. With SET EOLIN you can tell KEDIT to treat only a carriage return (optionally followed by a linefeed) or only a linefeed (optionally preceded by a carriage return) as signalling the end of a line. (You can also use EOLIN NONE, a special case discussed in Section 12.2.4, "EOLIN NONE and EOLOUT NONE".)

KEDIT cannot properly process lines that are longer than the WIDTH value. WIDTH is controlled by the SET INITIALWIDTH command and by the WIDTH initialization option. The default WIDTH is 10000, but you can set it as high as 999999. The WIDTH value cannot be changed once your KEDIT session has begun.

The last line of a file can optionally be followed by one or more end-of-file (character code 26) characters.

If EOFIN ALLOW is in effect, as it is by default, files can contain embedded end-of-file characters. If EOFIN PREVENT is in effect, the first end-of-file character encountered is taken as the end of the file.

Null characters (character code 0) are allowed in files and are handled like any other character.

If TABSIN ON is in effect, tab characters (character code 9) are expanded as files are read in to sequences of blanks, according to standard tab positions 1, 9, 17, etc. (You can use SET TABSIN to change this standard eight-column tab increment). With TABSIN OFF, tab characters are handled like any other character.

If TRAILING ON is in effect, trailing blanks are preserved. With the default of TRAILING OFF, trailing blanks are not preserved, and KEDIT ignores any blanks following the last nonblank character in the lines that it reads in.

If TRANSLATEIN OEMTOANSI is in effect, KEDIT converts your file, as it is read in, from the OEM character set to the ANSI character set. See Section 3.7.2, "Converting between OEM and ANSI", for a discussion of character set conversion issues.

LRECL and RECFM settings affect how a file is written to disk. Except when EOLIN NONE is in effect (this special case is discussed in Section 12.2.4, "EOLIN NONE and EOLOUT NONE"), RECFM and LRECL have no effect on how a file is read in.

Notes

Some observations based on these rules:

KEDIT cannot properly process files that are in binary format. You cannot, for example, directly edit Excel spreadsheets or graphics files with KEDIT.

Many word processors produce files whose lines don't end with carriage return or carriage return-linefeed pairs. With these word processors, carriage returns might appear only at the end of a paragraph, or carriage returns with the high bit set (character code 141) can end lines. KEDIT cannot properly process these files.

Many database programs produce files with fixed length records not separated by carriage returns or carriage return-linefeed pairs; KEDIT's normal file processing does not work with these files. You might be able to process these files by using EOLIN NONE and EOLOUT NONE, as discussed below in Section 12.2.4, "EOLIN NONE and EOLOUT NONE".

Files with lines longer than the WIDTH value can be read in, but the longer lines will not be handled properly by KEDIT; they will either be split or truncated as they are read in. It can occasionally be useful to view such a file with KEDIT, but such files should not be written back to disk, since these long lines will be split in undesirable places and possibly changed unpredictably.

12.2.2 Editing a File

Here are some things you should know about how KEDIT handles your file while it is being edited:

Some editors handle files internally as sequences of characters, with end-of-line characters that can be added, deleted, or changed like any other character. KEDIT instead treats files as sequences of lines. Files are stored internally as a doubly-linked list of lines. Line endings are implicit; carriage-return and linefeed characters are not actually used internally to mark the ends of lines.

While you are editing a file, you can enter and manipulate any of the 256 possible characters, including null characters, tab characters, carriage return characters, linefeed characters, and end-of-file characters.

Carriage return and linefeed characters are treated like any other characters while a file is being edited. Inserting or deleting these characters does not cause lines to be split or joined; the SPLIT, JOIN, and SPLTJOIN commands handle these operations.

Some editors give special handling to tab characters; KEDIT does not. "Tabbing" with KEDIT (usually assigned to the Tab and F4 keys) simply moves the cursor to the next tab column, and does not insert a tab character into your file.

SET EOFIN and SET EOFOUT affect end-of-file character processing when a file is read in from disk or written out to disk. They have no effect on the editing of files in memory, where end-of-file characters are treated like any other character.

If TRAILING ON is in effect, KEDIT preserves any trailing blanks at the end of a line, and keeps track of any trailing blanks that you add to a line or remove from a line during your editing session. With the default of TRAILING OFF, KEDIT does not keep track of the number of trailing blanks at the end of a line; KEDIT internally stores text up to the last nonblank character of a line, and does not store trailing blanks.

You cannot change or manipulate text beyond the truncation column of a line. The truncation column, controlled by SET TRUNC, is normally set equal to the WIDTH setting.

12.2.3 Writing a File to Disk

Here are the rules that KEDIT follows when writing your file to disk:

Any carriage returns or linefeeds within a line of your file are written to disk like any other character. Note that if KEDIT reads such a file back in again, KEDIT will usually interpret any carriage returns or linefeeds as marking the end of a line.

After each line is written to disk, KEDIT adds an end-of-line sequence, which is determined by SET EOLOUT and is normally a carriage return-linefeed pair. You can also use SET EOLOUT to specify that only a linefeed be written after each line, or only a carriage return, or (in a special case discussed below in Section 12.2.4, "EOLIN NONE and EOLOUT NONE") that no end-of-line sequence should be written.

A special case is the character sequence written after the last line of your file, which is determined by SET EOFOUT . This sequence is usually the end-of-line sequence determined by SET EOLOUT. But you can use SET EOFOUT to specify that an end-of-file character should be added to this sequence, that only an end-of-file character should be written, or that no characters at all should be written.

If TRAILING ON is in effect, any trailing blanks in your file are written to disk. With the default of TRAILING OFF, trailing blanks are not preserved, and KEDIT ignores any blanks following the last nonblank character in the lines that it writes to disk.

If TRANSLATEOUT ANSITOOEM is in effect, KEDIT converts your file, as it is written out, from the ANSI character set to the OEM character set. See Section 3.7.2, "Converting between OEM and ANSI", for a discussion of character set conversion issues.

If TABSOUT ON is in effect, sequences of blanks within a line are compressed to tabs according to standard tab settings 1, 9, 17, etc. (You can use SET TABSIN to change this standard eight-column tab increment).

If RECFM VARYING, the default, is in effect, text through the last character of each line is written to disk. A trailing blank might be added, depending on the setting of TRAILING. Any lines longer than the LRECL setting are truncated with-

out warning to the LRECL setting before being written to disk. By default, LRECL is set equal to the WIDTH value.

If RECFM FIXED is in effect, all lines shorter than the logical record length (LRECL) setting are padded with blanks up to the LRECL setting, and lines longer than the LRECL setting are truncated without warning to the LRECL setting before being written to disk.

12.2.4 EOLIN NONE and EOLOUT NONE

KEDIT was designed to work with text files that are organized into lines, with an end-of-line sequence (such as a carriage return-linefeed) marking the end of each line. You might occasionally need to work with files that are instead organized into fixed-length records with no explicit end-of-line sequences. For example, you might have a data file that is 8000 bytes long, containing 100 records with 80 bytes of data each. EOLIN NONE and EOLOUT NONE are intended to help in special situations like these.

With EOLIN NONE, KEDIT does not look for end-of-line sequences as it reads in your file. Instead, it reads your file as a series of fixed-length records, using the LRECL setting to determine how many bytes make up each record.

With EOLOUT NONE, KEDIT does not add end-of-line sequences to the lines that it writes to disk.

Usage

It is somewhat tricky to use EOLIN NONE and EOLOUT NONE properly, because they are usually used with a special alternate profile, and because they require that several related KEDIT options also be set. For these reasons, we recommend that EOLIN NONE and EOLOUT NONE be used only by advanced KEDIT users. We also recommend that you use them first with test files, to be sure you understand how they work and that you do not inadvertently change the formats of the files you work with.

To have an effect when a file is read in, EOLIN NONE must be set before the file is read in, so it is normally issued from a profile. Since you would not want to put EOLIN NONE in effect in your standard profile, it is usually issued from an alternate profile. When you set EOLIN NONE, you also need to set LRECL to the record length of the file you will work with. You should also be sure that TABSIN OFF and EOFIN ALLOW are in effect, so that tab characters and end-of-file characters do not receive special treatment as the file is read in.

To use EOLOUT NONE, you also need to set LRECL, but the LRECL value used to read the file in with EOLIN NONE is usually the same value you want to use when writing the file out. You also need to set RECFM FIXED so that KEDIT will write your data to disk as fixed length records (each record LRECL bytes long, with no end-of-line sequence), and you want EOFOUT EOL and TABSOUT OFF in effect, to be sure that no extra end-of-file character is written and that blanks aren't compressed to tabs.

The alternate profile that you would use to edit a file with no end-of-line sequences that has, for example, 80-byte records would look something like the following. (See Chapter 9, "Tailoring KEDIT", for discussion of profiles and alternate profiles.)

```
'set reprofile on'
'set eolin none'
'set lrecl 80'
'set tabsin off'
'set eofin allow'
'set eolout none'
'set recfm fixed'
'set tabsout off'
'set eofout none'
```

If this profile was stored on disk as EOLNONE.KEX, you could use this KEDIT command to edit your file:

```
kedit data.fil (profile eolnone
```

12.2.5 TABSAVE

SET TABSAVE is a specialized command that deals with an issue affecting a small number of KEDIT users.

KEDIT's TABSIN/TABSOUT processing can sometimes lead an "unchanged" line to be written back to disk differently than when it was read in – existing tabs can be replaced with spaces, or vice versa. Some version control systems undesirably see these all as "changed" lines.

TABSAVE ON avoids this by checking to see whether the current line is subject to the problem and, if so, by saving an exact copy of the character sequence of the original line. Later, when the file is saved, and an "unchanged" line is about to be written back to disk, KEDIT doesn't write the line out in the normal way, but instead writes back its saved copy of the exact original version of the line.

See the discussion of SET TABSAVE in the Reference Manual for full details on SET TABSAVE.

12.3 Long Filenames

Here are some notes about the support provided by KEDIT for the long filenames available with all current versions of Windows. (In Windows 3.1 and earlier, and under DOS, file names were limited to a maximum of 8 characters and file extensions to a maximum of 3 characters.)

When you use filenames that contain blanks or parentheses in commands issued from a macro or from the command line, you will need to enclose these names in double quotes. For example, from the KEDIT command line:

```
KEDIT "A name with blanks"
```

SET FCASE controls whether KEDIT handles all fileids internally in lowercase or keeps them in mixed case, as is necessary to create and preserve mixed case fileids.

With the default of FCASE ASIS (''as is''), KEDIT displays filenames in the same case (upper, lower, or mixed) that the names have on disk, and creates new files using exactly the combination of upper- and lowercase characters that you specify. An exception comes within DIR.DIR files, where names that are in lowercase or mixed case are displayed as is, but names that are in uppercase are displayed in lowercase, since this is generally easier to read.

With FCASE LOWER, KEDIT displays all filenames in lowercase in DIR.DIR files and on the ID line, and in uppercase on the title bar. New files are created with lowercase names, regardless of the case in which you enter the name.

Whenever a disk file with a long name is created, Windows also creates a short DOS-compatible name for the file, so that the file can be accessed by older programs that can't handle long names. KEDIT uses only the long names for these files, and doesn't keep track of or display the short name. You can specify the short name for a file that you want to work with, but KEDIT will automatically convert the short name to the long name. For example, consider a file on your disk called

```
C:\files\A long name.txt
```

with which Windows has associated the DOS-compatible name

```
C:\files\along~1.txt
```

You can begin editing the file with the command

```
KEDIT "C:\files\A long name.txt"
```

or with the command

```
C:\files\along~1.txt
```

In either case, assuming the default of FCASE ASIS is in effect, KEDIT will display the fileid on the title bar as

```
C:\files\A long name.txt
```

You can enter filenames in lowercase, uppercase, or mixed case. When searching for a file on disk, KEDIT does a case insensitive comparison of the fileid, so if you have a file on disk called, for example,

D:\Sample.TxT

you could begin editing the file by using any of these commands:

```
KEDIT D:\Sample.Txt
```

```
KEDIT D:\SAMPLE.TXT
```

```
KEDIT D:\sAMPLE.tXT
```

With the default of FCASE ASIS in effect, KEDIT will display all fileids using the case in which they actually exist on disk. So in the preceding example, the fileid would be displayed on the title bar as

```
D:\Sample.TxT
```

regardless of the command you use to begin editing the file.

The case that you use to enter a fileid does matter when you are editing a new file or changing the name of an existing file. With the default of FCASE ASIS in effect, KEDIT will save the file to disk using the exact name that you entered, with the same combination of upper- and lowercase letters.

The display of long filenames in DIR.DIR files is affected by the SET DIRFORMAT command, whose first two operands control the amount of space set aside in DIR.DIR files for file names and for file extensions. (The third operand of SET DIRFORMAT controls whether KEDIT uses 2 or 4 digits for the year in the date field of DIR.DIR files.) By default, KEDIT sets aside 30 columns in DIR.DIR files for filenames and 10 columns for file extensions. This corresponds to the command

```
SET DIRFORMAT 30 10
```

As a special case, you can specify 0 as the value for file extensions. This causes KEDIT to display the name and extension together as a unit in the columns normally set aside for the file name.

Appendix A. XEDIT Compatibility

KEDIT supports many of the most commonly used commands and features of XEDIT, the mainframe text editor used with IBM's VM/CMS operating system. Not all XEDIT features are available in KEDIT, and there are a number of differences, both in default settings and how some commands are implemented. This chapter reviews some of the major differences.

Quick Start

Here are some things that XEDIT users immediately ask about when moving to KEDIT:

The prefix area and scale line are off by default. To turn them on, select SET Command from the Options menu and, in the resulting dialog box, select the PREFIX option and turn it on, and then select the SCALE option and enable the scale line. Then select Save Settings from the Options menu and press the Save button in the resulting dialog box to make your new settings take effect in future editing sessions.

KEDIT's keyboard behavior takes a little getting used to. With KEDIT's default interface settings, the keyboard behavior is much like that of other Windows applications. Here are the keys that you can use for some common XEDIT-related tasks:

To move the cursor from the file area to the command line, use the F12 key. Any pending prefix commands will also be executed when you press F12.

To add new lines to the file, press the Enter key while the cursor is in the file area.

To execute a command that you have typed on the command line, press the Enter key while the cursor is on the command line.

To move to the beginning of the next line of the file area, press Ctrl+Enter.

KEDIT and XEDIT

Here are some of the reasons for the differences between KEDIT and XEDIT:

KEDIT and XEDIT run on different hardware. KEDIT can act on each keystroke you enter rather than processing an entire screen of data at a time, as XEDIT does. Many common editing operations, such as deleting or adding lines, are assigned to single keys or key combinations in KEDIT. The PC allows KEDIT to implement automatic scrolling and other features not available on a 3270 terminal. So KEDIT's display and keyboard handling differ from XEDIT's.

As a Windows application, KEDIT's visual appearance is quite different from XEDIT's. For example, KEDIT displays files in overlapping, resizable windows, while XEDIT uses tiled, non-overlapping windows. Much of your interaction with KEDIT is through menus, dialog boxes, scroll bars, and other aspects of the Windows user interface that don't directly correspond to elements of the XEDIT user interface.

The underlying operating systems used by KEDIT and XEDIT are different. For example, KEDIT and XEDIT use different file naming conventions, because DOS fileids have different rules than CMS fileids.

Overview of differences

Here are some of the major differences between KEDIT and XEDIT:

The keys on the PC keyboard do not always have a direct counterpart on the 3270 keyboard. Consequently, some default key assignments differ between KEDIT and XEDIT. For example, the 3270 terminal has an Enter key that, when pressed, executes any pending prefix commands, executes any commands on the command line and puts the cursor on the command line. A separate new-line key moves the cursor to the beginning of the next line on the screen. The PC keyboard has no separate new-line key. (But see the SET RIGHTCTRL command, which lets you use the right Ctrl key as an Enter key.)

By default, the Enter key adds new lines to your file when the cursor is in the file area and behaves like XEDIT's Enter key when the cursor is on the command line. Use Ctrl+Enter to move the cursor to the next line of the file area, and use the F12 key to move the cursor to the command line and to execute pending prefix commands.

Default settings of some function keys are different in KEDIT than in XEDIT. KEDIT uses the Page Up and Page Down keys to page backward and forward in your file, and so has other functions assigned to F7 and F8. The default assignments for F5, F10, and F12 are also different from XEDIT's. The SET PF command is not supported in KEDIT; instead, keys are redefined via the DEFINE command.

The default settings of several SET options are different in KEDIT than in XEDIT. KEDIT uses PREFIX OFF, STAY ON, SCALE OFF, and CASE MIXED IGNORE. XEDIT uses PREFIX ON, STAY OFF, SCALE ON, and the setting of CASE depends on the type of file you are editing.

The ID line, controlled in KEDIT by the SET IDLINE command, is normally displayed in XEDIT but is off by default in KEDIT.

To use KEDIT's equivalent of XEDIT's Input Mode, you must first issue the KEDIT command SET INPUTMODE FULL.

KEDIT has no Powerinput Mode. However, entering Input Mode with INPUTMODE FULL and WORDWRAP ON provides an approximation of Powerinput Mode.

In XEDIT, it is possible to move the cursor to any part of the screen. In KEDIT, it is impossible to position the cursor on certain protected areas of your document windows, such as reserved lines, the command line arrow, and the area above the top-of-file line or below the end-of-file line.

In XEDIT there are question mark (?) buffers for each view of a file. In KEDIT there is one global question mark buffer shared by all files.

FILE, SAVE, FFILE, and SSAVE are handled as synonyms in XEDIT, but are commands in KEDIT.

Several of KEDIT's commands, such as ALL, ALTER, and the X and S prefix commands, are implemented in XEDIT as macros.

KEDIT's handling of SCHANGE differs from XEDIT's.

KEDIT does not have support for XEDIT's UPDATE facility.

KEDIT cannot display long lines on multiple lines of your display, as XEDIT can. KEDIT always displays a line of your file on a single line of the display.

In XEDIT, your profile is called PROFILE XEDIT. In KEDIT, it is called WINPROF.KEX. Your profile is somewhat less important in KEDIT than in XEDIT, because many SET options can be controlled through the Options SET Command dialog box, and their values saved for use in future editing sessions via the Options Save Settings dialog box.

VMPROF.KEX

A KEDIT macro called VMPROF.KEX is normally installed by KEDIT's setup program in KEDITW's SAMPLES subdirectory. This macro redefines many keys to make KEDIT act more like XEDIT. Note that in most cases you will probably not want to use all of these key assignments, because KEDIT's default assignments, though different from XEDIT's, are quite useful on a PC. For example, in KEDIT the Cursor Up and Cursor Down keys cause automatic vertical scrolling, a powerful feature not available with XEDIT. In XEDIT, the Cursor Up and Cursor Down keys "wrap" around the top and bottom of the screen, respectively.

You might want to modify VMPROF.KEX to obtain the level of XEDIT compatibility you are most comfortable with. To use VMPROF.KEX, or your modification of it, you will need to rename it to WINPROF.KEX. WINPROF.KEX is normally kept in the "KEDIT Macros" subdirectory of your Windows Documents folder, which is sometimes known as the My Documents folder.

Macro facility differences

These are some of the differences between the KEDIT and XEDIT macro facilities:

XEDIT macros are written in EXEC 2 or in REXX, macro languages that are available in CMS. KEDIT macros are written in KEXX, which is a subset of the REXX language and is built into KEDIT.

Several facilities typically used in KEDIT macros are not available in XEDIT, including KEDIT's Boolean functions, implied EXTRACT functions, and the EDITV command.

Commands issued from KEDIT act relative to the *focus line*, rather than the current line. The focus line is the same as the current line when the cursor is on the command line, but when the cursor is in the file area, the focus line is the line on which the cursor is located. XEDIT commands always act relative to the current line. See Section 6.5, "The Focus Line", for a discussion of the focus line concept.

KEDIT does not support prefix macros.

KEDIT does not support the XEDIT READ and SET CTLCHAR commands used for defining input screens and obtaining user input with a macro. KEDIT's closest equivalents are the READV KEY command, which lets a macro read a single keystroke, and the DIALOG command, which displays and receives input from a dialog box.

Commands issued from KEDIT macros are not subject to LINEND processing, do not normally affect the contents of the = buffer, and, unless they are issued via KEDIT's SYNEX command, are not subject to synonym processing.

Appendix B. Glossary

= buffer Buffer that holds the last command entered on the command line. The command in the = buffer will be re-executed when you enter the = command.

? buffers Buffers that hold the 200 lines most recently entered from the command line. These commands can be redisplayed on the command line by successive entries of the ? command or by pressing F6. You can cycle through these commands with Ctrl+Cursor Up and Ctrl+Cursor Down. This is also referred to as KEDIT's ‘‘command history’’ feature.

.AUS File extension used for temporary files written to disk by KEDIT’s AUTOSAVE facility.

alteration count Count of changes made to your file during an editing session. Two numbers are given: the number of changes to your file since KEDIT’s AUTOSAVE facility last saved it to disk, and the number of changes to your file since the last time it was saved to disk because you issued the SAVE command. Displayed as the first two numbers that follow “Alt=” on the status line; the third number following Alt= is the undo count.

alternate profile A profile macro, specified via the PROFILE option on the KEDIT command line, executed instead of the default profile.

Alt+numeric keypad Characters that don’t appear on the keyboard can often be entered using the Alt key in combination with the keys on the numeric key pad. (For more details, see Section 4.6, “Entering Special Characters”.)

ANSI character set Character set used by most Windows fonts and by most Windows applications. KEDIT uses the ANSI character set in all of its dialog boxes, and it is used by default within KEDIT’s document windows. ANSI stands for the American National Standards Institute, which originally defined the character set. The main alternative is the OEM character set, which is the character set used by DOS.

ANSI font A font that uses the ANSI character set. Most Windows fonts, including KEDIT’s default fonts, are ANSI fonts.

ARBCHAR Wildcard characters used in string searches when ARBCHAR ON is in effect. Usually, ‘‘$’’ matches zero or more characters and ‘‘?’’ matches exactly one character.

autosave Automatic save of your file to a temporary disk file with an extension of .AUS after a specified number of changes have been made to the file. Controlled by SET AUTOSAVE.

.BAK File extension used for backup files created when files are written to disk. Creation of backup files is controlled by SET BACKUP.

block A marked area of your file that can be operated on as a unit by cut-and-paste operations and by such commands as COPY and MOVE. KEDIT supports both *persistent blocks*

(which remain marked until you explicitly unmark them) and *non-persistent blocks* (which are unmarked as soon as you move the cursor and which are often referred to as *selections*). Three different ''shapes'' are available: *line blocks*, *box blocks*, and *stream blocks*. (For more details, see Section 3.3, "Blocks and Selections".)

block target Marked block used as the operand of a command that takes target operands, making the marked block the target area for the command.

bookmark Named line target used to set and return to a specific point in the file, controlled via the Actions Bookmark dialog box and by the SET POINT command.

box block A KEDIT block involving a rectangular area of your file. Marked via the Alt+B key combination, by dragging with Alt+mouse button 1 with INTERFACE CUA in effect, or by dragging with both mouse buttons down with INTERFACE CLASSIC in effect.

box selection A non-persistent box block.

change bit One of the flag bits associated with a line. A line's change bit is set when the line is changed during a KEDIT session. Also controlled by the SET LINEFLAG command.

character code The numeric value used to encode a given character in a particular character set. For example, the British pound symbol has the character code 156 in the OEM character set and the character code 163 in the ANSI character set, while a lowercase ''a'' has the character code 61 in both the OEM and ANSI character sets.

character set A set of characters including alphanumerics, punctuation, and other symbols, along with an assignment of a character code to each of those characters. KEDIT for Windows normally uses the ANSI character set, but can also use the OEM character set; for both of these character sets, the character codes are in the range 0 to 255.

CLASSIC interface One of the two sets of keyboard and mouse conventions supported by KEDIT for Windows; the other is the CUA interface. With the CLASSIC interface, most keyboard and mouse behavior in KEDIT for Windows is compatible with the behavior of earlier text mode versions of KEDIT, even where this means that KEDIT for Windows behaves differently than other Windows applications.

click To press down one of the buttons on a mouse and then immediately release it.

clipboard A temporary storage area into which KEDIT and other applications can place data used in cut-and-paste operations. The data can then be pasted into a document that you are editing with KEDIT, or can be transferred to another application.

column pointer Imaginary pointer to the current column, used as a starting point for column commands such as CDELETE and CLOCATE when these commands are issued from the command line.

column target Special target used by CDELETE and CINSERT commands. (See also Section 6.4, "Column Targets")

command Instruction to KEDIT to perform a specific operation. You can issue commands from the KEDIT command line or from macros.

command line The line, usually beginning with an arrow ("====>"), on which KEDIT commands are entered.

command line selection A portion of the text on the command line, selected by dragging with mouse button 1 or by using Shift+cursor keys, that can be used in cut-and-paste operations. Command line selections are available only if INTERFACE CUA is in effect.

CUA interface The CUA (Common User Access) interface is a set of conventions, originally established by IBM, for how graphical PC applications should work. When the default of INTERFACE CUA is in effect, KEDIT for Windows uses a variant of the CUA interface, adapted from Microsoft's ''The Windows Interface: An Application Design Guide'' and from applications like Microsoft's Word for Windows. With the alternative of INTERFACE CLASSIC, KEDIT for Windows instead uses many of the conventions of earlier text mode versions of KEDIT.

current column Column of the current line relative to which column commands issued from the command line operate. (See also *focus column*)

current directory The directory that KEDIT uses by default for certain operations. For example, if you use the KEDIT command with a fileid that does not include a drive or path specification, the first place that KEDIT looks for the file is the current directory. The current directory is controlled by the File Directory dialog box and by the CHDIR command.

current file File in which the cursor is currently located.

current line The line of the file, normally displayed in the middle of the document window, that is the starting point for most commands issued from the command line. When the cursor is on the command line, KEDIT normally draws a box around the current line to emphasize its location. (See also *focus line*)

current window Document window in which the cursor is currently located.

cursor column Column in which the cursor is currently located.

cursor field Field in which the cursor is currently located, which can be either a line of the file, the prefix area of a line, or the command line.

cursor line Line of the file in which the cursor is currently located.

default profile Fileid of default profile run by KEDIT for Windows. Usually WINPROF.KEX, but this can be changed via the DEFPROFILE initialization option.

dialog box Pop-up box that presents information to, and optionally obtains input from, the user.

DIR.DIR file Temporary file created by KEDIT when you issue the DIR command. The DIR.DIR file displays information about the names, sizes, and dates of files that you specify via the DIR command.

document window Window used to display the contents of a file that you are editing. Several document windows may be present if you are editing several documents at a time, or have created additional windows to display multiple views of a file. Document windows are always displayed within the boundaries of KEDIT's *frame window*. KEDIT's document windows include a file area, with data from your file, a command line, and optional prefix area and scroll bars.

double-click To press down and release one of the buttons on a mouse twice in rapid succession.

drag To press down one of the buttons on a mouse and, without releasing the button, move the mouse.

end-of-file character Character code 26. Sometimes found as the last character of DOS files. SET EOFIN and SET EOFOUT control KEDIT's handling of the end-of-file character.

end-of-file line Imaginary line located just beyond the last line of your file.

end-of-line sequence Sequence of characters, normally a carriage return character and a linefeed character, that mark the end of each line of a disk file. SET EOLIN and SET EOLOUT control KEDIT's handling of end-of-line sequences.

end-of-range line Imaginary line located just below the last line in the range. Displayed only after you have issued the SET RANGE command.

excluded line Text line present in your file but temporarily not displayed due to use of selective editing commands like the ALL command or the X prefix command. Usually its place in the file is indicated on the screen by a shadow line.

field An area of the document window to which you can move the cursor and whose contents can be edited as a unit. The command line is a field, as is each line of the file area and each line of the prefix area.

file area The portion of the document window used for display of the contents of your file.

file locking KEDIT facility, controlled by SET LOCKING and related commands, in which files you edit are kept open throughout the editing session, to prevent access to or changes to the files by other users on your network or other processes on your own workstation.

fileid The filename and extension, and optionally drive and path specifier, used to refer to disk files.

fixed-pitch font A font in which all characters have the same width, as opposed to a *proportional font*, in which the width of the characters can vary. KEDIT uses only fixed-pitch fonts to display text in document windows.

flag bit Associated with each line in your file are three flag bits: the new bit, the change bit, and the tag bit. Line class targets can refer to lines whose flag bits are set a certain way, and the highlighting facility refers to these bits when highlighting altered or tagged lines.

focus column Column relative to which KEDIT's column commands (CLOCATE, CINSERT, etc.) operate. When the cursor is on the command line, the column pointer column is the

focus column. When the cursor is in the file area, the cursor column is the focus column.

focus line The line of the file relative to which KEDIT commands operate. When the cursor is on the command line, the current line is the focus line. When the cursor is in the file area, the cursor line is the focus line. (See also Section 6.5, "The Focus Line")

font A graphical representation of a character set, in a given size and style. Fonts can be fixed-pitch (monospaced), in which each character has the same width, or proportional, in which different characters have different widths. KEDIT uses only fixed-pitch fonts to display text in document windows.

frame window The main window for a KEDIT session, which can contain one or more *document windows*.

FSA File Storage Area. The area of Windows memory used by KEDIT to hold the contents of the files you are editing, as well as control information for each line of your files, your macro definitions, and the values of KEXX and EDITV variables.

highlighting facility KEDIT facility, controlled by the SET HIGHLIGHT command and the TAG command, that lets you highlight certain lines of your file on the screen depending on their contents, their selection level, or whether they have been altered.

ID line Optional identification line at the top of a document window with information about the file in that window: the fileid, the cursor line and column, the file size, and alteration and undo counts. The ID line is normally not used in KEDIT for Windows, because all of the information it contains is available through the status line and the document window's title bar.

initialization options Options specified via the KEDIT environment variable or on the KEDIT command line. They specify such things as the maximum width that KEDIT will handle and alternate profile macros. (See also Reference Manual Chapter 2, "Invoking KEDIT")

INTERFACE CLASSIC When INTERFACE CLASSIC is in effect, KEDIT uses its CLASSIC interface, in which most of its keyboard and mouse behavior is compatible with the behavior of earlier text mode versions of KEDIT. The alternative is INTERFACE CUA, in which KEDIT's behavior is compatible with that of most other Windows applications. INTERFACE CLASSIC is controlled by the Options Interface dialog box and by the SET INTERFACE command.

INTERFACE CUA When INTERFACE CUA is in effect, KEDIT uses its CUA interface, in which most of its keyboard and mouse behavior is compatible with that of other Windows applications. The alternative is INTERFACE CLASSIC, in which KEDIT's behavior is compatible with that of earlier text mode versions of KEDIT. INTERFACE CUA is controlled by the Options Interface dialog box and by the SET INTERFACE command.

input mode Special mode used for inputting text after the INPUT command is issued with INPUTMODE FULL or INPUTMODE LINE in effect.

Insert Mode Mode in which each character that you type will be inserted at the cursor position, shifting any existing text at the cursor position to the right. Controlled by the Insert key and

indicated on the status line by INS, as opposed to the OVR that indicates Overtype Mode.

ISA Internal Storage Area. Memory area used by KEDIT to hold certain internal control information.

KEDIT Language Definition A KEDIT Language Definition file is a file, which normally has an extension of .KLD, containing the language-specific rules used by the syntax coloring facility to decide which text should be displayed as comments, strings, numbers, etc. .KLD files are loaded via the SET PARSER command.

.KEX Extension used for a file that contains a single KEDIT macro that can be executed with the MACRO command or loaded into memory with the DEFINE command.

KEXX Macro language built into KEDIT. KEXX is a subset of the REXX language.

.KLD KEDIT Language Definition. Extension used for KEDIT Language Definition files, which contain the language-specific rules used by the syntax coloring facility to decide which text should be displayed as comments, strings, numbers, etc. .KLD files are loaded via the SET PARSER command.

.KML KEDIT Macro Library. Extension used for files that contain a number of KEDIT macro definitions which can be loaded into memory as a group with the DEFINE command.

line block A KEDIT block involving consecutive lines of text. Marked via the Alt+L key combination, by dragging with button 1 in the margin area or in the prefix area, by dragging with Ctrl+mouse button 1 with INTERFACE CUA in effect, or by dragging with mouse button 2 with INTERFACE CLASSIC in effect. KEDIT supports both persistent line blocks and (with INTERFACE CUA in effect) non-persistent line blocks, which are also referred to as line selections.

line class target A type of target involving a line identified not according to some string contained in the line, but by some attribute of the line. Line class targets include BLANK, NEW, CHANGED, ALTERED, TAGGED, and SELECT targets.

line selection A non-persistent line block.

linend character Character (normally "#") used to separate multiple commands entered on a single KEDIT command line when LINEND ON is in effect.

logical record length The maximum length of lines written to disk by File Save and by the KEDIT commands FILE, SAVE and PUT. With RECFM FIXED, all lines are padded or truncated to this length. With RECFM VARYING, longer lines are truncated to this length. Controlled by SET LRECL command, initially equal to the WIDTH value.

macro Sequence of instructions that controls a set of actions that you want KEDIT to perform. KEDIT macros are written in the KEXX macro language; they can loop, test conditions, and issue KEDIT commands.

margin area Area between the border of a document window and the start of the first column of text in the document window. Also referred to as the window margin area. When the mouse

is over this area, the mouse pointer changes to an arrow pointing to the upper right. The margin area provides a convenient way to mark line blocks; simply drag with mouse button 1 while the mouse pointer is in the margin area. The margin area is controlled by SET WINMARGIN.

margins The FLOW command and Ctrl+F key combination reformat paragraphs to fit within the left and right margins. The margin columns are also used as boundaries by other commands, such as the LEFTADJUST command, and by the wordwrap facility. The left and right margin columns, along with paragraph indent value, are controlled with the SET MARGINS command.

MDI The MDI (Multiple Document Interface) is a method that KEDIT, and many other Windows applications that work with several files at a time, use to organize output on your display. An MDI application has one frame window, within which one or more document windows are displayed.

message line The message line is the line of the document window where KEDIT messages are displayed. By default, the message line is at the top of the document window, and is used to display a line of your file when there is no message displayed. For display of multiple messages at a time, you can define several message lines, which are then referred to as the message area.

minimal truncation Shortest legal abbreviation of a command, SET option, QUERY or EXTRACT operand, or initialization option.

mouse pointer A visible indicator on your screen of the location at which mouse operations will take place. Moving the mouse on your work surface will move the mouse pointer on your screen.

named line Line to which you have assigned a name with the SET POINT command. The named line can be used as a target in KEDIT commands. Named lines are also controlled via the Edit Bookmark dialog box.

new bit One of the flag bits associated with a line. A line's new bit is set when the line is added to your file during a KEDIT session.

non-persistent block A block that remains marked only until the next cursor movement. Non-persistent blocks are also referred to as selections. Most Windows applications only support non-persistent blocks, but KEDIT also supports persistent blocks, which remain marked until you explicitly unmark them. Non-persistent blocks are only available when INTERFACE CUA is in effect.

OEM character set Character set used by MS-DOS and by most DOS text mode applications. KEDIT supports fonts that use the OEM character set, but the ANSI character set, which is used by Windows itself and by most Windows applications, is generally preferred. The OEM character set varies somewhat from country to country, and the name comes about because the OEM character set is the character set supplied by your OEM (Original Equipment Manufacturer) for use with DOS.

OEM font A font that uses the OEM character set.

Overtype Mode Mode in which each character that you type replaces any existing character at the cursor location. Controlled by the Insert key and indicated on the status line by OVR, as opposed to the INS that indicates Insert Mode.

paragraph A set of lines that is treated as a unit when you reformat text or use a PARAGRAPH target. Paragraphs are normally delimited by blank lines, but this can be controlled with the SET FORMAT command.

parser Code within KEDIT that scans a file that you are editing and determines which text will be treated as keywords, numbers, strings, etc. for the purposes of syntax coloring. Parsers are defined by the SET PARSER command, which associates a parser name with a KEDIT Language Definition file that contains the rules to be used for parsing a particular language.

persistent block A marked area of your file on which you can perform multiple operations, and which remains marked until explicitly unmarked. The alternative is a non-persistent block, also referred to as a selection, which is unmarked as soon as the cursor is repositioned.

pixel The smallest unit that can be individually written to your display screen; each pixel corresponds to an individual ''dot'' on your display. The resolution of PC displays is measured in pixels, and KEDIT is typically used on systems whose resolution is 640 by 480 (which means 640 pixels horizontally by 480 pixels vertically), 800 by 600, 1024 by 768, or 1280 by 1024.

pop-up Dialog box, menu, or help information displayed in a temporary box that overlays part of the screen and is activated by a macro, mouse action or keyboard command.

prefix area An optionally displayed area of the window, provided mainly for XEDIT compatibility, in which prefix commands are entered. (See also Chapter 7, "The Prefix Area")

prefix command Special commands entered in the prefix area, normally executed when the F12 key (with INTERFACE CUA in effect) or the Home key (with INTERFACE CLASSIC) is hit, that act relative to the line in whose prefix area they are entered. (See also Chapter 7, "The Prefix Area")

profile KEDIT macro, normally WINPROF.KEX, that is automatically run at the start of each KEDIT session and, if REPROFILE ON is in effect, whenever a new file is added to the ring.

proportional font A font in which different characters can have different widths, as opposed to a *fixed-pitch font*, in which the width of all characters is the same. KEDIT uses only fixed-pitch fonts to display text in document windows.

Quick Find Toolbar item that displays the most recent search string. You can use the Find Next button to search for the next occurrence of this string, or you can activate the Quick Find item to make changes to the string or to select from a dropdown list of other recent search strings.

range The portion of your file in which KEDIT commands can operate. Normally, the range is the entire file, but you can use the SET RANGE command to specify the starting and ending lines of the portion to which commands will be limited.

regular expression Special type of string target that uses various pattern matching operators to match text. Described in detail in Section 6.6, "Regular Expressions".

remembered operands KEDIT "remembers" the last operands of several KEDIT commands (ALTER, CHANGE, CLOCATE, COUNT, FIND, LOCATE, SCHANGE and TFIND). If you issue these commands with no operands, KEDIT reuses the operands used when these commands were last issued from the command line.

reserved line Non-editable line optionally displayed in a document window whose location, color, and contents are controlled via the SET RESERVED command.

return code Numeric value set by each KEDIT command indicating success or failure of the command. For commands issued from macros, the return code is placed in the variable RC.

REXX General purpose macro language originally developed by IBM. KEXX, a large subset of the REXX language, is built into KEDIT.

ring The set of all files currently being edited by KEDIT.

saved setting The value of a KEDIT SET option that has been saved, by using the Options Save Settings dialog box or by using the Save Setting button of the Options SET Command dialog box, for use in future editing sessions.

scale line Optionally displayed, non-editable line indicating column locations within your file. Controlled by SET SCALE.

scope The specification which controls whether or not excluded lines will be processed by most KEDIT commands. Controlled by SET SCOPE. With SCOPE DISPLAY, excluded lines are not processed by most KEDIT commands. With SCOPE ALL, they are.

selection Selections are non-persistent blocks. That is, they are blocks that remain marked only until the next cursor movement. Most Windows applications only support selections, but KEDIT also supports persistent blocks, which remain marked until you explicitly unmark them. Selections are only available when INTERFACE CUA is in effect.

selection level Number in the range 0 to 255 assigned to each line of the file either directly through the SET SELECT command or indirectly by using the Edit Selective Editing dialog box, the ALL command, or the prefix commands X or S. Lines whose selection level falls outside the range specified by SET DISPLAY are excluded from the display.

selective editing KEDIT facility that lets you work with only a subset of the lines in a file. Most often used to work with those lines in a file that contain a particular string that has been specified via the Edit Selective Editing dialog box or via the ALL command. (See also Chapter 8, "Selective Line Editing and Highlighting")

SET option An option whose value is controlled by the SET command, such as ARBCHAR or ZONE. Most SET options can also be set via the Options SET Command dialog box, and the values of most SET options can be displayed via the Options Status dialog box. (See also Reference Manual Chapter 4, "The SET Command")

shadow line Non-editable line displayed in place of excluded lines if SHADOW ON is in effect.

slider A small box that appears in a scroll bar to indicate your relative position in a file.

status line Optionally displayed non-editable line at the bottom of the frame window giving information about your KEDIT session. The default status line includes your line and column location in the current file, alterations and undo levels for the current file, the size of the current file, the number of files in the ring, the number of document windows, Insert/Overtype Mode status, file locking status, and the time of day. Controlled by SET STATUSLINE.

stream block A KEDIT block involving a stream of consecutive characters spanning one or more lines of your file (typically, a phrase, sentence, or group of sentences that you want to treat as a unit). Marked by the Alt+Z key combination, by using Shift+cursor key combinations with INTERFACE CUA in effect, or by dragging with mouse button 1. KEDIT supports both persistent stream blocks and (with INTERFACE CUA in effect) non-persistent stream blocks, which are also referred to as stream selections.

stream selection A non-persistent stream block.

syntax coloring KEDIT facility that shows different types of text within a file in different colors. When syntax coloring is enabled, a *parser* scans the file, classifying the text of the file into keywords, comments, strings, etc. which are then displayed in the appropriate color. Syntax coloring is controlled by the SET COLORING command.

system menu Menu activated by clicking on an icon at the left of a window's title bar that lets you move, size, or close the window. The frame window's system menu controls the frame window, and closing the frame window closes KEDIT. A document window's system menu controls the document window. If a document window is maximized, its system menu icon appears at the left of the menu bar.

tab column One of the columns, controlled by the SET TABS command, that the cursor can move to when you hit the F4 key or, if the prefix area is not displayed, the Tab and Shift+Tab keys.

tab line Optionally displayed non-editable line indicating the position of each tab column. Controlled by SET TABLINE.

tag bit One of the flag bits associated with a line. The tag bit is most often set via the TAG command, which puts HIGHLIGHT TAGGED in effect, causing tagged lines to be highlighted.

target A way of referring to some location in your file. Types of targets include absolute line number targets, relative line number targets, string targets, named line targets, line class targets, and group targets. (See also Chapter 6, "Targets")

target area The portion of the file on which a command will operate, determined by a target operand of the command. The target area is usually a group of lines but, depending on the command involved, can also be a box or stream block.

target highlighting KEDIT facility, controlled by the SET THIGHLIGHT, in which string targets found by the LOCATE, CLOCATE, and TFIND commands, by the Edit Find and Edit Replace dialog boxes, and by the Quick Find toolbar item are temporarily highlighted on your screen.

text mode Mode of operation, used by most DOS programs (including KEDIT), in which an application is limited to characters from a predefined character set, appearing in fixed positions on the screen. Contrast this with the graphics mode used by Windows applications, which can involve text in different fonts, icons, and other graphic elements. *Text mode KEDIT* refers to versions of KEDIT developed for use as text mode applications running under DOS or OS/2.

timestamp The time and date information associated with a disk file by the operating system. With TIMECHECK ON, KEDIT checks this timestamp information before writing a file to disk, and if the timestamp has changed since you began editing the file, warns you that the file has been changed.

toolbar Row of redefinable, mouse-selectable buttons displayed on the screen when TOOLBAR ON is in effect. The default toolbar buttons let you open, save, or print a file, manipulate DIR.DIR files, etc.

top-of-file line Imaginary line located just above the first line of your file.

top-of-range line Imaginary line located just above the first line in the range. Displayed only after you have issued the SET RANGE command.

truncation column The last column of your file that can be affected by most editing operations. Data that any command attempts to place beyond the truncation column is normally truncated. Controlled by SET TRUNC, initially equal to the WIDTH value.

typing-replaces-selection Convention used by KEDIT, and by many other Windows applications, in which any text that you type while a selection is marked causes the contents of the selection to be deleted, with the text that you type entered into your file in its place. Typing-replaces-selection is only available when INTERFACE CUA is in effect. It only works properly when you are in Insert Mode; if you are in Overtype Mode, typing when a selection is marked will delete the selection, but the new text will replace the text that followed the selection, instead of being inserted in front of it.

undo level A set of changes to a file, normally all changes made to a file by a single keystroke, or by a single command or macro issued from the command line, that is processed as a unit by the UNDO and REDO commands.

Universal Naming Convention (UNC) A naming convention, often used with files stored on a network, in which files are referred to not by a drive letter, path, and filename, but by a server name,

the name of a shared resource on the server, and then a path and filename. UNC names always begin with a pair of backslashes.

```
\\SERVER2\COMMON\SAMPLES\TEST.FIL
```

In this example, SERVER2 is the server name, COMMON is a shared resource on the server, SAMPLES is a subdirectory, and TEST.FIL is a file name and extension.

UNTITLED file A temporary file that KEDIT creates when you begin a KEDIT session without specifying a fileid, or when you use the File New menu item. If you attempt to save an UNTITLED file to disk, KEDIT will ask you to specify a permanent name for the file.

vershift The number of columns by which the displayed text is offset from the current verify setting because automatic horizontal scrolling has moved the window to the left or the right or because you have issued the LEFT or RIGHT commands. The number is positive if the display is offset to the right and negative if the display is offset to the left.

view The portion of the file visible in one window. If the same file is displayed in multiple windows, you have multiple views of the file. Each view can have a different current line, VERIFY setting, ZONE columns, etc.

width Maximum line length that can be handled in a given KEDIT session. The default is 10000, but a width of up to 999999 can be specified using SET INITIALWIDTH or the WIDTH initialization option.

window Rectangular area in which output from some Windows application appears. MDI (Multiple Document Window) applications like KEDIT have a frame window, within which one or more document windows are displayed.

window margin area Another name for the *margin area*. Area between the border of a document window and the start of the first column of text in the document window. When the mouse is over this area, the mouse pointer changes to an arrow pointing to the upper right. The margin area provides a convenient way to mark line blocks; simply drag with mouse button 1 while the mouse pointer is in the margin area. The window margin area is controlled by SET WINMARGIN.

word target A string target in which the string to be located must begin and end at a word boundary.

wordwrap Automatic addition of a new line whenever the text being entered would extend beyond the right margin. Controlled by SET WORDWRAP.

zone The left and right zone columns determine the range of columns to which KEDIT restricts itself when doing string searches and related operations. Controlled by SET ZONE.

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z