KEDIT for Windows Reference Manual
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 Reference Manual. 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 User's Guide 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

1.1 Overview of Documentation

This is the second of two volumes of documentation for KEDIT for Windows, the *KEDIT for Windows Reference Manual*. The first volume is the *KEDIT for Windows User's Guide*.

This Reference Manual has nine chapters:

Chapter 1, "Introduction"

Chapter 2, "Invoking KEDIT"

Chapter 3, "KEDIT Commands"

Chapter 4, "The SET Command"

Chapter 5, "QUERY and EXTRACT"

Chapter 6, "Macro Reference"

Chapter 7, "Built-in Macro Handling"

Chapter 8, "KEDIT Language Definition Files"

Chapter 9, "Error Messages and Return Codes"

The complete details of KEDIT commands and options, including separate chapters on initialization options, SET options, and the QUERY and EXTRACT commands, can be found in this Reference Manual.

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

Chapter 1, "Introduction''

Chapter 2, "Getting Started''

Chapter 3, "Using KEDIT for Windows''

Chapter 4, "Keyboard and Mouse''

Chapter 5, "Menus and Toolbars''

Chapter 6, "Targets''

Chapter 7, "The Prefix Area''

Chapter 8, "Selective Line Editing and Highlighting''

Chapter 9, “Tailoring KEDIT”

Chapter 10, “Using Macros”

Chapter 11, “Sample Macros”

Chapter 12, “File Processing”

Appendix A, “XEDIT Compatibility”

Appendix B, “Glossary”

If you are just starting with KEDIT, you should first work with the User’s Guide. After you have gained some experience with KEDIT, you will most often use this Reference Manual.

1.2 Syntax Conventions

Many KEDIT commands and options can be abbreviated. For example, the DEFINE command can be entered as DEF, DEFI, DEFIN, or DEFINE. The shortest legal abbreviation for a command is known as the command’s *minimal truncation*. The minimal truncation for DEFINE is DEF. Minimal truncations will be indicated in the section where commands and their options are discussed by showing the minimal truncation in uppercase and the rest of the command’s name in lowercase. So the DEFINE command is shown as

```
DEFine
```

When you have a choice between two options, the options will be separated by a vertical bar (“|”). For example, the commands that allow you to set something ON or OFF will show ON and OFF like this:

```
ON|OFF
```

Optional operands are given in brackets, and operands for which you must substitute an appropriate value are given in italics. For example, the ADD command (which has A as its minimal truncation) can take a number *n* as its operand or can have no operands, in which case one line is added to your file. This is shown as

```
Add [n]
```

Be sure not to actually enter the brackets and vertical bars that appear in descriptions of command syntax. They are not part of the actual command and are only used to help describe the command. While uppercase and lowercase are used in command descriptions to indicate minimal truncations, you can actually enter commands in any combination of uppercase and lowercase.

In examples where it is important to indicate the presence of a specific number of blanks (for example, in the description of the JOIN command), the manual sometimes uses the symbol “␣” to represent a blank.

When a complete indication of the syntax of a command according to the conventions given here would be too complicated, a simplified version of the syntax is given and the text explains the details.

Menu items and toolbar buttons are referred to by using initial capital letters on the words involved. For example, the text refers to the Edit Find dialog box and the Find Next toolbar button.

Chapter 2. Invoking KEDIT

2.1 Running KEDITW32.EXE

To use KEDIT for Windows, you need to run KEDITW32.EXE module. You normally do this by double-clicking on the KEDIT for Windows program icon. But you can also run KEDIT via the Windows Start menu, or you can enter the command yourself from within a Command Prompt window.

The KEDITW32 command is usually issued with no parameters or options. When this happens, KEDIT for Windows starts by displaying a new file called UNTITLED.1. You then either add text to this file or, more often, use File Open to select some existing file to edit.

It is also possible to have KEDIT for Windows start by editing one or more files specified on the command line or to start by displaying a directory listing in a DIR.DIR file. And you can also specify command line options that control certain aspects of the upcoming editing session.

Here are the different forms of the KEDITW32 command:

Editing a file

```
KEDITW32 [fileid ...] [(options [)]]
```

KEDITW32 Tells Windows to load in and execute the KEDITW32.EXE module.

fileid … Tells KEDIT the name of the file or files that you want to edit. If you don't specify a *fileid*, KEDIT will start by editing a new file called UNTITLED.1.

options Lets you specify special initialization options, discussed in Section 2.2, "KEDIT Initialization Options", that will affect this KEDIT session. A left parenthesis must separate the *fileid* specification from the *options* specifications, which can optionally be followed by a right parenthesis.

Examples

```
KEDITW32
```

Start KEDIT for Windows. Since you specified no *fileid*, KEDIT will begin by editing a new file called UNTITLED.1.

```
KEDITW32 TEST.C
```

Start KEDIT for Windows, telling it that you want to edit the file TEST.C.

```
KEDITW32 DATA1.TXT DATA2.TXT
```

Start KEDIT for Windows, telling it that you want to edit the files DATA1.TXT and DATA2.TXT.

```
KEDITW32 C:\TEST\*.TXT
```

Start KEDIT for Windows, telling it that you want to edit all files in the C:\TEST directory with an extension of .TXT.

```
KEDITW32 TARGET.TXT (PROFILE ALTPROF
```

Start KEDIT for Windows, telling it that you want to edit the file TARGET.TXT and that, instead of automatically executing your WINPROF.KEX file it should use ALTPROF.KEX as your profile.

Viewing a directory

KEDITW32 DIR [*filespec* ...] [(*options* [)]]

KEDITW32 Tells Windows to load in and execute the KEDITW32.EXE module.

DIR Tells KEDIT that you want to start out with a directory display. KEDIT will display the directory listing in a file called DIR.DIR, just as it does when you issue the KEDIT DIR command.

filespec … Specifies the files to be included in the directory display. File specifications can use the usual wildcard characters, asterisk (“*”) and question mark (“?”). If no *filespec* is given, KEDIT lists all files in the current directory, as discussed in the first example below.

options Lets you specify initialization options, as discussed in Section 2.2, “KEDIT Initialization Options”.

Examples

```
KEDITW32 DIR
```

Start KEDIT with a directory listing of all files in the current directory. (When KEDIT for Windows starts up, the current directory is normally your Windows Documents or My Documents folder. If you invoke KEDIT via a desktop shortcut, you can override this by specifying a different “Start In” directory in the shortcut’s Properties dialog box. SET INITIALDIR also affects KEDIT for Windows’ initial current directory.)

```
KEDITW32 DIR *.C *.H
```

Start KEDIT with a directory listing of all files in the current directory that have .C or .H as their extension.

```
KEDITW32 DIR D:\*.* (PROFILE DIRPROF
```

Start KEDIT with a directory listing of all files in the root directory of the D: drive, and tell KEDIT that instead of automatically executing your WINPROF.KEX file, it should use DIRPROF.KEX as your profile.

2.2 KEDIT Initialization Options

The options that you specify on the KEDITW or KEDITW32 command line used to start a KEDIT for Windows session are known as *initialization options*, because their primary use is to control some aspects of how KEDIT initializes itself at the start of an editing session. For example, there is a NOREG option that tells KEDIT not to process configuration information saved in the Windows registry, and a NOPROFILE option that tells KEDIT to bypass execution of your profile.

Several of the options discussed here can also be used from within a KEDIT for Windows session, when you use the KEDIT command to add additional files to the ring. For example, if REPROFILE ON is in effect, KEDIT normally re-executes your profile whenever a new file is added to the ring, but you can override this by using the NOPROFILE option on the KEDIT command line used to add an additional file to the ring.

The initialization options that you can specify are:

COLumn *m* The COLUMN option tells KEDIT that, as soon as the file is loaded, the cursor should be moved into the file area to the specified column of the current line. The current line is initially set to the line specified via the LINE option, or to the Top-of-File line if the LINE option is not used. An example:

```
KEDITW32 SAMPLE.FIL (LINE 62 COLUMN 12
```

This tells KEDIT to begin editing SAMPLE.FIL, and to start with line 62 as the current line, with the cursor positioned in column 12 of that line.

Line and column values can optionally be enclosed in double-quotes. For example:

```
KEDITW32 SAMPLE.FIL (LINE "62" COLUMN "12"
```

DEFPROFile *fileid* The DEFPROFILE option lets you change the fileid of the default profile to something other than WINPROF.KEX, or lets you give a drive and path specification for WINPROF.KEX, thereby avoiding a path search. With REPROFILE ON, this default profile may be re-executed many times during a KEDIT session, whenever a new file is added to the ring.

An example:

```
KEDITW32 TEST.FIL (DEFPROFILE C:\MACROS\MYPROF.KEX
```

If you specify the DEFPROFILE option on the command line, it overrides any DEFPROFILE value previously saved in the Windows registry.

FRAMEsize MINimized|MAXimized|NORMal|RECALL The FRAMESIZE option lets you specify that KEDIT should start with a minimized frame window, a maximized frame window, a normal (non-minimized, non-maximized) frame window, or whatever size frame win-

dow was in effect at the end of the last KEDIT for Windows session. If the FRAMESIZE option is not specified, KEDIT sizes the frame window according to the value of the INITIALFRAMESIZE option saved in the Windows registry which has RECALL as its default.

An example:

```
KEDITW32 TEST.FIL (FRAMESIZE MAX
```

INSTANCE SINGLE|MULTIPLE Use the INSTANCE option to specify whether, if another instance of KEDIT for Windows is already running when you try to invoke KEDIT, a new instance of KEDIT for Windows should be started. With INSTANCE SINGLE, the existing instance is activated, with any files that you specified on the command line added to the existing instance's ring of files, and no new instance is started. With INSTANCE MULTIPLE, a new instance is started and multiple copies of KEDIT for Windows will be running simultaneously.

If the INSTANCE option is not specified on the command line, KEDIT uses the value of SET INSTANCE that is saved in the Windows registry; this defaults to INSTANCE SINGLE.

An example:

```
KEDITW32 ANOTHER.CPY (INSTANCE MULTIPLE
```

LINE *n* The LINE option tells KEDIT that, as soon as the file is loaded, the specified line of the file should become the current line.

An example:

```
KEDITW32 SAMPLE.FIL (LINE 62
```

This tells KEDIT to begin editing SAMPLE.FIL, and to start with line 62 as the current line.

The LINE option does not affect the cursor position; the cursor is left on the command line and is not automatically moved to the current line. However, you can use the COLUMN option to move the cursor to a specified column of the current line.

Line and column values can optionally be enclosed in double-quotes. For example:

```
KEDITW32 SAMPLE.FIL (LINE "62"
```

LOCK The LOCK option tells KEDIT to lock the file you edit, so that no other programs can access the file until you have finished editing it. The LOCK option forces the file to be locked, even if LOCKING OFF is in effect.

MACROPath ON|OFF|*envvar*|*dirlist* The MACROPATH option can be used to control which directories KEDIT looks in when it searches for macros. By default, KEDIT searches the current directory and all directories specified in the PATH environment variable. Then it looks in the "KEDIT Macros" subdirectory of your Windows Documents or My Documents folder, the directory from which KEDIT for Windows was loaded, and the USER and SAMPLES subdirectories of that directory. See the discussion of SET MACROPATH for a description of the values that you can specify for the MACROPATH option.

If you specify the MACROPATH option on the KEDITW32 command line, it overrides any MACROPATH value saved in the Windows registry.

An example:

```
KEDITW32 TEST.FIL (MACROPATH KEXPATH
```

NEW KEDIT normally looks for the file that you want to edit on disk, reading it in if it exists, and editing a new file by that name only if no existing copy of the file can be found. Use the NEW option to tell KEDIT not to bother looking for your file on disk, because you want to edit a new file by that name regardless of whether the file already exists. KEDIT does not look in the current directory for your file and does not do a path search for your file, but instead adds an empty file with the specified fileid to the ring.

NODEFEXT The NODEFEXT option tells KEDIT to ignore the current setting of DEFEXT when determining a fileid, and to act as if DEFEXT OFF were in effect. NODEFEXT is always in effect for the first file added to the ring. The NODEFEXT option is provided for use on the KEDIT command line used to add additional files to the ring or move within the ring. It is used mainly in macros that work with fileids that might not include extensions.

NOFILEMENU When you finish editing a file, KEDIT normally adds its fileid to the list of recently-edited files displayed at the bottom of the File menu. Use the NOFILEMENU option to prevent a file (for example, a work file used temporarily by a KEDIT macro) from being added to the list of recently-edited files.

NOINI The NOINI does the same thing as the NOREG option; for details see the description below of the NOREG option.

(KEDIT for Windows now stores its configuration information in the Windows registry, but KEDIT for Windows 1.5 and earlier stored this information in the KEDITW.INI file. So NOREG is the newer name for this option, but for compatibility reasons NOINI remains available.)

NOLOCK The NOLOCK option tells KEDIT not to lock the file you edit. If LOCKING ON is in effect, you can use the NOLOCK option to override it for a particular file.

NOMsg The NOMSG option causes MSGMODE OFF to be put into effect for the file being added to the ring. This is a rarely used option, provided mainly for XEDIT compatibility.

NOPROFile The NOPROFILE option suppresses execution of your profile. It affects only the files currently being added to the ring, and has no effect on files added to the ring later in your KEDIT session.

NOREG The NOREG option suppresses KEDIT's processing of settings previously saved in the Windows registry. KEDIT uses built-in default values for all of its SET options, and does not use the values saved the registry via Options Save Settings. KEDIT also uses default values for its window positions, fonts, etc., and does not load information from the registry on recently-edited files, commands, and search strings.

When the NOREG option is used KEDIT puts REGSAVE NOSTATE NOHISTORY into effect, so that the existing state and history information in the Windows registry will not be overwritten at the end of your KEDIT session.

If you use the NOREG option, the information on saved settings displayed by the Options Status and Options Save Settings dialog box will be based on KEDIT's default settings and not on your actual saved settings, since in this case KEDIT does not process the saved settings.

The NOREG option is primarily used when KEDIT is not behaving as you expect and you want to determine whether the unexpected behavior is caused by some value saved in the registry or is instead KEDIT's default behavior. Note that even with the NOREG option in effect, KEDIT will still execute your profile unless you also specify the NOPROFILE option.

The NOREG option does the same thing as the NOINI option, an older option that remains available for compatibility with previous versions of KEDIT.

PATH ON|OFF|*envvar*|*dirlist* The PATH option can be used to control which directories KEDIT looks in when it searches for files you want to edit. By default, KEDIT searches the current directory and all directories specified in the PATH environment variable. Then it looks in the "KEDIT Macros" subdirectory of your Windows Documents or My Documents folder, the directory from which KEDIT for Windows was loaded, and the USER and SAMPLES subdirectories of that directory. See the discussion of SET PATH for a description of the values that you can specify for the PATH option.

If you specify the PATH option on the KEDITW32 command line, it overrides any PATH value saved in the Windows registry.

An example:

```
KEDITW32 TEST.FIL (PATH EDITPATH
```

PROFDEBUG The PROFDEBUG option helps you debug problems with your KEDIT profile by running your profile with the KEXX debugger active. It does this by internally issuing the command DEBUG WINPROF.KEX instead of the command MACRO WINPROF.KEX to run your KEDIT profile. This turns on the debugging window and runs your profile with interactive tracing in effect.

PROFile *fileid* The PROFILE option tells KEDIT not to run the default profile macro (controlled by the DEFPROFILE option and usually WINPROF.KEX) but instead to run the specified profile. Unlike the DEFPROFILE option, the PROFILE option only affects the profile for the files currently being added to the ring, and does not affect the profile to be used when additional files are added to the ring later in your editing session.

An example:

```
KEDITW32 TEST.FIL (PROFILE ALTPROF
```

UNTITLED The UNTITLED option tells KEDIT that you want to edit a new, untitled file, whose file name and extension are of the form UNTITLED.*n*.

If the UNTITLED option is used, the command line involved cannot specify a fileid, since KEDIT automatically generates a new UNTITLED.*n* fileid.

If no fileid is specified on the command used to start a KEDIT session, the UNTITLED option is assumed and need not be explicitly specified; this is why KEDIT for Windows automatically starts with a file called UNTITLED.1 if invoked with no fileid. The UNTITLED option is also used by the macro that handles the File New menu item.

Width *n* The WIDTH option, whose value can range from 1024 to 999999, controls the length of the longest line that KEDIT can read in or process. The default value for WIDTH is 10000, meaning that you can edit files with lines up to 10000 characters long; any lines that are longer are automatically split when KEDIT reads them in. To usefully edit files with longer lines, you will need use to a larger WIDTH value.

If you frequently need to edit files with lines longer than 10000 characters long, you can issue the SET INITIALWIDTH command within a KEDIT session; this value will be automatically saved in the Windows registry and will determine the WIDTH value put into effect by default in future KEDIT sessions. If you specify the WIDTH option on the command line used to invoke KEDIT for Windows, it overrides any INITIALWIDTH value saved in the Windows registry.

The WIDTH value is established during KEDIT initialization, and cannot be changed in the middle of a KEDIT session.

An example:

```
KEDITW32 (WIDTH 32000
```

Notes

Several of these options—PROFILE, NOPROFILE, LINE, COLUMN, PROFDEBUG, LOCK, NOLOCK, NEW, NODEFEXT, NOFILEMENU, UNTITLED, and NOMSG—can also be given from within KEDIT for Windows when you issue the KEDIT command to add additional files to the ring. These options let you

specify the profile to be executed, etc., for the newly-added file. You can also use the PROFILE, NOPROFILE, LINE, COLUMN, PROFDEBUG, and NOMSG options with the DIR, DIRAPPEND, and MACROS commands, since these commands can also cause a file to be added to the ring. Because of the special processing done internally by these commands, the other options are not relevant.

Most of these options are rarely used, and KEDIT for Windows is usually invoked without any options. Initialization options are most often used to override KEDIT's normal behavior during a particular editing session. For example, you could use the WIDTH option to change the maximum line length for a particular editing session, or use the DEFPROFILE option to change the default profile from WINPROF.KEX to some other fileid for a particular editing session. But to make changes to the maximum line length or default profile that would affect all future editing sessions, you would instead issue the SET INITIALWIDTH or SET DEFPROFILE commands from within KEDIT. If you then save their new values in the Windows registry via Options Save Settings, the saved settings will affect all future editing sessions. (Options Save Settings is required for SET DEFPROFILE; the registry is updated automatically for SET INITIALWIDTH and for a few other options.)

Several of the initialization options affect behavior that cannot be controlled directly from your profile because the processing involved takes place before KEDIT runs your profile. For example, KEDIT must determine the WIDTH value before your profile is executed so that it can set up some of the memory buffers that it needs to use while processing your profile. Similarly, the decision on what profile to run for the first file added to the ring must obviously be made before your profile can be executed, so any SET DEFPROFILE commands in your profile can only affect the default profile used for future files added to the ring, not the profile used for the first file added to the ring. (SET INITIALWIDTH or SET DEFPROFILE commands issued in a previous session and saved in the Windows registry would, however, affect the first file added to the ring in the current session.)

You can also specify initialization options through the environment variable KEDITW. During initialization, KEDIT for Windows looks for this environment variable and processes any options it contains.

```
SET KEDITW=options
```

The KEDITW environment variable is rarely used. Since it affects all KEDIT for Windows sessions and cannot be changed without first exiting Windows, it is only useful for options that you want in effect for all KEDIT sessions. But this type of option is usually controlled through the KEDIT SET command. For example, if you always want to use a WIDTH value of 32000 instead of the default of 10000, you can issue the command SET INITIALWIDTH 32000 from within KEDIT; this value will then be saved in the Windows registry and will take effect in all future KEDIT sessions.

Multiple options can be specified with the KEDITW environment variable or on the command line. For example,

```
SET KEDITW=DEFPROF C:\MACROS\MYPROF WIDTH 32000
KEDITW PROG.PAS (PROFILE PASCAL LOCK
```

If you specify the same option twice or you give conflicting options, the option specified last takes precedence. Options specified on the command line take precedence over options specified via the KEDITW= environment variable.

Passing arguments to your profile

When KEDIT runs your profile (either at the start of an editing session or when REPROFILE ON is in effect and an additional file is being added to the ring), it passes an argument string to your profile. This string consists of the name of the file being edited (including its full path specification) and any options that you specify on the command line, including the left parenthesis, optional right parenthesis, and any text that follows the right parenthesis. Your profile can use the ARG(1) function or the PARSE ARG instruction to access this information.

Any text that follows the optional right parenthesis on the command line is passed to your profile but is otherwise ignored by KEDIT. You can take advantage of this to pass parameters of your own to your profile. For example, consider the command line

```
KEDITW32 TEST.C (PROFILE SPECIAL) USER PARM
```

Your profile (in this case, the PROFILE option would cause the profile SPECIAL.KEX to run) could use the following to get at the parameter information passed to it:

```
parse arg fileid . '(' options ')' extra
```

This would set three variables. The variable *fileid* would get the fileid involved, including the full drive and path specification that KEDIT will use to edit the file. The variable *options* would get any initialization options specified, in this case "PROFILE SPECIAL". The variable *extra* would get the text that follows the right parenthesis, in this case " USER PARM". Your profile could then make decisions based on the values of these variables.

2.3 Initialization Processing

This section describes some of the processing done during KEDIT initialization, so that you can better understand the relationship between KEDIT's initialization options, KEDIT's section of the Windows registry, and your profile.

Single and multiple instances

When KEDIT for Windows is loaded, it checks to see if another instance of KEDIT for Windows is already executing. If not, KEDIT continues to initialize in the normal way. But if another instance is active, KEDIT needs to determine whether INSTANCE SINGLE should be in effect (in which case the existing instance of KEDIT will be activated and the new instance of KEDIT will terminate) or whether INSTANCE MULTIPLE should be in effect (in which case the new instance of KEDIT will continue, running simultaneously with the existing instance).

If the INSTANCE SINGLE or INSTANCE MULTIPLE initialization option has been specified, KEDIT puts the specified value into effect. If not, and if the NOREG and NOINI initialization options have not been specified, KEDIT checks the Windows registry to see if it contains a saved setting of INSTANCE MULTIPLE, and if so puts

INSTANCE MULTIPLE into effect. Otherwise, KEDIT puts the default of INSTANCE SINGLE into effect. KEDIT then acts accordingly, activating the existing instance for INSTANCE SINGLE or continuing to initialize the new instance for INSTANCE MULTIPLE.

Registry

Next, unless the NOREG or NOINI initialization options have been specified, KEDIT reads information previously saved in KEDIT's section of the Windows registry. Saved settings from the registry are put into effect, overriding the default settings built into KEDIT. KEDIT also processes the registry's status information (on KEDIT's fonts, window positions, etc.) and history information (KEDIT's recently-edited files, recently-issued commands, etc.).

Initialization options

At this point the initialization options (other than INSTANCE, NOREG, and NOINI, which have already been processed) are put into effect. Some of these options override the effect of KEDIT default settings or settings loaded from the Windows registry. For example, the WIDTH initialization option overrides any INITIALWIDTH setting loaded from the registry, and the PATH and MACROPATH initialization options override PATH and MACROPATH settings loaded from the registry.

Profile processing

KEDIT next searches for the initial file to be edited, if necessary doing a path search controlled by the PATH value, to locate the file. Then, unless the NOPROFILE option has been specified, KEDIT runs your profile.

If you have not used the PROFILE initialization option to specify the name of the profile to execute, KEDIT uses as the name of your profile the value of DEFPROFILE, which defaults to WINPROF.KEX. KEDIT looks for your profile in the current directory of the current drive. If KEDIT can't find it there, it then does a path search, controlled by the MACROPATH value, for your profile. Finally, it looks in the "KEDIT Macros" subdirectory of your Windows Documents or My Documents folder, the directory from which KEDIT for Windows was loaded, and the USER and SAMPLES subdirectories of that directory. If your profile still can't be found, KEDIT assumes that it doesn't exist.

Note that KEDIT begins to execute your profile after it has processed settings saved in the Windows registry, has processed any initialization options, and has determined which file will be edited, but before it has loaded the file into memory. This means that your profile can make decisions based on the fileid involved, that SET commands issued from your profile override the values of settings loaded from the registry, and that your profile can issue commands, like SET TABSIN and SET EOFIN, that affect the handling of your file during the loading process.

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. (Exceptions to this include the SET command, the EXTRACT command, implied EXTRACT functions, and Boolean functions, which are processed as they are encountered in your profile, and do not force loading of your file. This can affect the information they return. For example, SIZE.1() will return 0 if issued before your file is loaded, regardless of the size of your file. On the other hand, items like FEXT.1() don't depend on your file being loaded and can be usefully examined early in 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.)

Here is a full list of the commands that can be issued from your profile without forcing the loading of your file: SET, EXTRACT, DEFINE, CHDIR, CHDRIVE, CMSG, COMMAND, DEBUG, DMSG, DOS, DOSNOWAIT, DOSQUIET, EDITV, EMSG, ERASE, EXTRACT, HELP, HIT, IMMEDIATE, LEFT, MACRO, MSG, NOMSG, PRESERVE, PURGE, QQUIT, QUERY, QUIT, RENAME, RESTORE, RGTLEFT, RIGHT, SET, SHOWDLG, SYNEX, WINDOW, WINEXEC, WINHELP, WMSG, =, and ?.

The positioning done by the LINE and COLUMN initialization options takes place immediately after the file is loaded, and before KEDIT process any commands in your profile that operate on the contents of your file.

2.4 Editing Additional Files

The previous section described the initialization processing done by KEDIT when the first file is added to the ring at the start of an editing session. From then on, whenever a file is added to the ring, similar but somewhat simpler processing takes place.

There is no need to worry about how many instances of KEDIT are active, since the additional file will be edited by the current instance of KEDIT. KEDIT keeps a copy of the saved settings from the Windows registry in memory throughout the editing session, so KEDIT does not need to re-read these settings when a new file is added to the ring.

Initialization options

KEDIT does not re-process the initialization options set via the KEDITW= environment variable, but KEDIT does process any initialization options present on the KEDIT command line used to add the additional file to the ring. Only the PROFILE, NOPROFILE, PROFDEBUG, LOCK, NOLOCK, NEW, NODEFEXT, NOFILEMENU, UNTITLED, and NOMSG options are valid at this point.

Profile processing

When a file is added to the ring, KEDIT needs to initialize all of its File level settings, as well as the View level settings for your initial view of the file. How this is done depends on whether your profile will be re-executed, which in turn depends on the value of SET REPROFILE.

If REPROFILE ON is in effect, or if the PROFILE initialization option is specified, KEDIT first initializes the File and View level settings according to the saved settings in the Windows registry. (If you specified the NOREG or NOINI when invoking KEDIT, KEDIT instead uses its built-in default values and does not use the saved settings from the registry). KEDIT then runs your profile, which may then make changes to these settings.

If REPROFILE OFF is in effect, your profile will not be re-executed. KEDIT copies most of the File and View level settings from the current file to the newly-added file.

(ALT, LRECL, RECFM, DISPLAY, SCOPE, TRUNC, VERIFY, and ZONE are exceptions to this; their initial defaults are used.)

Notes

If you specify multiple files at a time on the KEDIT command line used to add additional files to the ring after a KEDIT session has started, the processing described in this section takes place repeatedly, once for each file added to the ring.

If you specify multiple files at a time on the command line used to invoke KEDIT for Windows, the full KEDIT initialization processing described in the preceding section takes place for the first of those files and the processing described in this section takes place for the rest of those files.

Chapter 3. KEDIT Commands

This chapter gives detailed information on all KEDIT commands, with the exception of the SET, QUERY, and EXTRACT commands. Full documentation on the SET command is in Chapter 4, "The SET Command", and details of the QUERY and EXTRACT commands are in Chapter 5, "QUERY and EXTRACT".

ADD

Format **`Add [n]`**

Description The ADD command adds *n* blank lines to your file. If *n* is not given, one line is added.

The blank lines are added below the focus line. The first blank line added becomes the new focus line and the cursor is positioned in the left margin column of this line.

See also SOS LINEADD

Examples

```
ADD
```

Add one blank line below the focus line. (You can also press function key F2 to add one line to your file.)

```
ADD 5
```

Add five blank lines below the focus line.

ALERT

Format **`ALERT /prompt/ [options]`**

where *options* can be:

```
EDITfield [/initial/]
TITLE /title/
OK|OKCANCEL|YESNO|YESNOCANCEL
DEFButton n
ICONExclamation|ICONInformation|ICONQuestion|ICONStop
FIXEDfont
PASSWORD
```

Description Use the ALERT command within KEDIT macros to display dialog boxes that present error messages or messages of special importance to users of the macro. The more-frequently-used DIALOG command is used for normal dialog boxes.

See the DIALOG command for a full discussion of the options involved. The ALERT command and the DIALOG command take the same operands and behave identically, with these exceptions:

Unless you explicitly specify a different type of icon, the ICONSTOP (stop sign) icon is displayed when you use the ALERT command.

If BEEP ON is in effect, the speaker will beep when an ALERT dialog box is displayed.

The results from ALERT are returned in the macro variables ALERT.0, ALERT.1, and ALERT.2. The DIALOG command instead uses DIALOG.0, DIALOG.1, and DIALOG.2.

See also DIALOG, POPUP

Examples

```
ALERT /Invalid name specified/ TITLE /Error/
ALERT /Option invalid; continue ?/ YESNO DEFBUTTON 2
```

ALL

Format **ALL [*target*]**

Description

The ALL command causes KEDIT to search through your file for all lines containing the specified *target*, using KEDIT's selective line editing facilities to select these lines for display. All other lines of your file are temporarily excluded from the display.

When you once again want to work with all lines of your file, you can issue the ALL command with no operands, and all lines of your file will again be selected.

Note that you can also use the Edit Selective Editing dialog box to control the selective line editing facility.

User's Guide Chapter 8, "Selective Line Editing and Highlighting", gives a complete introduction to the ALL command and to KEDIT's selective line editing facilities.

After completion of the ALL command with a *target* operand, the first selected line becomes the focus line, and KEDIT puts SCOPE DISPLAY into effect. The ALL command gives lines that match the specified *target* a selection level of 1, gives lines that don't match a selection level of 0, and sets DISPLAY to 1 1 so that only matching lines will be selected for display. With SCOPE DISPLAY, lines that are excluded from your display are also excluded from processing by all KEDIT commands except FILE, SAVE, and SORT. If you put SCOPE ALL into effect, KEDIT commands operate on all lines, even lines that are excluded from the display.

If SHADOW ON (the default) is in effect, a shadow line appears on your display wherever lines have been excluded, indicating how many lines are excluded. With SHADOW OFF, excluded lines are not represented at all on your display.

The ALL command with no operands resets the selection level of all lines in the file to 0, and then sets DISPLAY to 0 0, so that all lines of the file are selected for display. ALL with no operands does not change the focus line location.

See also

User's Guide Chapter 8, "Selective Line Editing and Highlighting", LESS, MORE, TAG, SET DISPLAY, SET SCOPE, SET SELECT, SET SHADOW, QUERY NBSCOPE

Examples

```
ALL /telephone/
```

All lines of your file containing the string ''telephone'' are selected for display; all other lines are excluded. The first line of your file containing ''telephone'' becomes the focus line. (If your file does not contain ''telephone'', KEDIT issues an error message and takes no further action.)

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

All lines of your file containing ''upper'' or ''lower'' are selected for display.

```
ALL
```

Issuing the ALL command with no operands causes all lines of the file to be selected for display, effectively canceling the effect of a previous ALL command that caused lines to be excluded.

```
ALL BLANK
DELETE *
ALL
```

In this example, all blank lines are deleted from a file. The first ALL command selects all blank lines in the file for display and further processing and makes the first selected line the focus line. The DELETE command then deletes all selected lines (in this case, all blank lines) from the file. The second ALL command then causes all lines remaining in the file to once again be selected.

```
ALL BLOCK
```

This is handled as a special case. All lines in the currently defined block are selected, and all other lines in the file are excluded.

ALTER

Format

ALter *char1 char2* [*target* [*n* [*m*]]]

Description

The ALTER command is similar to the CHANGE command, but rather than changing one string of characters to another, it changes one character, *char1*, to another, *char2*.

Either or both of the characters can be special characters not represented on the PC's keyboard. To understand this description of the ALTER command, you should first read the description of the CHANGE command. The *target*, *n*, and *m* operands work in the same way as they do with the CHANGE command.

With the ALTER command, occurrences of *char1* (the first character that you specify) are changed to *char2* (the second character that you specify). You can specify *char1* and *char2* in either of two ways. If the character appears on a key on the keyboard, you can simply enter it. For example, to change an "A" into a "B" you could enter

```
ALTER A B
```

You can also specify *char1* and *char2* by giving the character codes for the characters involved. This allows you to easily work with special characters not on the PC's keyboard. You enter the two- or three-digit *decimal* value of the code for the character you want. For example, to change a formfeed character (character code 12) to a tab character (character code 9), you could enter

```
ALTER 12 09
```

The two methods can be mixed:

```
ALTER 09 +
```

This example would change a tab character into a plus sign. (KEDIT requires "09" and not simply "9" in these examples so that it can distinguish between the tab character, which has character code 9, and the character "9", which has character code 57.)

When issued from a macro, the ALTER command sets the macro variable ALTER.0 to 2, returns the number of occurrences changed in ALTER.1, and returns the number of lines changed in ALTER.2.

See also CHANGE

Examples

```
ALTER 09 32 ALL *
```

Change all occurrences of tab characters (character code 9) throughout the entire file to blanks (character code 32).

```
ALTER + 12 :15 *
```

Change all occurrences of "+", in all lines from the focus line up to but not including line 15, to formfeed characters (character code 12).

ANSITOOEM

Format **ANSITOOEM [*target*]**

Description Use the ANSITOOEM command to convert text in a specified portion of your file from the ANSI character set to the OEM character set.

All text within the specified target area that falls within the current ZONE columns is converted. If the target area is a box block, its entire contents are converted, regardless of the ZONE settings.

See also OEMTOANSI, User's Guide Section 3.7, "Character Sets"

Examples

```
ANSITOOEM ALL
```

Convert all lines of the file from ANSI to OEM.

BACKWARD

Format **BAckward [*n*|*|HALF|*m* Lines]**

Description The BACKWARD command causes KEDIT to scroll the current window backward, towards the top of your file.

BAckward
KEDIT scrolls one window backward in your file. This command is normally assigned to the Page Up key.

BAckward *n*
KEDIT scrolls *n* windows backward in your file, as if you had pressed the Page Up key *n* times.

BAckward *
KEDIT scrolls backward all the way to the top of your file, making the top-of-file line become the focus line.

BAckward HALF
KEDIT scrolls one half-window backward in your file.

BAckward *m* Lines
KEDIT scrolls backward *m* lines in your file.

You can also move forward and backward in your file by using the mouse to manipulate the vertical scroll bar.

See also FORWARD

Examples

```
BACKWARD 2
```

KEDIT scrolls backward two windows in your file, as if you had pressed the Page Up key twice.

```
BACKWARD 4 LINES
```

KEDIT scrolls four lines backward in your file.

BOTTOM

Format **`Bottom`**

Description The BOTTOM command makes the last line of the file become the focus line. With INTERFACE CUA in effect, you can also press Ctrl+End to get to the end of your file. With INTERFACE CLASSIC you can instead press Ctrl+Page Down.

See also TOP

CANCEL

Format **`CANCEL`**

Description The CANCEL command causes KEDIT to internally issue QUIT commands for all files in the ring. Any files that have been changed since their last SAVE will remain in the ring, since QUIT only affects files that have not been changed.

See also QUIT, QQUIT

CAPPEND

Format **`CAppend [`*`text`*`]`**

Description The CAPPEND command (''column append'') sets the focus column to be one character beyond the last character of the focus line and then appends any text that you specify to the focus line, beginning at the focus column.

For example, if the focus line consisted of

```
Hello
```

and you entered the command

```
CA␣␣there.
```

the focus line would be changed to

```
Hello there.
```

The focus column would be positioned at the blank between ''Hello'' and ''there''.

Note that in the above example, the ''CA'' is followed by two blanks. The first blank following the command is ignored, but any additional blanks (in this case there is one) are appended to your text. Had you issued the command with only one blank

```
CA␣there.
```

no blanks would have been appended after the ''o'' of ''Hello'' and the focus line would look like this:

```
Hellothere.
```

If CAPPEND is entered with no operands, no change is made to the focus line, but the focus column is set just beyond the last character of the line.

CAPPEND can be used in connection with the REPEAT command to append the same text to the end of a group of lines.

Examples

```
CAPPEND␣.
```

A period is added after the last nonblank character of the focus line. The focus column is set to the column with the period.

```
CAPPEND␣␣ABCDEF
```

Two blanks have been entered after CAPPEND. A blank and the letters ''ABCDEF'' are placed at the end of the focus line, and the column pointer is set to point to the blank.

CDELETE

Format

CDelete [*column-target*]

Description

The CDELETE command (''column delete'') deletes text, starting at the focus column and continuing through the column indicated by the column target operand. (If no operand is given, only the character at the focus column position is deleted.)

Column targets, discussed in detail in User's Guide Chapter 6, "Targets", are a special type of target used only with the CDELETE and CLOCATE commands.

If the column target that you specify is on the focus line, only characters on the focus line are deleted. If the column target is not located on the focus line (you specified a string column target, STREAM ON was in effect, and KEDIT had to search beyond the focus line to find the string) characters on two or more lines are deleted. Any lines on which all characters are deleted are simply deleted from your file. If only some characters on a line are deleted, the line is not deleted from your file, but the required characters are removed from the line.

After the deletion, the last line on which characters were deleted becomes the focus line and the focus column's location is unchanged.

See also

SET STREAM

Examples

```
CDELETE 2
```

The character at the focus column and the character following it are deleted.

```
CDELETE -3
```

The character at the focus column and the two characters preceding it are deleted.

```
CDELETE *
```

All characters on the focus line, starting from the character at the focus column, are deleted.

```
:1
CL :73
CDELETE *
REPEAT *
```

This sequence of commands would move you to line 1 of the file and make column 73 become the focus column. Then it would go through the file, removing all characters in the file located at or to the right of column 73.

```
CDELETE /Hello/
```

All characters from the character at the focus column up to (but not including) the ''H'' in the next occurrence of ''Hello'' are deleted. If STREAM OFF is in effect, the occurrence of ''Hello'' must be in the focus line or an error will result.

CENTER

Format **CEnter [*target*]**

Description Text in the specified *target* area is centered within the current margins.

You can use CENTER BLOCK to center line blocks, box blocks, and one-line stream blocks. Box blocks and one-line stream blocks are given special handling: KEDIT centers the text within the block boundaries, and text outside the block is not affected. You cannot center a multi-line stream block.

See also LEFTADJUST, RIGHTADJUST, SET MARGINS

Examples

```
CENTER
```

Text in the focus line is centered within the current margins. This is the default assignment for Shift+Ctrl+C.

```
CENTER ALL
```

All text in the file is centered within the current margins.

CFIRST, CLAST

Format

```
CFirst
CLAst
```

Description

The CFIRST command (“column first”) makes the left zone column become the focus column.

The CLAST command (“column last”) makes the right zone column become the focus column.

CHANGE

Format

Change */string1/string2/* [*target* [*n* [*m*]]]

Description

Use the CHANGE command to change occurrences of one string to another. Note that the Edit Replace dialog box provides a simpler but somewhat less powerful way to make such changes.

The CHANGE command causes KEDIT to search your file for occurrences of *string1* and to change them to *string2*. KEDIT examines all lines in the *target* area and changes the first *n* occurrences of *string1* in each line to *string2*. (An additional operand, *m*, lets you specify that KEDIT start with the *m*th occurrence of *string1* rather than with the first occurrence.) If no *target* is given, KEDIT only searches the focus line. If *n* is not given, only the first occurrence of *string1* in each line is changed to *string2*. You may enter an asterisk (“*”) instead of a number *n* to indicate that all occurrences of *string1* on a given line should be changed to *string2*. For example,

```
CHANGE /Hello/Goodbye/ 10 *
```

This example tells KEDIT to change all occurrences of “Hello” in the focus line and the nine lines following it, for a total of ten lines, to “Goodbye”.

String1 and *string2* are set off by delimiter characters, which are normally slash characters (“/”). However, any special character (characters other than letters, numbers, and blanks) can be used as the delimiter character. The only restriction is that the delimiter character cannot appear anywhere within *string1* or *string2*.

If *string1* is longer than *string2*, the line of text involved will be shortened by the change. If *string1* is shorter than *string2*, the line will be lengthened by the change. If the resulting line would extend beyond the truncation column, all characters pushed beyond the truncation column will be lost.

When searching a line for text that matches *string1*, KEDIT is affected by the settings of ARBCHAR, CASE, and ZONE. The description of SET HEX discusses the hexadecimal and decimal notation you can use for *string1* and *string2* if HEX ON is in effect.

WORD, PREFIX, and SUFFIX (which can be truncated to ‘‘W’’, ‘‘P’’, and ‘‘S’’) can precede */string1/*, just as they can with string targets, to indicate that the occurrences of *string1* that you want to change must occur at word boundaries (or, for PREFIX or SUFFIX, be preceded or followed by word boundaries). For example,

```
CHANGE WORD /The/Some/ 1 *
```

would change any occurrences of the word ‘‘The’’ on the focus line, but would not change occurrences of the word ‘‘These’’.

You can also precede *string1* with REGEXP (which can be truncated to ‘‘R’’). This allows you to use regular expression notation in *string1* and can use references to tagged regular expressions in *string2*. For example,

```
CHANGE R /ABC{[0-9]}XYZ/&1&1/ ALL *
```

would change all occurrences in your file of ‘‘ABC’’ followed by a digit followed by ‘‘XYZ’’ to two occurrences of that digit. So ‘‘ABC5XYZ’’ would become ‘‘55’’. KEDIT’s regular expression support is described in detail in User’s Guide Chapter 6, “Targets”.

If STAY ON is in effect, the line that is the focus line when you enter the CHANGE command remains the focus line when the CHANGE command has completed. If STAY OFF is in effect, the last line scanned becomes the new focus line after the CHANGE command completes.

You can make the CHANGE command operate within the currently-defined block by using BLOCK as the *target* operand. The CHANGE command will operate within the portions of a line or stream block that are within the current zones. Box blocks are given special handling: KEDIT looks for *string1* within the boundaries of the box block, regardless of the zone settings, and does not shift text in or out of the box while making changes.

If you issue the CHANGE command with no operands, KEDIT re-executes the last CHANGE command issued from the command line.

When issued from a macro, the CHANGE command sets the macro variable CHANGE.0 to 3, returns the number of occurrences changed in CHANGE.1, returns the number of lines changed in CHANGE.2, and returns the number of lines truncated (because the changed text would have extended beyond the truncation column) in CHANGE.3.

See also

ALTER, COUNT, SCHANGE, SET ARBCHAR, SET CASE

Examples

```
CHANGE /A/B/* *
```

Changes all occurrences of the letter ‘‘A’’ to the letter ‘‘B’’ in all lines from the focus line through the bottom of the file.

```
CHANGE /A/B/ALL *
```

Changes all occurrences, in all lines of your file, of the letter ‘‘A’’ to the letter ‘‘B’’.

CHANGE /1234/5678/

Changes the first occurrence of "1234" in the focus line to "5678".

CHANGE /87 years ago/Four score and seven years ago/

Changes the first occurrence of "87 years ago" in the focus line to "Four score and seven years ago".

CHANGE ;A/B;C/D; 1 *

Changes all occurrences of "A/B" in the focus line to "C/D". Note that you could not use "/" as a delimiter here, since *string1* and *string2* both contain slashes. A semicolon (";") was used instead.

CHANGE /large/small/ PARA 2 3

"PARA" indicates that changes are to occur in the focus paragraph. The "2" indicates that at most two occurrences per line of "large" are to be affected. The "3" indicates that changes are to start with the third occurrence of "large" on each line. So this command changes the third and fourth occurrences of "large" to "small" on each line of the focus paragraph.

CHANGE

Since no operand is given here, KEDIT re-executes the last CHANGE command issued from the command line.

CHDIR, CHDRIVE

Format

CHDir [*d:*]*path*
CHDRive *d*[:]

Description

Use the CHDIR command to make a different directory become the current directory. If both a drive and directory are specified, KEDIT changes both the current drive and the current directory.

You can use CHDIR or its minimal truncation, CHD, to change to a new current directory from within KEDIT. CD, which is often used for this purpose from the DOS command line, cannot be used, because KEDIT treats CD as an abbreviation of the CDELETE command. Another way to change the current directory from within KEDIT is to use the File Directory dialog box.

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 the CHDIR

command does not affect the default directory for the File Open dialog box, which KEDIT keeps separate track of.

CHDIR =

is handled as a special case; the drive and directory of the current file become your current drive and directory.

Use the CHDRIVE command to make a different drive become the current drive; whatever current directory is already in effect for that drive will become KEDIT's current directory.

You can use the QUERY DIRECTORY command to determine the current drive and directory, or to determine the current directory of a specified drive.

See also

SET INITIALDIR

Examples

CHDIR \PROJECT

Makes the \PROJECT directory of the current drive become the current directory.

CHDIR E:\PROJECT

Makes E: the current drive and makes E:\PROJECT the current directory.

CHDIR "E:\My Project"

Makes E: the current drive and makes E:\My Project the current directory. Note that directory names containing blanks must be enclosed in double quotes.

CHDIR =

If you are editing, for example, C:\TEST\NOTES.TXT, then the C: drive becomes the current drive and C:\TEST becomes the current directory.

CHDRIVE D:

Makes the D: drive become the current drive; the D: drive's current directory becomes your new current directory.

CHDIR \\SERVER\PROJECTS\TEST

This makes the directory \\SERVER\PROJECTS\TEST, accessed using a UNC (Universal Naming Convention) name, become the current directory. Note that when a UNC directory is the current directory, there is no current drive.

CINSERT

Format

CInsert *text*

Description

The CINSERT command (''column insert'') inserts the text you specify into the focus line, beginning at the focus column location. Text that was at or to the right of the focus column location is shifted to the right to make room for the inserted text.

A single blank separates CINSERT from the text that you want inserted. If CINSERT is followed by more than one blank, all blanks after the first are taken to be part of the text that you are inserting. KEDIT inserts the text that you enter, from the character after the blank following CINSERT through the end of the command string, including any trailing blanks.

Here is an example of the CINSERT command. Assume the focus line looks like this:

```
ABCDEJKLM
```

If you issue

```
CLOCATE :6
```

the focus column will be at the ''J'' in column 6 of the focus line. If you then issue

```
CINSERT FGHI
```

KEDIT will insert ''FGHI'', starting at column 6 and pushing ''JKLM'' to the right, yielding

```
ABCDEFGHIJKLM
```

The CINSERT command does not change the focus column or focus line location. Any characters that would be inserted beyond the truncation column, or that would be shifted beyond the truncation column by the insertion, are truncated.

See also

CREPLACE, COVERLAY

Examples

```
CINSERT␣␣Hello
```

Here, the ''CINSERT'' is followed by two blanks, so the second blank becomes part of the text to be inserted. KEDIT inserts a blank followed by ''Hello'' into the focus line, starting at the focus column location.

```
CI X
```

KEDIT inserts an ''X'' into the focus line, starting at the focus column location.

```
CI X␣␣
```

KEDIT inserts an ''X'' followed by two blanks, starting at the focus column location.

CLIPBOARD

Format

```
CLIPboard COPY|CUT|PASTE
CLIPboard APPEND CUT|COPY
CLIPboard PUT text
CLIPboard APPEND PUT text
CLIPboard CLEAR
```

Description

The CLIPBOARD command moves data to or from the Windows clipboard. It is used mainly in the macros that handle the clipboard related items on the Edit menu, and in their toolbar and keyboard equivalents.

CLIPboard COPY

If the cursor is on the command line, the contents of the command line selection are copied to the clipboard, replacing any previous contents of the clipboard. If there is no command line selection, KEDIT will instead copy the currently marked block to the clipboard. Command line selections are possible only if INTERFACE CUA is in effect.

If the cursor is in the file area, the contents of the currently marked block are copied to the clipboard, replacing the previous contents of the clipboard. An error message is displayed if there is no block marked in the current file.

The Edit Copy menu item, the Copy to Clipboard toolbar button, Ctrl+Delete, and (if INTERFACE CUA is in effect) Ctrl+C all issue the CLIPBOARD COPY command.

CLIPboard CUT

This does the same thing as CLIPBOARD COPY (that is, it copies the contents of a marked block or command line selection to the clipboard), and then it deletes the data involved from your file or from the command line.

The Edit Cut menu item, the Cut to Clipboard toolbar button, Shift+Del, and (if INTERFACE CUA is in effect) Ctrl+X all issue the CLIPBOARD CUT command.

CLIPboard PASTE

If the cursor is on the command line, data from the clipboard is copied to the KEDIT command line at the cursor position, replacing any existing command line selection. You can only paste a single line of text from the clipboard to the command line. If there are multiple lines of text in the clipboard, attempts to paste to the command line will cause an error message. Any tab characters in data pasted to the command line are changed to blanks.

If the cursor is in the file area, data from the clipboard is copied into your file. If INTERFACE CUA is in effect and you have just marked an anchored block (that is, marked a block with the mouse or with Shift+cursor-pad-key), KEDIT first deletes the block, then positions the cursor at the location of the deleted block, and then inserts the clipboard's contents at that location. Otherwise, KEDIT inserts the marked block at the cursor position.

If the clipboard data came from a KEDIT stream block, from a KEDIT command line selection, or from some application other than KEDIT for Windows, it is inserted into your file at the cursor position as a stream of text. If the data came from a KEDIT box block, it is inserted at the cursor position as a rectangular section of text. If the data came from a KEDIT line block, it is inserted below the cursor line as a group of new lines.

Tab characters in clipboard data pasted into your file from other applications are expanded to blanks. If TABSIN OFF is not in effect, KEDIT does this according to the current TABSIN column settings; otherwise KEDIT uses the current SET TABS columns.

The Edit Paste menu item, the Paste from Clipboard toolbar button, Shift+Ins, and Ctrl+V all issue the CLIPBOARD PASTE command.

CLIPboard APPEND CUT
CLIPboard APPEND COPY

If there is no text in the clipboard, these commands have the same effect as CLIPBOARD CUT and CLIPBOARD COPY. Otherwise, these commands append the contents of the current selection to the existing clipboard text.

If text is already in the clipboard and it does not end in a new line sequence (a carriage return and a linefeed), these characters are added to the existing clipboard text before the contents of the selection are appended.

CLIPboard PUT *text*
CLIPboard APPEND PUT *text*

CLIPBOARD PUT provides a way to directly set the contents of the clipboard from within a KEDIT macro.

The specified text replaces, or is appended to, any existing text in the clipboard.

For CLIPBOARD APPEND PUT, if text is already in the clipboard and it does not end in a new line sequence (a carriage return and a linefeed), these characters are added to the existing clipboard text before the new text is appended.

CLIPboard CLEAR

The clipboard is emptied of any data that it contains.

CLOCATE

Format **CLocate *column-target***

Description Use the CLOCATE command (“column locate”) to locate a column target. KEDIT changes the focus column (and in some situations the focus line) so that it is positioned at the column target that you specify.

Column targets, discussed in detail in User’s Guide Chapter 6, “Targets”, are a special type of target used only with the CDELETE and CLOCATE commands.

If you issue the CLOCATE command with no operands, KEDIT will re-execute the last CLOCATE command issued from the command line, searching again for the same column target.

Operation of the CLOCATE command is affected by the settings of ZONE, CASE, ARBCHAR, VARBLANK, HEX, WRAP, STAY, and STREAM. If THIGHLIGHT ON is in effect, as it is by default, KEDIT will highlight strings found by CLOCATE on your display.

Examples

```
CL 1
```

The focus column moves one column to the right of its current position.

```
CL :20
```

The focus column moves to column 20.

```
CL *
```

The focus column moves one column beyond the right zone column.

```
CL /ABC/
```

The focus column moves to the first character of the next occurrence of "ABC". If STREAM OFF is in effect, this occurrence must be on the focus line. If STREAM ON is in effect and the occurrence is not on the focus line, the focus line location also changes.

```
CL ~/X/
```

The next column that does not contain an "X" becomes the focus column.

```
CL -/Hello/
```

KEDIT searches the focus line, starting one column to the left of the focus column, for the string "Hello". If "Hello" is not found by the time the search reaches the left zone column, and if STREAM ON is in effect, KEDIT searches the lines preceding the focus line, from right to left.

CMATCH

Format

```
CMATCH [OUTER|INNER]
```

Description

Use the CMATCH ("column match") command to find a parenthesis, brace, or bracket that matches the character in the focus column. When you work with programming languages like C that make frequent use of balanced pairs of braces and parentheses, the CMATCH command can help you verify that matching braces and parentheses are properly positioned.

CMATCH is most useful when assigned to a key or to a toolbar button, so that if you press the key with the cursor positioned on, for example, a left parenthesis, the cursor

will move to the matching right parenthesis. By default, the CMATCH command is assigned to Shift+F3.

If the character at the focus column is a "{" or "}", the CMATCH command moves the focus column to the matching "}" or "{". The same thing happens with "(" and ")", with "[" and "]", and with "<" and ">".

If you are working in a file with syntax coloring enabled and you are using a parser that defines matching items like parentheses, the CMATCH command will properly handle nested pairs of matching items and will skip over text in quotes or in comments.

If syntax coloring is not enabled, or if the active parser does not handle the item at the focus column, CMATCH will still handle nested pairs of parentheses, braces, and brackets, but will not properly handle parentheses, braces, and brackets inside comments or quotes.

CMATCH can also handle matching keywords that are defined in a syntax coloring parser. For example, in a KEXX macro containing the following:

```
if a = 5 then do
  j = 17
  do I = 1 to 10
   say i*j
  end
end
```

you can place the cursor on the DO at the end of the first line and press Shift+F3 to move the cursor to the corresponding END in the last line.

The OUTER and INNER operands control what happens with matching items for which the syntax coloring parser has defined beginning, middle, and end elements. With CMATCH OUTER and CMATCH with no operands, the cursor will move between the beginning and end elements. With CMATCH INNER, the cursor will move to the next beginning, middle, or end element.

For example, in the following C code each of #if, #elif, #else, and #endif are highlighted by the syntax coloring facility:

```
#if defined(a)
  x = 17;
#elif defined(b)
  x = 19;
#else
  x = 20;
#endif
```

If the cursor is positioned on any of these items, CMATCH OUTER and CMATCH with no operands will move the cursor to the next outer element. If the cursor is on #if, #elif, or #else, it will move to #endif. If the cursor is on #endif, it will cycle back to #if.

With CMATCH INNER, the cursor will stop at inner items as well as outer items, moving from #if to #elif to #else to #endif and back to #if again.

CMSG

Format **CMSG [*text*]**

Description The CMSG (“command line message”) command, used primarily in macros, displays the specified *text* on the command line.

See also DMSG, EMSG, MSG, WMSG

Examples

```
CMSG modify zone
```

KEDIT displays “modify zone” on the command line.

COMMAND

Format **COMMAND *command***

Description KEDIT usually checks each command issued from the command line to see if you have used the SET SYNONYM command to redefine its behavior. If so, KEDIT processes the command as specified in the synonym definition. The COMMAND command allows you to bypass this synonym processing, so that the command is executed exactly as you specified it, and any synonym for the command is ignored.

COMMAND is useful mainly when issued from the command line, because the normal action for commands issued from macros is to bypass synonym processing. When you want synonym processing to apply to a command issued from within a macro, you must specifically request it via the SYNEX command.

The effect of the COMMAND command is similar to the effect of preserving the status of SYNONYM, setting SYNONYM OFF, issuing the desired command, and restoring the status of SYNONYM.

See also SYNEX, SET SYNONYM

Examples

```
COMMAND DELETE 3
```

KEDIT directly executes the DELETE 3 command, regardless of whether you have defined a synonym for the DELETE command.

COMPRESS

Format **COMPress [*target*]**

Description

The COMPRESS command compresses strings of one or more blanks in your text, replacing them with tab characters. If you issue the COMPRESS command with no operands, the focus line is compressed. Otherwise, all lines in the target area are compressed. After the compression, the focus line location is unchanged if STAY ON (the default) is in effect; otherwise, the last line compressed becomes the new focus line.

To compress a line, the COMPRESS command looks at each tab column in the line (as set with the SET TABS command). Any string of one or more blanks leading up to the tab column is replaced by a single tab character (character code 9). For example, assume that you have issued

```
SET TABS 1 6 11 16
```

and that the focus line looks like this (with each pair of characters starting in a tab position):

```
AB    CD    EF    GH
```

Compressing the line yields the following (with tab characters shown here as plus signs ("+")):

```
AB+CD+EF+GH
```

The COMPRESS command can be useful in combination with the EXPAND command, which expands lines by replacing tab characters with strings of blanks extending to the next tab position. For example, if you had a 20-line table with entries lined up in columns 1, 6, 11, 16 and wanted to readjust the table entries to start in columns 1, 10, 20, 30 you could first issue

```
SET TABS 1 6 11 16
```

and then compress the table with

```
COMPRESS 20
```

Then you would set up tab positions for the new table:

```
SET TABS 1 10 20 30
```

and finally expand the tab characters created by the COMPRESS command according to the new tab positions:

```
EXPAND 20
```

In the above example, expanding the sample line using tab positions 1, 10, 20, 30 yields

```
AB          CD          EF          GH
```

Compressing a line of text that contains no tab characters and then, without changing the tab settings, immediately expanding it restores the line to its original state.

The COMPRESS and EXPAND commands are relatively rarely used. More frequently used are the SET TABSIN and SET TABSOUT commands, which control whether KEDIT automatically expands tab characters in a file when it reads the file in from disk and compresses them when it writes the file out to disk.

See also EXPAND, SET TABS, SET TABSIN, SET TABSOUT

Examples

```
COMPRESS BLOCK
```

All lines in the currently marked line block are compressed according to the current tab settings.

```
COMPRESS :12
```

All lines from the focus line to line 12 (but not including line 12) are compressed according to the current tab settings.

COPY

Format

COpy *target1 target2*
COpy BLOCK

Description Use the COPY command to copy text from one location to another.

There are two forms of the command:

COpy *target1 target2*

All text in the target area specified by *target1* is copied, with the copied text placed immediately after the line specified by *target2*. With this form of the command, the target area specified by *target1* cannot be a box or stream block and text can only be copied within the current file, and not from one file in the ring to another.

COpy BLOCK

This form of the COPY command copies all text in the currently marked block. The block can be in the current file or in another file, allowing you to copy text from one file to another. If the block is a line block, text is copied after the focus line. If the block is a box or stream block, text is copied to the left of the text at the focus column position. The block mark moves with the copied text; the old copy of the block is no longer marked.

COPY BLOCK is assigned by default to Alt+C and Alt+K. Alt+K copies the block but leaves the copy marked. Alt+C copies the block, and then uses the RESET command to unmark the block.

You can also copy blocks by using the Copy button on the default bottom toolbar, by using Ctrl+mouse button 1 to drag and drop the block, and by using the Edit menu to copy the block to the clipboard and then paste it to a new location.

See also MOVE

Examples

In the following examples, assume that line 4 is the focus line.

```
COPY :12 /ABC/
```

Lines 4 through 11 are copied following the next line containing "ABC".

```
COPY 6 :15
```

Lines 4 through 9 of the current file (a total of six lines) are copied following line 15 of the current file.

```
COPY BLOCK -*
```

A copy of all lines in the currently marked block is placed following the top-of-file line. This form of the command works only if there is a line block marked in the current file.

```
COPY BLOCK
```

If the currently marked block is a line block, KEDIT copies its contents following line 4 of the file. If the block is a box or stream block, KEDIT copies the block to the left of the focus column position.

COUNT

Format

COUnt /*string*/ [*target*]

Description

Use the COUNT command to count the number of occurrences of a given *string* in the specified *target* area of your file. (If no *target* is specified, KEDIT counts the occurrences of *string* in the focus line.)

The *string* that you search for is specified in the same way as the string you search for with the CHANGE command. Slash ("/") characters are normally used to delimit strings, but you can use any special character that does not appear within the string.

You can make the COUNT command operate within the currently-defined block by using BLOCK as the *target* operand. The COUNT command will operate within the portions of a line or stream block that are within the current zones, and will operate within the entire area of a box block, regardless of the zone columns.

If the *target* area is a box block, KEDIT counts all occurrences of *string* within the boundaries of the block. Otherwise, KEDIT counts all occurrences of *string* within the current ZONE settings.

The search for occurrences of *string* is also affected by the settings of ARBCHAR, CASE, and HEX, in the same way that these settings affect the search for the *string1* operand of the CHANGE command.

When the COUNT command completes, it displays a message giving the number of occurrences of the string that were found and the number of lines examined that contained occurrences of the string. If STAY OFF is in effect, the last line examined by the

COUNT command becomes the focus line. With STAY ON, the focus line location is not affected by the COUNT command.

COUNT with no operands reissues the last COUNT command issued from the command line.

When issued from a macro, the COUNT command sets the macro variable COUNT.0 to 2, returns the number of occurrences counted in COUNT.1, and returns the number of lines containing at least one occurrence in COUNT.2.

Examples

```
COUNT /a/
```

KEDIT counts the number of occurrences of the letter ‘‘a’’ in the focus line. Depending on the setting of the third operand of SET CASE, only lowercase ‘‘a’’s or both lowercase and uppercase ‘‘a’’s are counted.

```
COUNT /Mozart/ ALL
```

KEDIT counts all occurrences of ‘‘Mozart’’ in your file.

```
NOMSG COUNT /Mozart/ ALL
```

This command would be useful if issued from within a macro. Preceding the COUNT command with NOMSG will suppress any messages to the screen, but the macro variable COUNT.1 will be set to the number of occurrences of ‘‘Mozart’’ in your file, and COUNT.2 will be set to the number of lines involved.

COVERLAY

Format

COVerlay *text*

Description

The COVERLAY command (‘‘column overlay’’) overlays the text in the focus line, starting at the focus column, with the *text* you specify.

A single blank separates COVERLAY from the ‘‘overlay text’’ that will overlay data in the focus line. Wherever there is a blank in the overlay text, the corresponding character in the focus line is not changed. Wherever there is an underscore character (‘‘_’’) in the overlay text, the corresponding character in the focus line is replaced by a blank. All other characters in the overlay text replace the corresponding characters in the focus line.

Assume that the focus column is at column 1 of your text and that the focus line looks like this:

```
ABCDEFGHIJ
```

If you then enter the command

```
COV 12_456␣89
```

the focus line will be changed to

```
12 456G89J
```

There is no way to put underscore characters ("_") into the overlay text without having them converted to blanks in the focus line.

The CREPLACE command does the same thing as the COVERLAY command, except that all characters in the text you specify with CREPLACE are placed into the focus line, without the special behavior that blanks and underscores have when used with the COVERLAY command.

You can use the COVERLAY command in connection with the REPEAT command to overlay text in a group of lines.

See also CINSERT, CREPLACE, OVERLAY

Examples Assume that the focus column is column 1.

```
COVERLAY |     |     |     |
```

This command would put bars in columns 1, 4, 7, and 10 of the focus line and leave the other characters of the focus line unchanged. The COVERLAY command, used in connection with the REPEAT command, can be used to help draw boxes and sidebars next to and around existing text.

CREPLACE

Format **CReplace *text***

Description The CREPLACE command ("column replace") replaces characters in the focus line, starting at the focus column location, with the specified *text*.

For example, assume that the focus line is:

```
To be or has to be
```

If you issue the command

```
CL :10
```

the focus column will be at the "h" of "has", which is in column 10. If you then issue

```
CREPLACE not
```

the "not" will be placed in columns 10 through 12, replacing "has":

```
To be or not to be
```

KEDIT replaces characters in the focus line with the text given with the CREPLACE command, from the character after the blank that follows "CREPLACE" through the end of the command string, which may contain trailing blanks.

The CREPLACE command does not change the focus column or focus line location.

See also CINSERT, COVERLAY

Examples

```
CREPLACE 1234
```

KEDIT replaces text, starting at the focus column, with ''1234''.

```
CREPLACE ABC␣␣
```

KEDIT replaces text, starting at the focus column, with ''ABC'' followed by two blanks.

CURSOR

Format

```
CURsor Screen line [col]
CURsor Screen UP|DOWN|LEFT|RIGHT
CURsor [Escreen] line [col]
CURsor [Escreen] UP|DOWN|LEFT|RIGHT
CURsor CMdline [col]
CURsor Column
CURsor File line [col]
CURsor Home
CURsor REVERT
```

Description The CURSOR command, used mainly within macros, positions the cursor within the current window. (Use SOS TABCMDF or SOS TABCMDB to move the cursor to a different window.)

CURsor Screen *line* [*col*]
Positions the cursor at a specific *line* and *col* of the document window; the upper left corner of the document window is line 1 column 1. If the specified position is not valid for the cursor (for example, the cursor would move to a reserved line or beyond the boundary of the window), an error occurs. *Line* and *col* can be specified as an absolute number or as an equal sign (''='), where a number indicates the line or column to which the cursor should move, and an equal sign indicates that the line or column position of the cursor should not change.

CURsor Screen UP|DOWN|LEFT|RIGHT
Moves the cursor one row or column in the indicated direction. If you try to move UP or DOWN past the highest or lowest possible line of the document window, the cursor wraps to the lowest or highest possible line. If you try to move LEFT or RIGHT past the edge of the document window, the cursor moves to the first character on the next line or to the last character on the previous line.

CURsor [Escreen] *line* [*col*]

For CURSOR ESCREEN, *line* and *col* are specified as absolute numbers or as equal signs, in the same way as they are for CURSOR SCREEN, but the lines and columns are numbered differently: The command line is line 0. The first line of the document window that can contain a file line is line 1, the second line of the document window that can contain a file line is line 2, etc. (That is, the ID line, tab line, scale line, etc., are not counted in the numbering.) If the line or column specified is beyond the boundary of the document window, the line or column number is adjusted to the document window boundary. If the line that the cursor would end up on is above the top-of-file line or below the end-of-file line, the cursor is moved to the top-of-file line or end-of-file line. Column numbering starts with the first column of the document window that can contain text and ends with the last column that can contain text; the prefix area is not counted.

CURsor [Escreen] UP|DOWN|LEFT|RIGHT

Moves the cursor one row or column in the indicated direction, possibly causing vertical or horizontal scrolling if you try to move the cursor beyond the boundary of the file area.

CURsor CMdline [*col*]

Moves the cursor to the specified column of the command line. If *col* is omitted, the cursor moves to column 1 of the command line.

CURsor Column

Moves the cursor to the column pointer column of the focus line.

CURsor File *line* [*col*]

Moves the cursor to the specified *line* and *col* of the file. If the line and column of the file that you specify are not currently displayed in the document window, KEDIT will not scroll the window to bring them into view. Instead, an error will occur.

CURsor Home

Moves the cursor to the first column of the command line if you issue it when the cursor is not on the command line. If you issue it when the cursor is on the command line, the cursor moves to the line (and, if possible, column) of the file that the cursor was on when the cursor was last in the file area. If the line of the file involved is no longer in the document window, the cursor moves to the current line. CURSOR HOME is assigned by default to Shift+F12.

CURsor REVERT

A rarely-used operation used to make CUA scrolling conventions work out in macros like the default Del and Bksp macros. If INTERFACE CUA is in effect, the cursor retains its position in the file when you use the scroll bars to scroll the file, and the cursor may scroll out of the document window. When this happens, the next non-scroll bar action normally forces the cursor back into the window at the nearest visible location in the file. CURSOR REVERT instead forces the file to be repositioned in the document window as it was before scrolling began, with the cursor in its original position, undoing the effect of the scrolling. KEDIT internally issues CURSOR REVERT commands before execution of the TEXT, EXTEND, and CLIPBOARD commands.

See also SOS

Examples

```
CURSOR SCREEN 8 7
```

The cursor moves to line 8 and column 7 of the document window, where the upper left corner of the document window is line 1 column 1. An error occurs if this is not a valid cursor position.

```
CURSOR ESCREEN 8 7
CURSOR 8 7
```

These are equivalent, and move the cursor to the eighth line of the window that can contain a line of your file, and to the seventh column of the file area.

```
CURSOR ESCREEN = 9
CURSOR = 9
```

These are equivalent, and move the cursor to the ninth column of the file area of whatever line the cursor is on.

```
CURSOR FILE 29 44
```

This moves the cursor to line 29, column 44 of your file. If line 29, column 44 of your file is not displayed in the current window, an error occurs.

DEBUG

Format

DEBUG [/*tracesetting*] *macroname* [*text*]
DEBUG START *macroname*
DEBUG STOP *macroname*

Description

The DEBUG command lets you control which KEXX macros have tracing in effect when they begin execution. This lets you trace the execution of a KEXX macro that is not working properly without the need to edit the text of the macro to insert (and then later remove) TRACE instructions.

DEBUG [/*tracesetting*] *macroname* [*text*]

This form of the DEBUG command is like the MACRO command, in that you specify the name of a macro to run (*macroname*) and an optional argument string (*text*) to pass to the macro. The difference is that the debugging window will be turned on if it is off and tracing will be in effect when execution of the macro begins. KEDIT will use the initial *tracesetting* that you can optionally specify, or it will use the default trace setting controlled by SET DEBUGGING (which is normally +R, for interactive tracing of all clauses and expression results).

DEBUG START *macroname*

It is sometimes difficult to issue the DEBUG command for a macro that is not working correctly. For example, if you have a macro assigned to Alt+A that only fails if that key is pressed with the cursor on the current line, moving the cursor to

the command line to issue DEBUG ALT+A will not work. The second form of the DEBUG command can be used to deal with this problem.

Macroname must be an in-memory macro, and it is added to an internal list maintained by KEDIT. Whenever that macro begins execution, the trace setting controlled by SET DEBUGGING (normally +R) will be put into effect.

Using our example, if Alt+A is failing only when it is pressed with the cursor on the current line, you could issue the command

```
DEBUG START ALT+A
```

Then you can move the cursor to the current line and press Alt+A. The ALT+A macro will then run with the debugger active.

DEBUG STOP *macroname*

Cancels the effect of DEBUG START. In our example, when you no longer want to trace execution of ALT+A, you can enter the command

```
DEBUG STOP ALT+A
```

See also

User's Guide Section 10.4, "Debugging KEXX Macros", PROFDEBUG initialization option, MACRO, SET DEBUGGING, KEXX TRACE instruction

Examples

```
DEBUG COUNTER
```

If the debugging window is not already on, KEDIT turns it on. Then KEDIT puts the level of tracing defined by SET DEBUGGING (which normally specifies +R, for interactive tracing of clauses and results) into effect and runs the macro COUNTER.

```
DEBUG /+I COUNTER
```

If the debugging window is not already on, KEDIT turns it on. Then KEDIT puts a tracing level of +I (which means interactive tracing of clauses and all intermediate and final expression results) into effect and runs the macro COUNTER.

```
DEBUG START F5
```

KEDIT adds the F5 macro to an internal list of macros that will run with the level of tracing specified by SET DEBUGGING (normally +R) in effect. Whenever F5 is pressed, assuming the debugging window is on, KEDIT will trace it as it executes.

DEFINE

Format

DEFine *macroname macrodefinition*
DEFine *macroname*
DEFine *fileid*

Description

Use the DEFINE command to give KEDIT the definitions of in-memory macros. The macros can then be executed via the MACRO command or, when the name of a macro corresponds to KEDIT's name for a key on your keyboard, by pressing that key.

DEFine *macroname macrodefinition*

This form of the DEFINE command tells KEDIT to define a macro called *macroname* and to assign to it the specified *macrodefinition*. This method is useful only for short macros whose definition can fit completely on a single line.

DEFine *macroname*

When you give a *macroname* but no definition for it, KEDIT displays the current definition of *macroname*, which should be the name of an in-memory macro. This form of the DEFINE command is provided as a shorthand equivalent for the QUERY MACRO command.

DEFine *fileid*

With this form of the DEFINE command KEDIT loads macro definitions, which may be many lines long, into memory from the disk file specified by *fileid*.

If *fileid* has an extension of .KML, the file can contain the names and definitions of many macros. The file must be in KEDIT Macro Library format, which is discussed in User's Guide Chapter 10, "Using Macros". KEDIT searches for .KML files in the same places that it searches for .KEX files, which are discussed next.

If *fileid* has an extension of .KEX (or any extension other than .KML), the file must contain the definition of a single macro, with the filename component of the specified *fileid* taken as the name of the macro being defined.

If the macro is not in memory, KEDIT reads the macro in from a disk file. If *fileid* contains a drive or path specification, KEDIT reads the macro from the specified drive or directory. Otherwise, KEDIT looks for the macro in your current directory. If this fails, KEDIT looks in any directories that you have specified via SET MACROPATH. Finally, it looks in the "KEDIT Macros" subdirectory of your Windows Documents or My Documents folder, the directory from which KEDIT for Windows was loaded, and the USER and SAMPLES subdirectories of that directory. If the macro still cannot be located, an error message is issued and the MACRO command fails.

As discussed in User's Guide Section 10.2.3, "Storing Your Macros", we normally recommend that macros that you create be kept in the "KEDIT Macros" subdirectory of your Windows Documents folder (which is sometimes known as the My Documents folder).

See also User's Guide Chapter 10, "Using Macros", MACRO, MACROS, PURGE, SET MACROPATH

Examples

```
DEFINE F1 do I = 1 to 20; 'input' I; end
```

This DEFINE command assigns to the F1 key a macro that inputs the numbers from 1 through 20 into your file.

```
"DEFINE F1 do I = 1 to 20; 'input' I; end"
```

This is how the DEFINE command from the first example might look if you included it in your PROFILE.KEX file. Since in this case the DEFINE command is a literal string contained in a macro, it should be enclosed in quotes, and since the string contains single quotes, double quotes are used.

```
DEFINE ESCAPE 'QUIT'
```

This defines a macro called ESCAPE which simply issues a QUIT command. Note that ESCAPE is not the name of a KEDIT key. You may have intended to assign this macro to the ''escape'' key on the keyboard, but ESC is the correct name to use for that key, so the escape key would not have been redefined. (See Chapter 7, "Built-in Macro Handling", for a discussion of the key names used by KEDIT.) This type of mistake can sometimes lead to confusion, because while ESCAPE is not the name of any KEDIT key, you have issued a valid DEFINE command for a macro called ESCAPE that could be run with the command MACRO ESCAPE.

```
DEFINE RETRY
```

Since only a macro name is specified here, but no definition is given for it, KEDIT displays the current definition of a macro called RETRY.

```
DEFINE KEYS.KML
```

KEDIT assumes that KEYS.KML is a file in KEDIT Macro Library format, and loads into memory all the definitions the file contains.

```
DEFINE RETRY.KEX
```

KEDIT defines an in-memory macro called RETRY, reading its definition from the file RETRY.KEX.

DELETE

Format **DELete [*target*]**

Description Use the DELETE command to remove text from the file you are editing. All text in the specified *target* area is deleted. If you don't specify a target, only the focus line is deleted.

By default, Alt+D issues the related SOS LINEDEL command, which deletes one line from your file. The default Alt+G definition, as well as the Delete button on the bottom

toolbar, issues the DELETE BLOCK command to delete the currently marked block. If INTERFACE CUA is in effect, you can also delete a block by pressing the Del key immediately after marking the block with the mouse. If you unintentionally delete too much text from your file, you can use KEDIT's undo facility to try to recover it.

See also SOS LINEDEL

Examples

```
DELETE
```

Deletes the focus line.

```
DELETE 5
```

Deletes the focus line and the four lines following it, for a total of five lines.

```
DELETE *
```

Deletes all lines from the focus line through the end of the file.

```
DELETE ALL
```

Deletes all lines of the file.

DIALOG

Format **DIALOG** ***/prompt/*** **[*options*]**

where *options* can be:

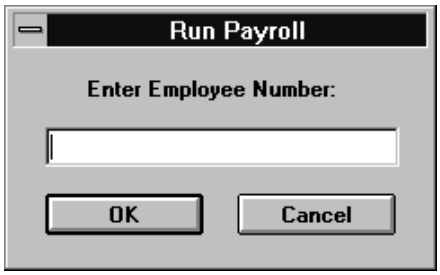
```
EDITfield [/initial/]
TITLE /title/
OK|OKCANCEL|YESNO|YESNOCANCEL
DEFButton n
ICONExclamation|ICONInformation|ICONQuestion|ICONStop
FIXEDfont
PASSWORD
```

Description Use the DIALOG command within KEDIT macros to display a simple dialog box that presents messages to, and optionally obtains input from, the user of the macro.

Here is how the DIALOG command might appear in a macro:

```
'DIALOG /Enter Employee Number:/ TITLE /Run Payroll/ EDITFIELD'
```

The resulting dialog box would look like this:



There are five areas in the dialog box that you can control:

An optional title field, specified by the TITLE operand of the DIALOG command, is displayed on the border at the top of the dialog box.

A prompt, specified as the first operand to the dialog command, is always displayed. You can use it to prompt a user for input or simply to display a message to the user.

An optional edit field, controlled by the EDITFIELD option of the DIALOG command, allows the user of your macro to enter data.

Mouse- and keyboard-selectable buttons, which can display choices like Yes/No or OK/Cancel, are used to end display of the dialog box. (When the EDITFIELD option is used, as it is in our example, OK and Cancel buttons are automatically supplied.)

A bitmapped icon, such as a question mark, can be displayed as part of the dialog box.

Here is a description of each of the possible operands for the DIALOG command. The /*prompt*/ operand is required; all of the others are optional.

`/prompt/`
The specified prompt text, which must be enclosed in delimiters, is displayed in the prompt field of the dialog box. If the text of the prompt is too wide for a dialog box, KEDIT automatically wraps the text across multiple lines, splitting it at word boundaries. KEDIT also starts a new line wherever the prompt contains a carriage return (character code 13), a linefeed (character code 10), or a carriage return-linefeed pair.

`TITLE /title/`
The specified title, which must be enclosed in delimiters, is displayed on the border at the top of the dialog box.

`EDITfield [/initial/]`
The EDITFIELD option tells KEDIT to display an editable field in the dialog box, so the user can enter a line of input for your macro. You can optionally specify, in delimiters, text to be used as the initial contents of the edit field. When the DIA-

LOG command completes, the contents of the edit field are returned to your macro in the variable DIALOG.1.

OK|OKCANCEL|YESNO|YESNOCANCEL
These options control which buttons are displayed at the bottom of the dialog box: "OK"; both "OK" and "Cancel"; both "Yes" and "No"; or "Yes", "No", and "Cancel". The default, unless the EDITFIELD option is present, is OK. If EDITFIELD is present, OKCANCEL is the default and the only allowed choice. When the DIALOG command completes, the text of the button used to close the dialog box is returned, in uppercase, in the variable DIALOG.2.

DEFButton *n*
DEFBUTTON, which defaults to 1 (for the leftmost button), controls which of the buttons at the bottom of the dialog box is initially highlighted and is returned in the variable DIALOG.2 if the user of the macro does not select some other button.

ICONExclamation|ICONInformation|ICONQuestion|ICONStop
These options control what icon appears in the dialog box.

By default, if the YESNO or YESNOCANCEL options are used, a question mark is displayed. Otherwise, the default action is to display no icon.

FIXEDfont
The prompt text is normally displayed in a proportional font; the FIXEDFONT option causes it to display in a fixed-pitch font. FIXEDFONT is useful if there are multiple lines of text in the prompt and you want certain columns of the text to line up.

PASSWORD
PASSWORD causes data entered into the resulting dialog box to be masked, so that potentially sensitive data is not visible on the screen. The PASSWORD operand is only valid when the EDITFIELD operand is also used, and it must come after the EDITFIELD operand.

DIALOG returns its results through macro variables, much as the EXTRACT command does. Three variables are set:

dialog.0	2
dialog.1	The contents of the edit field when the dialog box was closed, or the null string if no edit field was displayed.
dialog.2	The text, in uppercase, of the button selected when the dialog box was closed.

See also User's Guide Chapter 11, "Sample Macros", ALERT, POPUP, READV, SHOWDLG

DIRAPPEND

Format

DIR [*filespec* ...]
DIRAppend [*filespec* ...]

Description

The KEDIT DIR command is similar to the DOS DIR command. When you issue the DIR command, KEDIT creates a file in the PC's memory containing directory information about the files you specify. This file, called DIR.DIR, becomes the current file. The file is normally never written to disk. Instead, you work with the directory information and then press function key F3 to QUIT from the file to remove it from memory. If you use File Close to close a DIR.DIR file, it is removed from memory but not written to disk even if it has been modified.

If you issue the DIR command with no *filespec* operand, KEDIT lists information for all files in the current directory. Otherwise, *filespec* gives the file name and extension, and optionally drive and path specifier, of the files you are interested in. Asterisks ("*") and question marks ("?") act as wildcard characters, as they do for DIR commands issued from the Windows command prompt. As shown in an example below, the DIR command can process more than one *filespec* at a time.

KEDIT sorts the contents of the DIR.DIR file into alphabetical order based on the name and extension of the files involved. You can use the SET DEFSORT command to change this default sort order. You can also re-sort a DIR.DIR file into a different order by using the DIRSORT command.

You can use the SET DIRFORMAT command to control how many columns of the DIR.DIR file are set aside for the file name and for the file extension. By default, KEDIT sets aside 30 columns for the file name and 10 columns for the file extension.

With the default of FCASE ASIS in effect, fileid components in lowercase or in mixed case are displayed as is, but fileid components that are entirely in uppercase are displayed in lowercase, since this is generally easier to read.

Note that the maximum file size that the DIR command will display is 2**32-1 (that is, 4294967295), regardless of the actual size of very large files.

With the default key definitions, you can edit one of the files listed in the DIR.DIR file by placing the cursor on the line describing the file you want to edit and pressing Alt+X.

If you place the cursor on a line in the DIR.DIR file that describes a directory, as opposed to a file, Alt+X will replace the contents of the DIR.DIR file with a listing of the specified directory. While Alt+X lets you move down in the directory tree, Shift+Ctrl+X lets you move up in the tree. Shift+Ctrl+X replaces the contents of DIR.DIR with a listing of the parent directory of the file described on the cursor line.

You can also use the mouse to work with the DIR.DIR file. Double clicking button 1 on a line in a DIR.DIR file works like Alt+X does: if the line describes a file, KEDIT begins editing the file; if the line describes a directory, KEDIT gives you a listing of the directory. The Parent Directory button on the DIR.DIR toolbar works like

Shift+Ctrl+X, giving you a listing of a parent directory. Other DIR.DIR toolbar buttons let you sort a directory file by name, extension, size, or date.

If DIR.DIR is already in the ring when you issue the DIR command (because of a DIR command issued earlier in your KEDIT session) its contents will be replaced.

The DIRAPPEND command does exactly the same thing as the DIR command, except that if DIR.DIR is already in the ring when you issue the DIRAPPEND command, KEDIT does not replace its contents but instead adds information about the files you specify to the existing contents of DIR.DIR and then re-sorts the entire file in the order specified by SET DEFSORT.

Since DIR and DIRAPPEND can cause a file (DIR.DIR) to be added to the ring, the PROFILE, NOPROFILE, PROFDEBUG, and NOMSG options, as discussed in Chapter 2, "Invoking KEDIT", can also be given with these commands.

See also DIRSORT, SET DEFSORT, SET DIRFORMAT

Examples

```
DIR
```

A directory listing of your current directory is placed in a file in memory called DIR.DIR, which becomes the current file.

```
DIR D:X*.PAS
```

A directory listing of all files in the current directory of your D: drive whose name starts with "X" and whose extension is .PAS is placed in DIR.DIR.

```
DIR \
```

A directory listing of all files in the root directory of the current drive is placed in DIR.DIR.

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

A directory listing of all files in the current directory with an extension of .C, .H, or .ASM is placed in DIR.DIR.

```
DIRAPPEND *.COM
```

A directory listing of all files in the current directory with an extension of .COM is added to DIR.DIR, which is then re-sorted in the order specified by SET DIRSORT.

DIRSORT

Format `DIRSORT [Date|Extension|Name|Path|Size ...]`

Description The DIRSORT command lets you re-sort the contents of the DIR.DIR file based on the fields you specify. (The DIR command puts a directory listing into a file called DIR.DIR. It then sorts the listing in the order specified by the SET DEFSORT

command; by default the sort is done according to the name and extension of the files listed.)

The DIRSORT command is valid only if issued while the current file is the DIR.DIR file or is some other file with an extension of .DIR.

DIRSORT Date
Sorts DIR.DIR according to the date and time of each file, with the newest files listed first.

DIRSORT Extension
Puts DIR.DIR into alphabetical order according to the file extension.

DIRSORT Name
Puts DIR.DIR into alphabetical order according to the file name.

DIRSORT Path
Puts DIR.DIR into alphabetical order according to the drive and subdirectory of each file.

DIRSORT Size
Orders DIR.DIR according to the size of each file, with the largest files listed first.

These operands can be combined. For example,

```
DIRSORT EXTENSION SIZE
```

sorts DIR.DIR according to file extension and, if several files have the same extension, sorts these by size.

The toolbar for DIR.DIR files provides buttons that use the DIRSORT command to sort your file by file name, file extension, size and date.

See also DIR, DIRAPPEND, SET DEFSORT

DMSG

Format **DMSG [*text*]**

Description The DMSG (“debug message”) command displays the specified *text* in the debugging window. The command has no effect if DEBUGGING OFF is in effect.

The DMSG command is supplied as a macro debugging aid. You can display debugging information in the debugging window while your macro is executing to avoid interference with the normal KEDIT display.

See also DEBUG, MSG, SET DEBUGGING

Examples

```
DMSG Entering outer loop
```

If DEBUGGING ON is in effect, KEDIT displays ''Entering outer loop'' in the debugging window.

DOS, DOSNOWAIT, DOSQUIET

Format

DOS [*command*]
DOSNowait *command*
DOSQuiet *command*

Description

Use KEDIT's DOS command to run DOS commands from within KEDIT. You can run utilities, compilers, and most other DOS programs. After your commands have executed, you return to KEDIT and can resume editing where you left off.

The DOS command with no operands opens a DOS window and passes control to it, with KEDIT's window temporarily removed from your display. You remain in DOS, able to issue any number of DOS commands, until you return to KEDIT by issuing the EXIT command.

When you have a single command to execute, you can enter the command as an operand to KEDIT's DOS command. KEDIT executes the command in a DOS window. After the command has completed, press any key to return from the DOS window to KEDIT.

Using the DOSNOWAIT and DOSQUIET commands is similar to using the DOS command and specifying the command that you want to execute. The three commands differ in how they handle the DOS window that they create:

When you issue the DOS command and specify an operand as a command to be passed to DOS, the DOS window created to run the command displays the message ''Press any key to continue'' upon completion of the command and does not return control to KEDIT until you press a key. This ensures that you can see any output from that command before the DOS window disappears. KEDIT's window is removed from your display while the DOS window is active.

The DOSNOWAIT command creates a DOS window and runs the command that you specify. KEDIT's window is removed from your display while the DOS window is active. But after the command completes, the DOS window disappears and control immediately returns to KEDIT, without waiting for you to press a key.

The DOSQUIET command creates the DOS window as a minimized window, so that you normally do not see any output from your command even as it is executing. While the command is executing, KEDIT's window will remain on your screen but KEDIT will be waiting for the command to complete. If KEDIT receives any keystrokes or mouse clicks while the command is executing, it will restore the DOS window to its normal size on your display. When the command finishes executing, the DOS window is closed and control returns to KEDIT.

Use the WINEXEC command, and not the DOS, DOSNOWAIT, or DOSQUIET commands, to start another Windows program from within KEDIT. These commands can start other Windows programs, but only by first creating a DOS command window, and the WINEXEC command avoids this extra overhead.

The command string that KEDIT passes to DOS can be up to 4096 characters long, but because of certain parameters that KEDIT itself must include in this command string, the practical limit for the length of the command you issue via DOS, DOSNOWAIT, or DOSQUIET is approximately 4000 characters.

EXITCODE

When issued from within a macro the DOS and DOSQUIET commands, along with the WINEXEC WAIT command, return information about the exit code set by the command that was executed. (DOSNOWAIT and WINEXEC NOWAIT do not return an exit code.) The exit code is only returned if the DOS/DOSQUIET/WINEXEC WAIT command itself gets a return code of 0 (that is, it sets the macro variable RC to 0).

The exit code is a 32-bit value that is returned in the following macro variables:

`exitcode.0` 3

`exitcode.1` The exit code from the command, as a signed decimal number

`exitcode.2` The exit code from the command, as an unsigned decimal number

`exitcode.3` The exit code from the command, as a character string containing the 8 hexadecimal digits of the 32-bit exit code

For example, if you use the DOS command to start a command shell and you exit from that command shell by typing “EXIT -2”, the following variables would be set:

`exitcode.0` 3

`exitcode.1` -2

`exitcode.2` 4294967294

`exitcode.3` FFFFFFFE

If you start a DOS command prompt and close the command prompt window by double-clicking on its close button, or if you use Ctrl+C to interrupt a command that does not have its own handler for Ctrl+C, the exit code returned will be C000013A, which indicates an unhandled exception, yielding

`exitcode.0` 3

`exitcode.1` -1073741510

`exitcode.2` 3221225786

`exitcode.3` C000013A

Note that exitcode.1 and exitcode.2 in this example, along with exitcode.2 in the preceding example, are 10-digit values that are subject to rounding if you do arithmetic operations on them with the default NUMERIC DIGITS value of 9 in effect.

See also WINEXEC

Examples

```
DOS TYPE ABC.TXT
```

The command TYPE ABC.TXT will be executed in a Command Prompt window. After the TYPE command completes, the results will remain displayed in the DOS window until you press any key. You will then return to your KEDIT session.

```
DOSQ MKDIR \TEMP
```

MKDIR \TEMP will execute in a minimized DOS window, and KEDIT's window will remain on the display while it executes. When the command has completed, the minimized DOS session will end immediately, without waiting for you to press a key. The DOSQUIET command is useful for execution of commands like MKDIR, which usually generate no output.

```
DOS
```

Shells to a Command Prompt window, where you can issue a series of commands. To return to KEDIT, issue the EXIT command.

DOWN

Format **Down [*n*]**

Description The line *n* lines below the focus line becomes the new focus line. If *n* is not specified, the line one line below the focus line becomes the focus line.

The NEXT command and the LOCATE command with a relative line number target perform the same function as the DOWN command.

See also UP

Examples

```
DOWN
```

The line that is one line below the focus line becomes the new focus line.

```
DOWN 4
```

The line that is four lines below the focus line becomes the new focus line.

DUPLICATE

Format **DUPlicate [*n* [*target*]]**

Description Use the DUPLICATE command to duplicate one line or a group of lines *n* times. If no *target* is given, the focus line is duplicated. If a *target* is given, all lines in the target area are duplicated *n* times. If *n* is not specified, the focus line is duplicated once.

The first line added to the file by the duplication process becomes the new focus line.

The DUPLICATE command is assigned by default to function key F8 and to Alt+=.

Examples

```
DUP
```

The focus line is duplicated once. The newly-added line is inserted below the focus line and becomes the new focus line.

```
DUPLICATE 6
```

Duplicate the focus line six times.

```
DUP 4 6
```

The focus line and the five lines after it, for a total of six lines, are duplicated four times.

```
DUP 2 BLOCK
```

The currently marked line block is duplicated twice.

EDITV

Format

```
EDITV GET   varname1 [varname2 ...]
EDITV PUT   varname1 [varname2 ...]
EDITV SET   varname1 value [varname2 value ...]
EDITV SETL  varname value
EDITV LIST  [varname1 ...]

EDITV GETF  varname1 [varname2 ...]
EDITV PUTF  varname1 [varname2 ...]
EDITV SETF  varname1 value [varname2 value ...]
EDITV SETLF varname value
EDITV SETFL varname value
EDITV LISTF [varname1 ...]
```

Description

You may occasionally have a set of macros, executed at different times during a KEDIT session, that need to share information. The variables normally used within KEDIT macros (that is, KEXX variables) don't help in this situation, because they are local to the macro and their values are not retained after the macro finishes executing. The EDITV command, designed for use mainly from within KEDIT macros, provides a solution to this problem, allowing values determined in one KEDIT macro to be accessed by KEDIT macros that you run later. EDITV manipulates two classes of variables:

Edit variables, which are global to the editor and retain their values throughout a KEDIT session.

Multiple sets of file variables, one set for each file in the ring. File variables are specific to a particular file and retain their values until you remove the file from the ring with a FILE or QUIT command.

The EDITV command sets, retrieves, or lists these variables.

EDITV SET, which you can use either from within a KEDIT macro or from the command line, takes one or more pairs of variable names and values and for each pair assigns the specified edit variable the specified value. Variable names and values are separated by blanks, and the values therefore cannot contain blanks. For example,

```
EDITV SET X 17
```

sets the edit variable X to "17", while

```
EDITV SET X 17 Y 19 Z
```

sets the edit variable X to "17", Y to "19", and Z to a null string.

EDITV SETL takes the name of an edit variable and assigns to it a given value, which can contain blanks. For example,

```
EDITV SETL S This is a sentence.
```

sets the edit variable S to the string ‘‘This is a sentence.’’

EDITV PUT provides a direct method for assigning the values of one or more KEXX macro variables to edit variables of the same name. For example, consider the following KEDIT macro:

```
A = 18 * 2
B = 'Hello'
'EDITV PUT A B'
```

The first line sets the macro variable A to ‘‘36’’ and the second line sets the macro variable B to ‘‘Hello’’. The third line then sets the edit variable A to have the same value as the macro variable A, which is ‘‘36’’, and sets the edit variable B to have the same value as the macro variable B, which is ‘‘Hello’’. After the macro has finished, the macro variables A and B are gone, but the edit variables A and B remain, and can be accessed by subsequent EDITV commands.

EDITV GET does the opposite of what EDITV PUT does. It is used within a KEDIT macro to set the values of one or more macro variables to the values of edit variables of the same name. For example, assume that the macro in the above example had been run, and then the following macro was run:

```
'EDITV GET A B'
'INPUT' A B
```

This macro would retrieve the values of edit variables A and B, placing these values into the macro variables A and B. The macro would then input the text ‘‘36 Hello’’ into your file.

Note that EDITV GET and EDITV PUT are valid only when used within KEDIT macros, and cannot be used from the command line. Note also that variable names referred to when you use EDITV GET or EDITV PUT should normally appear within quoted strings; otherwise, the names of the variables may not be properly passed to the EDITV command.

EDITV LIST displays the values of one or more edit variables or (if no variable names are given) of all edit variables with non-null values.

If you attempt to GET or LIST the value of an edit variable that has not been assigned a value, a null string is returned.

EDITV GETF, EDITV PUTF, EDITV SETF, EDITV SETLF, EDITV SETFL, and EDITV LISTF follow the above pattern, except that instead of working with edit variables, they work with the file variables associated with the current file. There are multiple sets of file variables, one set for each file in the ring. (EDITV SETLF and EDITV SETFL both do the same thing; they are analogous to EDITV SETL.)

EMSG

Format **EMSG [*text*]**

Description The EMSG (“error message”) command, used mainly in macros, displays the specified *text* on the message line as an error message. If BEEP is ON, the EMSG command also causes the PC’s speaker to BEEP.

See also ALERT, CMSG, DMSG, MSG, WMSG

Examples

```
EMSG No block currently marked
```

KEDIT displays “No block currently marked” on the message line and, if BEEP is ON, the speaker beeps.

ERASE

Format **ERASE *fileid***

Description KEDIT’s ERASE command is similar to the DOS ERASE command. You give the file specification for the file you want erased from a disk. This involves the file name, extension, and optionally the drive and path specification. The specified file is erased from your disk.

The KEDIT ERASE command erases files on disk and has no effect on the copies in the PC’s memory of files currently being edited. It is less powerful than the DOS ERASE command, because it does not accept wildcard characters (“*“ and ”?”). You can use KEDIT’s DOS command to issue the “real” DOS ERASE command.

See also DOS, RENAME

Examples

```
ERASE C:ABC.PAS
```

The file ABC.PAS, in the current directory of the C: drive, is erased.

```
ERASE \SONGS\MUSIC.BAS
```

The file MUSIC.BAS, in the \SONGS directory of the current drive, is erased.

EXPAND

Format **EXPand [*target*]**

Description The EXPAND command expands tab characters in your text into strings of blanks. If you use the EXPAND command with no operands, KEDIT expands the tab characters in the focus line. Otherwise, KEDIT expands the tab characters in all lines of the specified *target* area.

To expand a line, the EXPAND command goes through the line looking for tab characters (character code 9). Whenever a tab character is encountered, it is replaced by a string of blanks extending to the next tab position (as set by the SET TABS command).

After the expansion, the focus line location is unchanged if STAY ON (the default) is in effect; otherwise, the last line expanded becomes the new focus line.

The EXPAND command is most often used in connection with the COMPRESS command. An example of compression and expansion is given in the discussion of the COMPRESS command.

The COMPRESS and EXPAND commands are rarely used. More frequently used are the SET TABSIN and SET TABSOUT commands, which control whether KEDIT automatically expands tab characters in a file to blanks when reading the file in from disk and compresses blanks to tabs when writing the file out to disk.

See also COMPRESS, SET TABS, SET TABSIN, SET TABSOUT

Examples

```
EXPAND ALL
```

All tab characters in all lines in the file are expanded according to the current tab settings.

```
EXPAND 2
```

Tab characters in the focus line and the line below it, for a total of two lines, are expanded according to the current tab settings.

EXTEND

Format **EXTEND *command***

Description EXTEND is a specialized command used mainly in the macros assigned to Shift+cursor-pad-keys that let you extend selections when INTERFACE CUA is in effect.

The EXTEND command extends the selection from its current anchor point (or, if there is no current selection, from the cursor position) to a new location. To determine the new location, KEDIT executes the *command*; the selection is extended to the position occupied by the cursor after the command completes. For example,

```
EXTEND SOS CRIGHT CRIGHT
```

would execute the command SOS CRIGHT CRIGHT, which would move the cursor two characters to the right. So an existing selection would be extended two characters to the right or, if there were no existing selection, a two-character selection would be marked.

If EXTEND is issued when INTERFACE CUA is not in effect, or when the cursor is in the prefix area, EXTEND executes the specified *command* but takes no other action. If the cursor is on the command line when you issue the EXTEND command, a command line selection is marked or extended; if the cursor is in the file area, a non-persistent stream block is marked or extended.

EXTRACT

Format

EXTract */operand/* **...**

Description

The EXTRACT command, valid only when issued from a macro, provides one method for a KEDIT macro to extract information from KEDIT. KEDIT returns information to your macro by setting the values of variables within your macro.

EXTRACT is fully discussed in Chapter 5, “QUERY and EXTRACT”.

FILE, FFILE

Format

FILE [*fileid***]**
FFile [*fileid***]**

Description

Use the FILE command when you have finished editing the current file and you want KEDIT to write the edited version of the file to disk. KEDIT writes the file to disk using the *fileid* that you specify, removes the file from memory, and (if you are editing multiple files) makes the previous file in the ring become the current file.

You will normally issue the FILE command without specifying the optional *fileid* operand. In this case, the file will be written to disk under its current fileid. The current fileid is displayed on the title bar of the file’s document window and, unless you have changed it with the SET FILEID command, is the fileid you originally used when you started editing the file with the KEDIT command. If you use the optional *fileid* operand, you can make use of the various shortcuts for specifying it discussed in connection with the SET FILEID command.

There is one case where the *fileid* operand is not optional. If you use the FILE command on an untitled file (the UNTITLED.*n* files used when you start KEDIT without specifying a fileid or use the File New menu item), you must tell KEDIT the *fileid* to use for the disk file that is created.

Using the File command is similar to using the File Close menu item on a modified file and, when prompted, specifying that the file should be saved. Note that the File Close menu item removes unmodified files from memory without writing them to disk, while the FILE command always writes a file to disk, even if it has not been modified.

Like the more-frequently-used FILE command, the FFILE command writes the current file to disk and then removes the current file from the ring. The difference between the commands is that there are two situations in which the FILE command will give you an error, to warn you that you may be inadvertently overwriting some data, while the FFILE command will write your file to disk regardless of the possible problem.

The first situation occurs when you begin editing a file with one fileid, but then want to write it to disk with a new fileid. If a file with the new fileid already exists, the FILE command gives you an error message, because you may not intend to overwrite the existing file.

A second situation occurs if some other process on your system, or some other user on your network, has changed the file you are editing during your editing session. KEDIT tries to determine if the file has changed by looking at the time the file was last updated and seeing if this has changed since you began editing the file (or last saved it to disk). If TIMECHECK ON, the default, is in effect, the FILE command gives you an error message if you attempt to write to a file whose timestamp has unexpectedly changed.

In both of these cases, you can use the FFILE command instead of the FILE command to indicate to KEDIT that you are aware of the situation and that you still want to write your file to disk.

See also

SAVE, SSAVE, QUIT, QQUIT, SET BACKUP, SET TIMECHECK

Examples

In these examples, assume you are editing a file called A:\PROG\SAMP.PAS and that the current default drive is C, with C:\MAIN as the current directory.

```
FILE
```

KEDIT would write the file to disk under the name A:\PROG\SAMP.PAS and then remove it from the ring.

```
FILE A:\
```

KEDIT would write the file to A:\SAMP.PAS and then remove it from the ring.

```
FILE =.CAL
```

KEDIT would write the file to disk under the name A:\PROG\SAMP.CAL and then remove it from the ring.

```
FILE C:
```

Since you are specifying a different drive for the file, but do not specify a directory, KEDIT uses the current directory of the new drive. KEDIT would write the file to C:\MAIN\SAMP.PAS and then remove it from the ring.

```
FILE SAMP3.PAS
```

KEDIT would write the file to A:\PROG\SAMP3.PAS (since you didn't specify a drive or path, the file's existing drive and path are used) and then remove it from the ring.

```
FILE \TEST\
```

KEDIT would write the file to A:\TEST\SAMP.PAS (the A:\TEST directory must already exist) and then remove it from the ring.

FILL, FILLBOX

Format

FILL [*char*]
FILLbox [*char*]

Description

The FILL command, assigned by default to Ctrl+I (and, with INTERFACE CLASSIC, to Alt+F) fills a block with copies of the specified character, overwriting the original contents of the block. If no block is marked in the current file, FILL gives you an error. If no *char* is specified, KEDIT fills the block with blanks.

The FILL command fills the entire contents of a box or stream block, from column 1 to the truncation column, with the specified character. With line blocks, FILL operates between from the left zone column to the right zone column.

FILLBOX does the same thing as FILL and exists for compatibility with older versions of KEDIT, in which the fill operation could only be performed on box blocks.

The Actions Fill dialog box or the Fill Block button on the bottom toolbar can also be used to fill a block with copies of the specified character.

If HEX ON is in effect, you can use hexadecimal or decimal notation to specify the fill character.

Examples

```
FILL
```

KEDIT replaces the contents of the currently marked block with blanks.

```
FILL x
```

KEDIT fills the currently marked block with copies of the letter ''x''.

```
FILL X'03'
```

Assuming that HEX ON is in effect, KEDIT fills the currently marked block with copies of the character with character code 3.

FIND, FINDUP, FUP

Format

Find *text*
FINDUp *text*
FUp *text*

Description

The FIND command searches forward through your file for a line that starts (in column 1) with the *text* that you specify. That line becomes the new focus line. The search starts with the line below the focus line and continues until the desired text is found or the end-of-file line is reached (wrapping to the top of the file if WRAP is ON).

FINDUP (which can also be given as FUP) does the same thing as the FIND command, except that KEDIT searches upward for the desired line, beginning with the line above the focus line.

These commands are included in KEDIT mainly for compatibility with XEDIT. The TFIND command is a more general command that is preferable in most situations.

KEDIT remembers the last operand used whenever the FIND, FINDUP, NFIND, or NFINDUP commands are issued from the command line and, if you later issue one of these commands with no operands, this ”remembered operand” is reused.

If CASE IGNORE is in effect, an alphabetic character in the search text will match either its uppercase or lowercase equivalent. Blanks in the search text act as wild card characters, matching any single character, blank or nonblank, in your file. Underscore (“_”) characters in the search text match blanks in your file.

See also

LOCATE, NFIND, NFINDUP, TFIND

Examples

```
FINDUP ABC
```

The next line that begins with “ABC” becomes the focus line.

```
FIND␣␣ABC
```

Here, FIND is followed by two blanks. The first blank separates the command FIND from the text to be searched for. The second blank is the first character of the text to be searched for. Since a blank matches any character in your file, this command will search for a line with any character in column 1 and “ABC” in columns 2 through 4.

```
FIND ABC DEF
```

This command searches for a line that has "ABC" in columns 1 through 3 and "DEF" in columns 5 through 7, with any character in column 4.

FLOW

Format

FLOW [*target*]

Description

The FLOW command reformats the text in a portion of your file. FLOW with no operands (which is assigned by default to Shift+Ctrl+F and, with INTERFACE CLASSIC, to Ctrl+F) reformats the text in the focus paragraph. FLOW with a *target* operand reformats the text in all paragraphs in the target area.

The FLOW command adjusts the text within a paragraph so that all lines of the paragraph start at the left margin column (except for the first line of the paragraph, which starts at the paragraph indent column), and all lines of the paragraph contain as much text as will fit between the left and right margins. (The left and right margins and the paragraph indent column are set with the SET MARGINS command.)

The SET FORMAT command affects other aspects of FLOW's behavior: whether paragraphs are right-justified (by sprinkling extra blanks between words on each line to force each line of the paragraph to end in the right margin column), how KEDIT determines where new paragraphs begin, and whether KEDIT places two blanks or one blank at the end of each sentence that it reformats.

When the FLOW command finishes, the line below the last line processed by the FLOW command becomes the focus line.

Notes

Be careful not to use the FLOW command in a file that does not contain a document divided into paragraphs. For example, if you issue the FLOW command in a data file or computer program, the FLOW command could reformat all of the text in the file into one large paragraph. (You could use the undo facility to try to recover from this.)

You should also be careful when using the FLOW command with a *target* operand. KEDIT will interpret all text in the target area as belonging to a paragraph, and reformat it accordingly, which is inappropriate if the target area includes titles, headings, etc.

The FLOW command works by deleting all of the lines of the paragraph involved, and inserting the reformatted lines into the file in their place. Any text in the original paragraph beyond the truncation column is lost, as are any line names associated with the lines of the original paragraph.

See also

User's Guide Section 3.11, "Word Processing Facilities", SET FORMAT, SET MARGINS

Examples

```
FLOW
```

This command is normally assigned to Shift+Ctrl+F. KEDIT reformats the paragraph in which the focus line is located.

```
FLOW ALL
```

KEDIT assumes that your file consists of a number of paragraphs that need to be reformatted, and KEDIT reformats all of them.

FORWARD

Format

FOrward [*n*|*|HALF|*m* Lines]

Description

The FORWARD command causes KEDIT to scroll the current window forward in your file.

FOrward
KEDIT scrolls one window forward in your file. This command is normally assigned to the Page Down key.

FOrward *n*
KEDIT scrolls *n* windows forward in your file, as if you had pressed the Page Down key *n* times.

FOrward *
KEDIT scrolls forward all the way to the end of your file, making the end-of-file line become the focus line.

FOrward HALF
KEDIT scrolls one half-window forward in your file.

FOrward *m* Lines
KEDIT scrolls forward *m* lines in your file.

You can also move forward and backward in your file by using the mouse to manipulate the vertical scroll bar.

See also

BACKWARD

Examples

```
FORWARD 2
```

KEDIT scrolls forward two windows in your file, as if you had pressed the Page Down key twice.

```
FORWARD 8 LINES
```

KEDIT scrolls eight lines forward in your file.

GET

Format **GET [*fileid* [*fromline* [*forlines*]]]**

Description

The GET command reads lines from the disk file you specify and inserts them into the file you are editing, below the focus line. The last line inserted becomes the new focus line. You can insert an entire disk file or, by using the optional *fromline* and *forlines* operands, you can insert any desired portion of a disk file.

fileid

The first operand of the GET command is the *fileid* of the disk file to be inserted. The disk file is not itself changed in any way; only a copy of its contents is inserted. If no path is specified, KEDIT looks in the current directory of the specified drive. If both drive and path are omitted, KEDIT looks in the current directory of the current drive and then, depending on the value of your PATH setting, does a path search for the file, and then 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.

The shortcuts for specifying a *fileid* that are discussed in connection with the SET FILEID command can also be used with the GET command.

fromline

Fromline is a numeric operand that tells KEDIT which line of the disk file is the first line you want inserted. If this operand is not given, the *forlines* operand cannot be given either, and the entire contents of the disk file, starting from line 1, are inserted into the current file.

forlines

Forlines tells KEDIT how many lines are to be inserted. If *forlines* is a number, that number of lines is inserted. If *forlines* is omitted or given as an asterisk ("*"), all lines starting from *fromline* through the end of the disk file are inserted.

GET with no operands is used to read in from disk the temporary file that the PUT command creates when you issue the PUT command without giving a fileid.

The GET command is affected by the settings of EOLIN, EOFIN, TABSIN, and TRANSLATEIN.

See also PUT, PUTD

Examples

```
GET A:ABC.TXT
```

KEDIT inserts all of ABC.TXT from the current directory of the A: drive into the current file below the focus line. After completion of the operation, the last line inserted becomes the new focus line.

```
GET A:ABC.TXT 12 1
```

Line 12 of A:ABC.TXT is inserted.

```
GET A:ABC.TXT 12
GET A:ABC.TXT 12 *
```

In both cases, KEDIT inserts text from A:ABC.TXT into the current file, starting from line 12 and continuing until it reaches the end of A:ABC.TXT.

```
GET A:\RW\ABC.TXT 1 15
```

The first fifteen lines of A:\RW\ABC.TXT are inserted into the current file.

HELP

Format

HELP [*topic*]

Description

Use the HELP command to display the KEDIT for Windows Help file.

HELP with no operands shows the table of contents for the Help file.

HELP *topic* shows the Help information for the specified topic. If help for the topic cannot be found, the Help program's Search dialog box is displayed, so that you can view the list of available Help topics.

The topics that you specify can include KEDIT commands, SET options, QUERY and EXTRACT operands, Boolean functions, KEXX instructions, and built-in KEXX functions. You can also get help for KEDIT error messages by specifying the error message number.

Examples

```
HELP
```

Display the table of contents for the KEDIT for Windows HELP file.

```
HELP CHANGE
HELP C
```

Display Help for the CHANGE command --- the Help command lets you specify legal truncations of the commands you want help for.

```
HELP SET WRAP
HELP WRAP
HELP SET WR
```

Display help for the SET WRAP command.

```
HELP QUERY ARBCHAR
HELP Q ARBCHAR
```

Display help for QUERY ARBCHAR.

```
HELP 2
HELP ERROR 2
```

Display help for KEDIT error message number 2.

```
HELP SUBSTR
HELP SUBSTR()
```

Display Help for the KEXX SUBSTR built-in function.

```
HELP SEARCH
HELP INDEX
HELP CONTENTS
```

You can also display the Search, Index, or Contents panes of the Help file. (Help Contents is equivalent to Help with no operands.)

See also SET HELPDIR

HEXTYPE

Format **HEXType**

Description The HEXTYPE command lets you see the internal character codes for the characters in the focus line.

For each character, KEDIT displays the decimal value of the code for the character, the hexadecimal value of the code for the character, and the character itself. The decimal and hexadecimal codes for a character are displayed vertically, directly above the character itself.

See also SET HEXDISPLAY, SET VERIFY

HISTUTIL

Format

```
HISTUTIL CLEAR MEMORY|REGISTRY type|ALL
HISTUTIL GET MEMORY|REGISTRY type
HISTUTIL INFO type
HISTUTIL RELOAD type|ALL
HISTUTIL SAVE type|ALL
```

Description HISTUTIL is a specialized command that can be used to query and manipulate the history information (recently-edited files, recent commands, etc.) that KEDIT maintains.

The *type* of history information involved can be any of the following:

CMDLINE	Command line history
DEBUGGER	KEXX debugger history
DIALOG	DIALOG/ALERT command EDITFIELD history
DIRECTORY	File Directory dialog history
FIND	Edit Find dialog, Quickfind, Find field of Edit Replace
RECENTFILES	File menu list of recently-edited files
REPLACEWITH	Replace With field of Edit Replace dialog
SELECTIVEEDITING	Edit Selective Editing history

The different forms of the HISTUTIL command work as follows:

HISTUTIL CLEAR

HISTUTIL CLEAR MEMORY|REGISTRY *type*|ALL

Deletes all history of the specified *type*, or of ALL types, from either MEMORY or the REGISTRY.

HISTUTIL CLEAR REGISTRY ALL is equivalent to REGUTIL CLEAR HISTORY.

HISTUTIL GET

HISTUTIL GET MEMORY|REGISTRY *type*

Valid only within macros. Retrieves the history values of the specified type, either from MEMORY or from the REGISTRY, setting the following variables:

HISTUTIL.0	*n*, where *n* is the number of history items of the type involved retrieved from memory or from the registry.
HISTUTIL.*i*	For *I* = 1 to *n*, holds the value of the *i*th item from the requested history list. The higher numbered items are the ones most recently added to the list.

HISTUTIL INFO

HISTUTIL INFO *type*

Valid only within macros. Returns information about a *type* of history list by setting these macro variables:

HISTUTIL.0	5
HISTUTIL.1	The *type* involved, in the mixed case used as the name of the section within the registry where KEDIT saves its history.
HISTUTIL.2	Number of history items of this type currently saved in memory by KEDIT.
HISTUTIL.3	Maximum number of history items of this type that KEDIT will save.
HISTUTIL.4	Maximum length, in characters, of individual history items of this type that KEDIT will save.

HISTUTIL.5	Prefix characters placed in front of values of this type saved by KEDIT. This is 3 for FIND and 0 for all other types; the notes below discuss the FIND prefix bytes.

For example, the results for HISTUTIL INFO CMDLINE might be:

HISTUTIL.0	5
HISTUTIL.1	"Cmdline" (quotes aren't part of the value returned)
HISTUTIL.2	92
HISTUTIL.3	100
HISTUTIL.4	1000
HISTUTIL.5	0

HISTUTIL RELOAD

HISTUTIL RELOAD *type|ALL*

Replaces the current in-memory history of the specified *type*, or of ALL types, with the history information of the type involved that is currently stored in the registry.

HISTUTIL SAVE

HISTUTIL SAVE *type|ALL*

Saves the current in-memory history of the specified *type*, or of ALL types, to the registry, replacing whatever history information of the type involved was previously in the registry.

HISTUTIL SAVE ALL is equivalent to REGUTIL SAVE HISTORY.

Notes

The HISTUTIL command depends to a larger extent than most KEDIT commands do on the KEDIT's internal implementation details, and is therefore more likely than most KEDIT commands are to be subject to incompatible changes in future versions of KEDIT. You should not develop macros that depend on the use of the HISTUTIL command unless you are aware of and are comfortable with this possibility.

KEDIT uses various undocumented heuristics, subject to change between versions of KEDIT, to maintain the information in its history lists. For example, duplicate items in different history lists are automatically deleted under different conditions. And the contents of some history lists can be affected by other KEDIT commands; the Find history list, for example, is in some circumstances affected by LOCATE command targets.

With the default of REGSAVE STATE HISTORY in effect, KEDIT will automatically update the history information in the Windows registry at the end of your KEDIT session, overriding the changes made to the registry by any HISTUTIL CLEAR|SAVE REGISTRY commands earlier in the session. If you want to avoid this, you can put REGSAVE STATE|NOSTATE NOHISTORY into effect.

In the Find history list, each search string is preceded by three prefix characters which depend on the options selected in the Edit Find dialog box. The first prefix

character used with FIND is 'c' or '_' depending on the Match Case option, the second is 'w' or '_' depending on the Find Whole Words Only option, and the third is 'r' or '_' depending on the Regular Expression option.

See also REGUTIL

HIT

Format **HIT *macroname1* [*macroname2* ...]**

Description The HIT command adds the macros you specify to an internal "HIT queue". Whenever KEDIT needs to process a key, it first looks to see if any macros are in the HIT queue. If there are, KEDIT doesn't actually read a key from the keyboard and process the macro assigned to that key. Instead, KEDIT processes the next macro in the HIT queue. If the HIT queue is empty, KEDIT reads from the keyboard. You can specify multiple macros in one or more HIT commands; the HIT queue can hold the names of up to twelve macros. The macros must be in-memory macros.

The HIT command is used mainly in macros and is included in KEDIT for compatibility with earlier versions of KEDIT, where it was useful for getting around some of the limitations of the key redefinition facilities. Most situations where the HIT command was used in the past are best handled now with the MACRO command.

The difference between using the MACRO command to cause a macro to be executed and using the HIT command to cause a macro to be executed is this: When you issue the MACRO command, the specified macro is immediately executed, and is effectively called as a subroutine. When you issue the HIT command, the current macro continues to run, and the HIT macro is not executed immediately, but is instead executed later, when KEDIT next needs keyboard input. Also, unlike the MACRO command, the HIT command requires that the macro be an in-memory macro; HIT will not look on disk for a macro in a .KEX file.

KEDIT does not look in the HIT queue when you use the READV or DIALOG commands, or when processing PULL instructions issued from macros, and HIT queue entries have no effect on the results of the QUERY LASTKEY command.

See also MACRO

Examples

```
HIT F2 XYZ
```

The next time KEDIT needs keyboard input, it processes the F2 macro as if you had pressed the F2 key, and then processes the macro XYZ, which must be an in-memory macro.

IMMEDIATE

Format

IMMediate *macrodefinition*

Description

The IMMEDIATE command allows you to enter the text of a one-line macro on the KEDIT command line and have the macro immediately executed by KEDIT. This is useful when you want to run an "on-the-fly" macro in a special situation, or when you want to try a short test macro to experiment with features of KEDIT macros.

Macros run via the IMMEDIATE command follow the same rules as any other KEDIT macro. The macro can contain more than one clause, with the individual clauses separated by semicolons.

See also

User's Guide Chapter 10, "Using Macros", DEFINE, MACRO, KEXX INTERPRET instruction

Examples

```
IMMEDIATE say 1234 * 5.67
```

KEDIT multiplies 1234 by 5.67 and displays the result, 6996.78.

```
IMMEDIATE do I = 3 to size.1() by 3; ':'I; 'uppercase'; end
```

Uppercases every third line of your file.

INIUTIL

Format

INIUTIL SAVE|CLEAR STATE|HISTory|SETTINGS
INIUTIL SAVE SET *option*
INIUTIL CONVERT SETTINGS
INIUTIL BACKUP CONFIG|HISTory
INIUTIL GET CONFIG|HISTORY *section name*

Description

The INIUTIL does the same thing as the REGUTIL command. See the description of the REGUTIL command for a full description of the operands involved.

(KEDIT for Windows now stores its configuration information in the Windows registry, but KEDIT for Windows 1.5 and earlier stored this information in the KEDITW.INI file. So REGUTIL is the newer name for this command, but for compatibility reasons INIUTIL remains available.)

See also

HISTUTIL, REGUTIL, SET REGSAVE

INPUT

Format `Input [text]`

Description The INPUT command causes the line of *text* that you specify to be added to your file after the focus line. The newly-added line becomes the new focus line.

If no *text* is specified (the command line consists only of the word "INPUT"), KEDIT adds a blank line to your file after the focus line, and makes the blank line become the new focus line. What KEDIT does then depends on the setting of INPUTMODE. With INPUTMODE OFF, the default, the cursor is positioned in the left margin column of the newly-added line. With INPUTMODE LINE or INPUTMODE FULL, KEDIT enters Input Mode, as discussed in the description of SET INPUTMODE. You leave Input Mode by pressing the Home key.

See also REPLACE, SET INPUTMODE

Examples

```
I Hello there.
```

A line consisting of "Hello there." is added to your file, becoming the new focus line.

```
INPUT
```

A blank line is added to your file. This line becomes the focus line. Depending on the setting of INPUTMODE, you may enter KEDIT's Input Mode.

JOIN

Format `Join [ALigned]`

Description The JOIN command joins two lines together into a single line. The JOIN command is usually issued from a macro assigned to a key. (It is assigned to Alt+J by default.) Text from the line below the focus line is joined to the focus line starting at the focus column, overlaying any text at or to the right of the focus column.

When you issue the JOIN command with no operands, text starting at column one of the line below the focus line is joined to the focus line. JOIN ALIGNED (which Alt+J normally uses) adjusts for the fact that the two lines involved may be indented from column one, and that you usually want the JOIN operation to ignore leading blanks that are present solely because of indentation. JOIN ALIGNED therefore looks at how many leading blanks the focus line has, and ignores up to that many leading blanks in the line below the focus line. (See the example below for an illustration of the difference between JOIN and JOIN ALIGNED.)

If the line that results from the join operation would be too long (that is, if the line would extend beyond the truncation column), KEDIT issues an error message and does not carry out the join.

The JOIN command does not affect the position of the focus line or focus column.

See also SPLIT, SPLTJOIN

Examples Assume that the focus line and the line below it have the following contents, with the first nonblank character of each line in column 5 and the cursor positioned at the asterisk:

```
␣␣␣␣First line's text␣*
␣␣␣␣Next line's text
```

JOIN would join all of the second line, including the leading blanks, yielding

```
␣␣␣␣First line's text␣␣␣ ␣␣Next line's text
```

Since the first line is indented by four columns, JOIN ALIGNED would ignore the four blanks at the start of the second line, resulting in

```
␣␣␣␣First line's text␣Next line's text
```

KEDIT

Format **Kedit [*fileid* ...] [(*options* [)]]**

Description Use the KEDIT command to begin editing one or more additional files. (A command called XEDIT, which performs exactly the same functions as the KEDIT command, is also available.)

You will usually specify the *fileid* of the file you want to edit. If your *fileid* contains both a drive and path specification, KEDIT looks only there for the file. If no path is specified, KEDIT looks in the current directory of the specified drive. If both drive and path are omitted, KEDIT looks in the default directory of the current drive, then does a path search controlled by the SET PATH option. 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. The shortcuts for specifying a *fileid* that are discussed in connection with the SET FILEID command can also be used with the KEDIT command.

If a file with the specified *fileid* is already being edited (that is, it is already in the ring of files being edited), that file becomes the current file. If there is no file with the specified *fileid* in the ring, the file is read in from disk and made the current file. If no file with that *fileid* exists even on disk, KEDIT adds an empty file with the given *fileid* to the ring and makes it the current file.

Note that using the KEDIT command and specifying a *fileid* is very similar to using the File Open dialog box to begin editing a file.

You can add more than one file to the ring at a time by specifying more than one *fileid* operand or by using wildcard characters in the *fileid* specification. Asterisks ("*") and question marks ("?") in a *fileid* act as wildcard characters, just as they do when you

issue DIR commands in KEDIT or from the Windows command prompt. Note that you are limited to a maximum of 500 files in the ring, and that since KEDIT loads all of the files you are editing into memory, loading a large number of files at once may be time consuming and may use up a significant amount of memory. If KEDIT is loading a set of files and encounters an error while loading one of those files, the KEDIT command is aborted and the remaining files are not processed.

If you issue the KEDIT command with no *fileid* operand, but you do specify the UNTITLED option

```
KEDIT (UNTITLED
```

KEDIT adds a new untitled file (UNTITLED.*n*) to the ring. This is equivalent to using the File New menu item.

If you issue the KEDIT command with no operands at all (this is the default definition for Shift+F4), the next file in the ring becomes the current file. If there is only one file in the ring, the KEDIT command issued with no operands has no effect. (You can use the QUERY RING command to see a list of all files in the ring.)

If you issue the KEDIT command with a minus sign ("-") as an operand

```
KEDIT -
```

the preceding file in the ring will become the current file.

You can also use the mouse to cycle through the ring of files you are editing. You can use the Next File button on the toolbar to move to the next file in the ring, and the Previous File button to move to the previous file in the ring.

When you issue the KEDIT command with the *fileid* operand to add a file to the ring, you can specify the PROFILE, NOPROFILE, PROFDEBUG, LOCK, NOLOCK, NOREG, NOINI, NOMSG, NEW, NODEFEXT, UNTITLED, and NOFILEMENU options for the command in the same way as discussed in Chapter 2, "Invoking KEDIT".

See also

Section 2.4, "Editing Additional Files", User's Guide Section 3.5, "Editing Multiple Files", SET FILEID

Examples

Assume you are currently editing two files, A:\ABC.DEF and A:\XYZ.XYZ, and that A:\ABC.DEF is the current file.

```
KEDIT
```

This would make A:\XYZ.XYZ, the next file in the ring, become the current file.

```
KEDIT
```

This would cause KEDIT to wrap around to the beginning of the ring again, making A:\ABC.DEF become the current file.

KEDIT C:\HELLO.TXT

Since C:\HELLO.TXT is not currently in the ring, KEDIT will read it in from disk, adding it to the ring and making it the current file. If C:\HELLO.TXT does not exist, KEDIT adds an empty file with that fileid to the ring.

KEDIT A:\ABC.DEF

This makes A:\ABC.DEF, which is already in the ring, become the current file.

KEDIT D:\FITNESS\WALK.TXT D:\FITNESS\RUN.TXT

KEDIT adds the files D:\FITNESS\WALK.TXT and D:\FITNESS\RUN.TXT to the ring.

KEDIT "D:\FITNESS\Go for a swim.TXT"

KEDIT adds the file "D:\FITNESS\Go for a swim.TXT" to the ring. Note that fileids that contain blanks must be enclosed in double quotes.

KEDIT D:\FITNESS*.TXT

KEDIT adds to the ring all files in the D:\FITNESS directory that have a file extension of TXT.

KEDIT \\SERVER\DISKC\BICYCLE.TXT

KEDIT adds the file \\SERVER\DISKC\BICYCLE.TXT to the ring. This example uses a UNC (Universal Naming Convention) name to specify a file located on a network server.

KHELP

Format **KHELP [*topic*]**

Description Use the KHELP command to display the KEDIT for Windows Help file. The KHELP command does the same thing as the HELP command; it is included in KEDIT for Windows mainly for compatibility with earlier versions of KEDIT. See the description of the HELP command for further details.

See also HELP

LEFT

Format **LEft [*n*|HALF]**

Description

The LEFT command scrolls your view of the file *n* columns to the left. The LEFT command does not affect the contents of your file; it only affects which columns of your file are displayed in the document window.

LEFT with no operands scrolls one column to the left. LEFT HALF scrolls half the width of the document window to the left.

For example, assume that you have issued the command

```
SET VERIFY 40 *
```

so that columns 40 through 119 of your file are visible in a window 80 columns wide. Using the LEFT and RIGHT commands, you can scroll left or right.

```
LEFT 10
```

would scroll 10 columns to the left, showing you columns 30 through 109 of your file.

```
LEFT 20
```

would then scroll an additional 20 columns to the left, showing you columns 10 through 89 of your file. (If you go far enough to the left, it is possible to have ''imaginary'' columns to the left of column 1 displayed. This is generally not useful, and is allowed only because it lets KEDIT handle certain situations involving multiple pairs of VERIFY columns consistently.)

The LEFT and RIGHT commands affect the value of VERSHIFT. VERSHIFT is a number that represents the number of columns that your view has been scrolled to the left or right of the columns specified by the VERIFY setting. For example, after the commands LEFT 10 and LEFT 20, VERSHIFT will be -30, since you have scrolled a total of 30 columns to the left. If you then issued a RIGHT 65 command, VERSHIFT would be set to 35 (that is, -30 + 65). (You can QUERY VERSHIFT, but you cannot directly set it. Its value is controlled by the LEFT and RIGHT commands, and by the RGTLEFT command. KEDIT's AUTOSCROLL facility also adjusts the value of VERSHIFT.) The commands LEFT 0 and RIGHT 0 are special cases. They reset the value of VERSHIFT to 0. It is also reset to 0 whenever you issue a SET VERIFY command.

You can also scroll left and right in your file by using the mouse to manipulate the horizontal scroll bar.

See also RGTLEFT, RIGHT, SET VERIFY

LEFTADJUST

Format **LEFTAdjust [*target*]**

Description Text in the *target* area is left-adjusted, with the leftmost nonblank character of each line moved to the left margin column.

You can use LEFTADJUST BLOCK, or the Leftadjust Block button on the bottom toolbar, to left-adjust line blocks, box blocks, and one-line stream blocks. Box blocks and one-line stream blocks are given special handling: KEDIT left-adjusts the text within the block boundaries, and text outside the block is not affected. You cannot left-adjust a multi-line stream block.

See also CENTER, RIGHTADJUST, SET MARGINS

Examples

```
LEFTADJUST
```

The focus line is left-adjusted. This is the default assignment for Ctrl+L.

```
LEFTA 12
```

KEDIT left-adjusts the text in the focus line and the eleven lines following it, for a total of twelve lines.

LESS

Format

```
LESS target
LESS TAG target
```

Description After you use the ALL command or the Edit Selective Editing dialog box to select a subset of the lines in your file for display, you may decide that you want to work with a smaller subset of your file. The first form of the LESS command, LESS *target*, lets you do this, removing from display all lines matching the specified target.

Similarly, after you use the TAG command to highlight a subset of the lines in your file, you can use the second form of the LESS command, LESS TAG *target*, to turn off highlighting for all lines that match the specified target.

A companion command to the LESS command is the MORE command, which lets you select or highlight more lines rather than fewer lines.

The LESS command works by manipulating the selection levels or tag bits of the lines in your file. Here are some of the details, which can be ignored by most users of the LESS command.

LESS *target* checks whether either of DISPLAY 0 1 or DISPLAY 1 1 is in effect. If so, it assumes that you have previously issued the ALL command to set the selection level of some lines to 1, and it resets the selection level of all lines matching the specified target to 0. Otherwise, LESS puts DISPLAY 1 1 into effect, sets the selection level of all lines that match the target to 0, and sets the selection level of all other lines to 1.

LESS TAG *target* checks whether HIGHLIGHT TAGGED is in effect. If so, it assumes that you have already used the TAG command to highlight some of the lines in your file, and turns the tag bit off for all lines that match the specified target. If HIGHLIGHT TAGGED is not in effect, LESS puts it into effect, turns off the tag bit for all lines that match the target, and turns it on for all other lines.

See also

User's Guide Chapter 8, "Selective Line Editing and Highlighting", ALL, TAG, MORE

Examples

```
ALL /ABC/
LESS /DEF/
```

In this example, you first issue the ALL command to display only the lines in your file that contain the string ''ABC''. If it then turns out that many of the lines with ''ABC'' also contain ''DEF'', and that these lines are not really of interest to you, you can use the LESS command in the example to remove them from display.

```
TAG /ABC/
LESS TAG /DEF/
```

In this example, you first issue the TAG command to highlight the lines in your file containing ''ABC''. If many of the highlighted lines also contain ''DEF'', and these lines are not of interest to you, you can use the LESS command to turn off their highlighting.

LOCATE

Format

[Locate] ***target***

Description

Use the LOCATE command to make the specified target line become the new focus line. Targets are fully described in User's Guide Chapter 6, "Targets", and are summarized below. You can also search for text by using the Edit Find dialog box.

The word ''LOCATE'' is optional in most cases, and you can normally specify targets without actually entering ''LOCATE''. So the following pairs of commands are equivalent:

```
/Fred/
LOCATE /Fred/
```

Both of these look for the next line containing "Fred".

```
:20
L :20
```

Both of these move to line 20 of your file.

```
10
LOCATE 10
```

Both of these move down 10 lines in your file.

An exception comes with targets that begin with an alphabetic character. For example, to locate the next altered line, you can use ‘‘LOCATE ALTERED’’ or can abbreviate it to ‘‘L ALTERED’’, but cannot use simply ‘‘ALTERED’’.

When using the LOCATE command (with or without actually using the word LOCATE), you can immediately follow the target specification with another KEDIT command that is to be carried out after the target line has been made the focus line. For example,

```
LOCATE :4 DELETE 3
```

This tells KEDIT to make line 4 of your file the focus line and then delete three lines (lines 4 through 6). A more complicated example would be

```
-* /abc/ -2 add
```

This tells KEDIT to make the top-of-file line the focus line, then search from there for the string ‘‘abc’’, then go up two lines and then add a blank line to the file.

Operation of the LOCATE command is affected by the settings of ZONE, CASE, ARBCHAR, VARBLANK, HEX, WRAP, and STAY. If THIGHLIGHT ON is in effect, as it is by default, KEDIT will highlight strings found by LOCATE on your display.

If you issue the LOCATE command with no operands, KEDIT re-executes the last LOCATE command that you issued from the command line. By default, Shift+F1 is set to issue a LOCATE command with no operands. This allows you to, for example, locate the line containing some string, make some changes to the line, and then press Shift+F1 to locate the next occurrence of the string.

See also

User’s Guide Chapter 6, “Targets”, CLOCATE, FIND, TFIND, SET THIGHLIGHT

Examples

```
/abc/
LOCATE /abc/
```

These two commands are equivalent. Both tell KEDIT to move the focus line pointer to the next line containing ‘‘abc’’.

```
:4
LOCATE :4
```

These two commands are equivalent. Both tell KEDIT to move the focus line pointer to line 4 of the file.

```
LOCATE ,abc/def,
,abc/def,
```

These two are also equivalent. Both tell KEDIT to make the next line in the file containing the string ''abc/def'' become the focus line. Since the string target contains a slash (''/''), which is the usual string delimiter, you have to use some other special character as the delimiter (in this case, a comma).

```
LOCATE WORD /Fred/
```

Locates the next line containing the word ''Fred''. The word ''LOCATE'' is not optional since the target specification begins with an alphabetic character.

Target summary

Here is a summary of the types of targets available for use with LOCATE and with other KEDIT commands that take targets as operands. See User's Guide Chapter 6, "Targets", for more discussion of KEDIT targets.

Absolute line number targets use a number preceded by a colon (':') and give the line number of the target line.

```
 :10     :30     :999
```

Relative line number targets use a number, optionally preceded by a minus sign ('-'), and specify the number of lines below or above the focus line of the target line. ''*'' and ''-*'' can be used to refer to the end-of-file line and the top-of-file line.

```
 18      -72     *
```

Named line targets use a line name preceded by a period ('.') to refer to lines given a name via the SET POINT command or the Actions Bookmark dialog.

```
.a     .xyz2
```

String targets refer to a line by specifying some portion of the text contained in the line. They can optionally be preceded by a minus sign ('-') to indicate that KEDIT is to look backward in the file for the specified string.

```
/this week/        -/tomorrow/
```

String targets can be preceded by a tilde ('~') to indicate a negative string search—a search for a line that does *not* contain the specified string.

```
~/yesterday/       -~/1776/
```

You can use logical operators to combine string targets. Use ampersands ('&') to look for a line that contains each of two or more strings, and use vertical bars ('|') to look for a line that contains any of two or more strings.

```
/now/ & /never/         /easy/ | /hard/
```

You can precede a string target with WORD (which can be abbreviated to ''W'') to indicate that the string must begin and end on a word boundary. PREFIX means that the string must start on a word boundary and SUFFIX means that it must end on a word boundary; these can be abbreviated to ''P'' and ''S''.

```
word /the/          prefix /un/
```

String target searches can be affected by the settings of ZONE, CASE, ARBCHAR, VARBLANK, HEX, WRAP, and STAY.

You can precede a string target with REGEXP (which can be abbreviated to "R") to indicate that the string target involved is specified using a regular expression. Regular expressions are summarized below.

```
regexp /[a-z]+./
```

Line class targets refer to lines according to whether they are blank (BLAnk), according to their selection level (SELect *n* [*m*]), or according to their flag bits (NEW, CHAnged, TAGged, ALTered).

```
blank       select 3          altered
```

Line class targets can be used with negation and with logical operators, and can be combined with string targets.

```
~blank        altered | /title/
```

Group targets refer to a group of lines, either ALL lines in the file, the currently marked BLOCK, or all lines in the focus PARAgraph.

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

LOCK

Format **LOCK**

Description Use the LOCK command to prevent other users on your network, or other processes on your own computer, from accessing the disk copy of a file that you are currently editing.

The LOCK command causes KEDIT to lock the current file. That is, KEDIT opens the disk copy of the current file and keeps it open until the file is removed from the ring or until the UNLOCK command is issued for the file.

Attempting to LOCK a file that is already locked causes an error message. If the current file is locked, KEDIT displays ''Locked'' on the status line. Additionally, if the current file is locked and IDLINE ON is in effect, its fileid will be preceded on the idline with an asterisk.

See also User's Guide Chapter 12, "File Processing", LOCK and NOLOCK initialization options, UNLOCK, SET LOCKING, SET SHARING

LOWERCASE

Format **LOWercase [*target*]**

Description Use the LOWERCASE command to convert uppercase characters in a specified portion of your file to lowercase characters.

If you give no operands, the LOWERCASE command converts the focus line to lowercase. Otherwise, all text within the specified target area that falls within the current ZONE columns is converted. If the target area is a box block, its entire contents are converted, regardless of the ZONE settings. If STAY OFF is in effect, the last line lowercased becomes the focus line. Otherwise, the focus line location does not change.

With the default of INTERNATIONAL NOCASE in effect, LOWERCASE treats only the 26 letters from ''A'' to ''Z'' and from ''a'' to ''z'' as alphabetic. With INTERNATIONAL CASE in effect the characters to be treated as alphabetic, and what their uppercase equivalents are, are determined by your Windows language drivers.

Note that you can use Shift+F6, the Actions Lowercase menu item, or the Lowercase Block button on the bottom toolbar to lowercase a block of text.

See also UPPERCASE, SET INTERNATIONAL

Examples

```
LOWERCASE
```

All uppercase characters in the focus line are converted to lowercase.

LOWERCASE 6

All uppercase characters in the focus line and in the five lines following it, for a total of six lines, are converted to lowercase.

LPREFIX

Format

LPrefix [*text*]

Description

The LPREFIX command places the *text* that you specify into the prefix area of the focus line and then processes all pending prefix commands.

The LPREFIX command is equivalent to your moving the cursor to the prefix area of the focus line, typing the *text* into the prefix area, and executing the SOS DOPREFIX command (which is part of the definition of the F12 key under INTERFACE CUA and of the Home key under INTERFACE CLASSIC). LPREFIX is useful from within macros that need to execute prefix commands. It also provides a way to enter and execute prefix commands when the prefix area is turned off.

See also

User's Guide Chapter 9, "Tailoring KEDIT", SET PREFIX

Examples

LPREFIX X

KEDIT places "X" into the prefix area of the focus line, and then executes this command, along with any other pending prefix commands. The X prefix command uses KEDIT's selective line editing facilities to exclude the focus line from the display.

LPREFIX DD

KEDIT places "DD" into the prefix area of the focus line, and then executes all pending prefix commands. If a matching DD prefix command is already pending, the specified block of lines will be deleted. Otherwise, the newly-entered DD command will remain pending.

MACRO

Format

MACRO *macroname* [*text*]

Description

The MACRO command causes KEDIT to run the KEXX language macro specified by *macroname*. Any *text* after the *macroname* is passed to the macro as an argument.

If IMPMACRO ON is in effect, as it is by default, you do not usually need to issue the MACRO command. Whenever KEDIT sees a command that it does not recognize it will automatically look for a macro by that name. This ''implied macro'' facility works only for macros whose names consist entirely of alphabetic characters.

KEDIT keeps in memory the macros involved in the default key and mousebar assignments, along with any macros that you have defined or loaded via the DEFINE command. If *macroname* specifies an in-memory macro, KEDIT can immediately run the macro.

If the macro is not in memory, KEDIT reads the macro in from a disk file and runs it, removing it from memory after it has completed execution. If *macroname* does not include a file extension, KEDIT assumes that the macro is in a file with an extension of .KEX. If *macroname* contains a DOS drive or path specification, KEDIT reads the macro from the specified drive or directory. Otherwise, KEDIT looks for the macro in your current directory. If this fails, KEDIT looks in any directories that you have specified via SET MACROPATH. Finally, 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. If the macro still cannot be located, an error message is issued and the MACRO command fails.

As discussed in User's Guide Section 10.2.3, "Storing Your Macros", we normally recommend that macros that you create be kept in the "KEDIT Macros" subdirectory of your Windows Documents folder (which is sometimes known as the My Documents folder).

See also

User's Guide Chapter 10, "Using Macros", MACROPATH initialization option, DEBUG, DEFINE, IMMEDIATE, SET IMPMACRO, SET MACROPATH

Examples

```
MACRO REVERT Hello
```

KEDIT tries to run an in-memory macro named REVERT. If none is found, KEDIT looks on disk for REVERT.KEX. When the macro is run, ''Hello'' is passed to it as an argument, which can be accessed via the ARG() function or PARSE ARG instruction.

```
MACRO F3
```

KEDIT tries to run an in-memory macro named F3. Since F3 is the name of a key, F3 will be found in memory. Issuing the command MACRO F3 is equivalent to pressing the F3 key.

```
MACRO D:\TEST
```

KEDIT runs the macro stored on disk in the file D:\TEST.KEX.

MACROS

Format

MACROS [ALL|CHANGED]
MACROS *macroname1* [*macroname2* ...]

Description

The MACROS command places the definitions of one or more of your in-memory macros into a file named MACROS.KML, so that you can view the definitions, modify them, and possibly save them to disk for future use.

Macro definitions are placed in MACROS.KML in the KEDIT Macro Library format normally used with .KML files. If MACROS.KML is already in the ring of files you are editing, the MACROS command replaces its contents. Otherwise, MACROS.KML is added to the ring. The file is created only in memory; use the FILE or SAVE commands if you want to save it to disk.

MACROS ALL
MACROS

MACROS ALL (or MACROS with no operands) places the definitions of all in-memory macros into MACROS.KML. (Default macro assignments for certain keys with ''uninteresting'' definitions, like the 3 key if it simply enters a ''3'', are not included.)

MACROS CHANGED

Places the definitions of all macros that you have defined via the DEFINE command into MACROS.KML, but does not include KEDIT's built-in default macro definitions.

MACROS *macroname1* [*macroname2* ...]

Places the definitions of the specified in-memory macros into MACROS.KML.

Since the MACROS command can cause a file (MACROS.KML) to be added to the ring, the PROFILE, NOPROFILE, PROFDEBUG, NOREG, NOINI, and NOMSG options, as discussed in Chapter 2, "Invoking KEDIT", can also be used with the MACROS command.

You can use QUERY MACRO *macroname* (or DEFINE *macroname* with no additional operands) to display the definition of a single macro in the message area. You can use MODIFY MACRO *macroname* to place the definition of a single macro, which must have a one line definition, on the command line for easy modification.

See also

DEFINE

Examples

```
MACROS
```

Places the definitions of all in-memory macros into MACROS.KML.

MACROS F1 F2 F3 F4

Places the definitions of F1, F2, F3, and F4 into MACROS.KML.

MARK

Format

```
MARK Line|Box|Stream [PERSISTent|SELection]
     [Anchor|Word] [RESET]
MARK CMDline [SELection] Anchor|Word [RESET]
MARK Line [PERSISTent|SELection] [Anchor] ALL
MARK REANCHOR line [col]
MARK PERSISTent
```

Description

The MARK command, used primarily within macros, marks the boundaries of a line, box, or stream block. KEDIT has a number of commands that can operate on the resulting marked block, allowing you to uppercase it, delete it, copy it to the clipboard, etc.

MARK Line|Box|Stream

Marks the focus line as one edge of a line block, or marks the focus column as one corner of a box or stream block.

You can only have one block marked at a time. If a block is marked in some file other than the current file when you issue the MARK command, that block is reset.

By default, Alt+L issues the command MARK LINE, Alt+B issues the command MARK BOX, and Alt+Z issues the command MARK STREAM. Since no other operands are specified, these commands mark persistent, unanchored line, box, or stream blocks.

PERSISTent|SELection

There are two types of blocks: persistent blocks and non-persistent blocks. Persistent blocks remain marked until you explicitly unmark them. Non-persistent blocks, which are usually referred to as ''selections'', are unmarked as soon as the cursor is repositioned. Use the PERSISTENT|SELECTION operand of the MARK command to specify whether a persistent block or a selection is to be marked; PERSISTENT is the default. Selections are available only when INTERFACE CUA is in effect, and the SELECTION operand is invalid if INTERFACE CLASSIC is in effect.

Anchor

The ANCHOR operand determines how an existing block changes size when a new boundary is marked. For an unanchored block, the block will extend from the focus line or column to the most distant existing boundary of the block. For an anchored block, the block extends from the focus line or column to a fixed ''anchor'' position. Persistent blocks are by default unanchored but you can use the ANCHOR operand to mark persistent anchored blocks. Selections are always anchored, so the ANCHOR option can be used for selections but is not necessary.

You can use the ANCHOR operand when you begin to mark a new block to indicate that the initial boundary of the block is to be the block's anchor.

You can then use the ANCHOR operand when you change the size of the block to indicate that the block should extend from the focus line or column to the anchor. If you use the ANCHOR operand when extending an existing block that does not have an anchor, the beginning of the block becomes the anchor.

By default, blocks marked with the mouse or (if INTERFACE CUA is in effect) with Shift+cursor-pad keys are anchored blocks while blocks marked with Alt+L, Alt+B, and Alt+Z are unanchored blocks.

Word
The WORD operand is valid only with stream blocks and causes the MARK command to mark or extend blocks on a per-word basis. If the focus column contains a nonblank character, the focus word and any trailing blanks are marked. If the focus column contains a blank, the inter-word spaces are marked. Using the WORD operand also causes the behavior associated with the ANCHOR operand to occur.

RESET
Causes any existing block to be reset, as if you had issued the RESET BLOCK command, so that the MARK command will mark a new block.

Four additional forms of the MARK command are available:

MARK CMDline [SELection] Anchor|Word [RESET]
MARK CMDLINE, available only when INTERFACE CUA is in effect, marks a command line selection. The SELECTION operand is optional but has no effect, since persistent command line blocks are not available. Command line selections are always anchored, and either the ANCHOR or WORD operand is required. If no command line selection currently exists, or if the RESET operand is used, the cursor position on the command line is marked as the start of a command line selection. Otherwise, the existing command line selection is extended to the cursor position.

MARK Line [PERSISTent|SELection] [Anchor] ALL
This form of the MARK command marks the entire file as a persistent line block (the default) or as a line selection. If you use it to mark the file as a line selection when the cursor is not already in the file area, the cursor is moved to the file area.

MARK REANCHOR *line* [*col*]
MARK REANCHOR extends an existing selection or persistent block to the line and (for box and stream blocks) column of the file that you specify, and then makes the block an anchored block, with the line and column to which the block was extended as the anchor.

MARK PERSISTent
MARK PERSISTENT changes a selection into a persistent block; it is used by the macro that handles the Edit Make Persistent menu item.

Notes

The MARK command is rarely issued from the command line, but is instead most often issued from within macros. In fact, most of the MARK command's operands exist only to help KEDIT's default mouse and keyboard macros work smoothly. If

you are making changes to KEDIT's default behavior in this area, you may need to use some of these operands. In more ''run-of-the-mill'' macros that need to mark and then operate on some portion of a file, it is usually easiest to stick to MARK LINE, MARK BOX, and MARK STREAM, issued with no other operands so that they mark persistent, unanchored blocks,

KEDIT will normally not let you have a zero-length selection, and considers a zero-length selection as equivalent to having no selection at all. A command like MARK STREAM SELECTION RESET would appear to create a selection extending from the cursor position to the cursor position, which would therefore be zero bytes in size, but KEDIT internally unmarks such a selection as soon as it is created. That is, after MARK STREAM SELECTION RESET, EXTRACT /BLOCK/ will indicate that there is no block marked, and not that there is a zero-length selection. An exception to this comes during the processing of mouse clicks (that is, during the processing of BUTTON1DOWN, etc.), where it is useful to create a zero-length selection that can then be extended by dragging the mouse. Zero-length selections are, therefore, allowed during the processing of mouse activity, but if a zero-length selection exists after the mouse button is released, it is automatically reset.

See also EXTEND, RESET, QUERY BLOCK, SET DRAG

Examples

```
MARK BOX
```

Mark the focus column as one corner of a box block. If there is no existing block in the current file, you will have a one-character box block. If there is an existing block, the block will extend from the focus column to the farthest existing corner of the block.

```
MARK LINE ANCHOR RESET
```

Reset any existing block, mark the focus line as a one-line line block, and make that the anchor for changes to the block.

MERGE

Format **MErge *target1 target2* [*col*]**

Description The MERGE command takes one group of lines, from the current line up to (but not including) the line specified by *target1*, and merges it with another group of lines, beginning at the line specified by *target2*.

Each line in the first group is merged into the corresponding line in the second group, with column 1 of the line from the first group positioned in the column specified by the *col* operand, which defaults to column 1. Nonblank text from the first line replaces the corresponding text from the second line; where there is a blank column in the first line, the corresponding position in the second line is left unchanged. So, for example, merging the line

```
A␣A␣
```

with the line

```
␣␣BB
```

would produce

```
A␣AB
```

After the merge operation is complete, the first group of lines (whose contents have been merged with the second group) is deleted.

Examples

Assume that a file contains the following six lines, and that line 1 is the focus line:

```
Ann
Pat       cycling
          h ppy
Fred and Jim went swimming
Jane and Bob went jogging
Edna and Sam were sad
```

The command ‘‘MERGE 3 :4 10’’ would result in:

```
Fred and Ann went swimming
Jane and Pat went cycling
Edna and Sam were happy
```

with the first group of lines, lines 1 through 3, merged with the second group of lines, starting in column 10 of line 4, and deleted from their original position in the file.

MODIFY

Format

MODify *option*
MODify MACRO *macroname*

Description

The MODIFY command displays the current value of one of the options that you can SET. The value is displayed on the command line in the form of a SET command for the option, allowing you to change the value by typing over it and then re-entering the modified value of the SET option. The MODIFY command takes one operand, the name of the option whose value you want to change. (The name of the option can be abbreviated using the same minimal truncations that the corresponding SET command allows.)

For compatibility with earlier versions of KEDIT, you can also use the MODIFY command with options that you can QUERY but cannot SET. For example, KEDIT allows you to use MODIFY TIME, even though the SET TIME command that is placed on the command line as a result is invalid.

You can use a second form of the command, MODIFY MACRO *macroname*, to get the definition of an in-memory macro, whose definition can only be one line long, in a DEFINE command on the command line for easy modification.

See also STATUS, Chapter 4, "The SET Command", Chapter 5, "QUERY and EXTRACT"

Examples

```
MODIFY BOUNDMARK
```

KEDIT will display current BOUNDMARK settings on the command line. You can then overtype or otherwise edit these settings and enter the modified values.

```
MODIFY DIRECTORY
```

KEDIT displays the current directory on the command line. As an exception to the usual behavior for the MODIFY command the current directory is preceded on the command line by "CHDIR" and not "SET DIRECTORY", because the CHDIR command is KEDIT's closest equivalent to the non-existent SET DIRECTORY command.

```
MODIFY MACRO XYZ
```

KEDIT will display the current definition of XYZ, which must be a one-line in-memory macro, on the command line. You can then overtype or otherwise edit the definition.

MORE

Format

MORE *target*
MORE TAG *target*

Description

After you use the ALL command or the Edit Selective Editing dialog box to select a subset of the lines in your file for display, you may decide that you want to work with a larger subset of your file. The first form of the MORE command, MORE *target*, lets you do this, adding all lines that match the specified target to the set of selected lines.

Similarly, after you use the TAG command to highlight a subset of the lines in your file, you can use the second form of the MORE command, MORE TAG *target*, to turn on highlighting for all lines that match the specified target.

A companion command to the MORE command is the LESS command, which lets you select or highlight fewer lines rather than more lines.

The MORE command works by manipulating the selection levels or tag bits of the lines in your file. Here are some of the details, which can be ignored by most users.

MORE *target* checks whether either of DISPLAY 0 1 or DISPLAY 1 1 is in effect. If so, MORE assumes that you have previously used the ALL command to select a subset of the lines in your file, and MORE sets the selection levels of all lines that match the

specified target to 1. Otherwise, issuing MORE *target* is equivalent to issuing ALL *target*.

MORE TAG *target* checks whether HIGHLIGHT TAGGED is in effect. If so, it assumes that you have already used the TAG command to highlight some of the lines in your file, and turns the tag bit on for all lines that match the specified target. If HIGHLIGHT TAGGED is not in effect, issuing MORE TAG *target* is equivalent to issuing TAG *target*.

See also User's Guide Chapter 8, "Selective Line Editing and Highlighting", ALL, TAG, LESS

Examples

```
ALL /ABC/
MORE /DEF/
```

In this example, you first issue the ALL command to display only the lines in your file that contain the string ''ABC''. If it then turns out that you also want to see any lines that contain ''DEF'', you can use the MORE command in the example to add them to the display.

```
TAG /ABC/
MORE TAG /DEF/
```

In this example, you first issue the TAG command to highlight the lines in your file containing ''ABC''. If you then decide that you would like to highlight lines with ''DEF'' as well, you can use the MORE command in the example to highlight them.

MOVE

Format

MOve *target1 target2*
MOve BLOCK

Description Use the MOVE command to move text from one location to another.

There are two forms of the command:

MOve *target1 target2*
All text in the target area specified by *target1* is moved, with the moved text placed immediately after the line specified by *target2*. With this form of the command, the target area specified by *target1* cannot be a box or stream block. Text can only be moved within the current file, and not from one file in the ring to another.

MOve BLOCK
This form of the MOVE command moves all text in the currently marked block. The block can be in the current file or in another file, allowing you to move text from one file to another. If the block is a line block, text is moved after the focus line. If the block is a box or stream block, text is moved to the left of the text at the focus column position. The block remains marked in its new position.

MOVE BLOCK is assigned by default to Alt+M, which uses the RESET command to unmark the block after the move is complete.

You can also move blocks by using the Move button on the bottom toolbar, by using mouse button 1 to drag and drop the block, or by using the Edit menu to cut the block to the clipboard and then paste it back at a new location.

See also COPY

Examples In the following examples, assume that line 6 is the focus line.

```
MOVE :12 /ABC/
```

Lines 6 through 11 are moved following the next line containing "ABC".

```
MOVE 4 :88
```

Lines 6 through 9 (a total of four lines) of the current file are moved following line 88 of the current file.

```
MOVE BLOCK -*
```

All lines in the currently marked block, which must be a line block in the current file for this form of the command, are moved following the top-of-file line.

```
MOVE BLOCK
```

If the currently marked block is a line block, KEDIT moves its contents following line 6 of the file. If the block is a box or stream block, KEDIT moves the block to the left of the text at the focus column position.

MSG

Format **MSG [*text*]**

Description The MSG ("message") command, displays the specified *text* on the message line.

More commonly used in macros than the MSG command is the SAY instruction, which also displays text on the KEDIT message line.

See also CMSG, DIALOG, DMSG, EMSG, WMSG, SET MSGLINE, KEXX SAY instruction

Examples

```
MSG Function complete
```

KEDIT displays "Function complete" on the message line.

NEXT

Format **Next [*n*]**

Description The line *n* lines below the focus line becomes the new focus line. If *n* is not specified, the line that is one line below the focus line becomes the focus line.

The DOWN command and the LOCATE command with a relative line number target perform the same function as the NEXT command.

See also UP

Examples

```
NEXT
```

The line one line below the focus line becomes the new focus line.

```
NEXT 8
```

The line eight lines below the focus line becomes the new focus line.

NFIND, NFINDUP, NFUP

Format

NFind *text*
NFINDUp *text*
NFUp *text*

Description The NFIND command searches forward through your file for a line that does *not* start (in column 1) with the text that you specify. That line becomes the new focus line. The search starts with the line below the focus line and continues until the desired line is found or the end-of-file line is reached (wrapping to the top of the file if WRAP is ON).

NFINDUP (which can also be given as NFUP) does the same thing as the NFIND command, except that KEDIT searches upward for the desired line, beginning with the line above the focus line.

These commands are included in KEDIT mainly for compatibility with XEDIT. The TFIND command is a more general command that is preferable in most situations.

KEDIT remembers the last operand used whenever the FIND, FINDUP, NFIND, or NFINDUP commands are issued from the command line and, if you later issue one of these commands with no operands, this ''remembered operand'' is reused.

If CASE IGNORE is in effect, an alphabetic character in the search text will match either its uppercase or lowercase equivalent. Blanks in the search text act as wild card characters, matching any single character in your file. Underscore (''_'') characters in the search text match blanks in your file.

See also FIND, FINDUP, TFIND

Examples

```
NFIND ABC
```

The next line that does not begin with ''ABC'' becomes the focus line.

```
NFINDUP ABC
```

The first line above the focus line that does not begin with ''ABC'' becomes the new focus line.

```
NFIND ABC DEF
```

This command searches for a line that does not contain ''ABC'' in columns 1 through 3 and ''DEF'' in columns 5 through 7.

```
NFIND ABC DEF
```

This searches for a line that does not have ''ABC DEF'' in columns 1 through 7.

NOMSG

Format **NOMSG** ***command***

Description NOMSG, used mainly in macros, takes as its operand the text of a command that you would like KEDIT to execute with MSGMODE OFF in effect. KEDIT preserves the MSGMODE setting, executes the specified command but does not display any messages generated by the command, and then restores the setting of MSGMODE. (Messages generated by the command, while not displayed, do affect the value returned by QUERY LASTMSG.)

NOMSG is often useful within macros, since macros often need to issue commands that may generate irrelevant messages. For example, you may need to issue a CHANGE command from within a macro, but may not want KEDIT to display the message giving the number of occurrences changed. Without the NOMSG command, you would often need to save and restore the status of MSGMODE within macros, which can cause problems if a macro that turned MSGMODE OFF terminates unexpectedly before turning MSGMODE back ON.

See also NOMSG initialization option, SET MSGMODE

Examples

```
NOMSG LOCATE /1234/
```

KEDIT executes the command LOCATE /1234/, but does not display any error message if 1234 is not found. A macro that issues this command could then examine the return code to determine if the command was successful.

OEMTOANSI

Format **OEMTOANSI [*target*]**

Description Use the OEMTOANSI command to convert text in a specified portion of your file from the OEM character set to the ANSI character set.

All text within the specified target area that falls within the current ZONE columns is converted. If the target area is a box block, its entire contents are converted, regardless of the ZONE settings.

See also ANSITOOEM, User's Guide Section 3.7, "Character Sets"

Examples

```
OEMTOANSI ALL
```

Convert all lines of the file from OEM to ANSI.

OVERLAY

Format **Overlay *text***

Description The OVERLAY command overlays the text in the focus line, starting in column 1, with the text that you specify.

All text that you enter, starting with the character after the first blank following the OVERLAY command, is taken as ''overlay text'' that will overlay data in the focus line. Wherever there is a blank in the overlay text, the corresponding character in the focus line is not changed. Wherever there is an underscore character ("_") in the overlay text, the corresponding character in the focus line is replaced by a blank. All other characters in the overlay text replace the corresponding characters in the focus line.

There is no way to put underscore characters ("_") into the overlay text without having them converted to blanks in the focus line.

The COVERLAY command does the same thing as the OVERLAY command, except that OVERLAY overlays text starting in column 1, while the COVERLAY command overlays text starting at the focus column.

See also COVERLAY, REPLACE

OVERLAYBOX

Format **OVERLAYBox**

Description The OVERLAYBOX command, normally assigned to Shift+Ctrl+O, overlays a portion of your file with the text in a block. OVERLAYBOX operates on line blocks, box blocks, and one-line stream blocks; it does not operate on multi-line stream blocks. The name OVERLAYBOX exists for historical reasons, since in earlier versions OVERLAYBOX could only operate on box blocks.

If OVERLAYBOX is issued when a box block or one-line stream block is marked, a copy of the text in the block overlays the lines of your file beginning at the focus line.

POPUP

Format **POPUP [MOUSE|TEXT|CENTER] /*item1*/[*item2*/ ...]**

Description Use the POPUP command from within a macro, normally a macro associated with a mouse action, to display a pop-up menu to the user of the macro. The user of the macro then uses the mouse or keyboard to make a choice from the menu, and this choice is returned to your macro.

The first, optional, operand determines where the pop-up menu appears. If MOUSE is specified, or if the operand is omitted, the upper left corner of the menu is positioned at the mouse pointer location. With TEXT, the upper left corner is positioned at the text cursor location, and with CENTER the upper left corner is positioned at the center of KEDIT's frame window.

The POPUP command takes as its remaining operands a delimited list of the menu items involved. The delimiters are normally slash characters ("/"), but can be any special character not contained in any of the menu items. Some special rules:

If a menu item begins with a tilde ("~"), that menu item will be grayed out and cannot be selected by the user.

If a menu item consists solely of a minus sign ("-"), that menu item is displayed as a separator line within the menu.

If a menu item contains an ampersand ("&"), the ampersand is not displayed but the character following the ampersand is underlined and the user of the macro can type that character to choose that menu item. To have an ampersand displayed as part of a menu item, use two consecutive ampersands. POPUP returns its results through macro variables, much as the EXTRACT command does.

On completion of the POPUP command, two macro variables are set:

popup.0 1

popup.1 The contents of the menu item selected by the user. This value is returned exactly as you specified it, and includes any ampersands present in your original string. If no item was selected (because the user pressed the Escape key or clicked outside of the menu), this variable is set to the null string.

See also DIALOG, READV

Examples

```
'popup /Add/Delete/~Replace/-/Upload/Download/'
if popup.1 = '' then call no_selection
else if popup.1 = 'Add' then call adder
else if popup.1 = 'Delete' then call deleter
     ...
```

In this portion of a macro, a pop-up menu with the items Add, Delete, Replace (grayed-out and not selectable), a separator line, Upload, and Download is displayed. After the POPUP command completes, the variable POPUP.1 is set to the null string if no item was selected, and otherwise it is set equal to the text of the item that was chosen.

PRESERVE

Format **PREServe**

Description The PRESERVE command, used mainly in macros, causes KEDIT to save the current values of most SET options so that you can temporarily change them, execute some commands with the new settings, and then restore their values with the RESTORE command.

The settings of ARBCHAR, ARROW, AUTOINDENT, AUTOSAVE, AUTOSCROLL, BACKUP, BOUNDMARK, CASE, CMDLINE, COLMARK, COLOR, COLORING, CURLINE, CURRBOX, DISPLAY, ECOLOR, EOFIN, EOFOUT, EOLIN, EOLOUT, FORMAT, HEX, HIGHLIGHT, IDLINE, IMPMACRO, INPUTMODE, INTERNATIONAL, LINEND, LRECL, MARGINS, MSGLINE, MSGMODE, NEWLINES, NUMBER, PCOLOR, PREFIX, PREFIXWIDTH, RECFM, SCALE, SCOPE, SCROLLBAR, SHADOW, STAY, STREAM, SYNONYM, TABLINE, TABS, TABSAVE, TABSIN, TABSOUT, THIGHLIGHT, TIMECHECK, TOFEOF, TRAILING, TRANSLATEIN, TRANSLATEOUT, TRUNC, UNDOING, VARBLANK, VERIFY, VERSHIFT, WORD, WORDWRAP, WRAP, and ZONE are preserved.

See also RESTORE

PRINT

Format

```
PRint [target [n]]
PRint LINE [text]
PRint STRING text
PRint FORMfeed
PRint CLOSE
```

Description

Use the PRINT command to send data to your printer.

Print output normally is handled by Windows, and you can use the File Print Setup and File Print dialog boxes to determine which printer, printer driver, printer font, and printer margins to use. You can also use the SET PRINTER command to specify that printer output is to bypass Windows and go directly to a specific printer port.

KEDIT's focus is on text editing, and its printing abilities are very limited. KEDIT does not automatically position your paper in the printer, print headings, footings, or page numbers, handle boldface or italicized output, or use multiple fonts or proportional fonts.

If your printer uses device-specific escape sequences to control boldface, etc., you can imbed these sequences in your file or in macros and use the PRINT command to send them to your printer, but this requires that you use the SET PRINTER command to route output directly to a specific printer port (such as LPT1:, LPT2:, etc.). With the default of PRINTER WINDOWS printer output goes through Windows device-independent printer handling, and escape sequences are not properly processed.

There are several different forms of the PRINT command:

PRint [*target* [*n*]]

KEDIT sends all text in the *target* area to your printer. The *target* area can specify lines within your file, or can be any kind of marked block. PRINT with no operands prints the focus line. KEDIT follows each line sent to your printer with a carriage return-linefeed pair. The *n* operand of the PRINT command tells KEDIT to start a new page after every *n* lines have been printed, allowing you to print *n* lines per page. If *n* is not specified (or is 0), KEDIT will print continuously, and will not force any page ejects after a set number of lines.

When printing has completed, the last line printed becomes the new focus line if STAY OFF is in effect; if STAY ON (the default) is in effect, the focus line location is unchanged.

PRint LINE [*text*]

This sends the specified *text* to the printer, followed by a carriage return-linefeed. If HEX ON is in effect, you can specify the *text* using hexadecimal or decimal notation. If no *text* is specified, KEDIT sends only a carriage return-linefeed to the printer.

PRint STRING *text*

This sends the specified *text* to the printer but, unlike PRINT LINE, does not follow it with a carriage return-linefeed. If HEX ON is in effect, you can specify the *text* using hexadecimal or decimal notation.

PRint FORMfeed

Tells the printer to do a page eject, so that any additional output will begin on a new page.

PRint CLOSE

If you are using a print spooler, or your printer is accessed through a network, your system software may hold your printer output and not send it to the printer until you tell the system that you are ready by ''closing'' your printer. By default, SET PRINTER's CLOSE|NOCLOSE value is set to CLOSE, and KEDIT closes your printer after each use of the PRINT command. If SET PRINTER's CLOSE|NOCLOSE value is set to NOCLOSE, so that the printer is not closed automatically after each use of the PRINT command, you can use the PRINT CLOSE command at any time during a KEDIT session to tell KEDIT to close your printer. Regardless of SET PRINTER's CLOSE|NOCLOSE setting, KEDIT closes your printer automatically when you leave KEDIT, when you switch from KEDIT to another application, and when you use the SET PRINTER command to switch to a different printer.

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 and PRINTER WINDOWS and PRINTCOLING ON are in effect.

Notes

KEDIT will print syntax-colored text in color when you print to a color printer with PRINTER WINDOWS and PRINTCOLING ON in effect.

When you want to print either the contents of the currently-marked block or the contents of the entire file, you can use the Print File button on the default toolbar, or the File Print dialog box.

If you are using multiple PRINT commands to print data on a single page (for example, by issuing multiple PRINT LINE commands from within a macro), you will probably want to change the SET PRINTER's CLOSE|NOCLOSE setting from its default of CLOSE to NOCLOSE. This will prevent KEDIT from automatically closing the printer, which normally also involves a page eject, after each of your PRINT commands. You can then use PRINT CLOSE when you are ready to close the printer.

See also

SET PRINTER, User's Guide Section 3.10, "Printing"

Examples

```
PRINT BLOCK
```

All text in the currently marked block is printed.

```
PRINT ALL
```

The entire file is printed.

PRINT

KEDIT prints the focus line.

PRINT * 60

All lines from the focus line through the end of the file are printed, with a formfeed character sent to the printer every 60 lines.

PRINT LINE Hello there.

KEDIT sends "Hello there." to your printer, followed by a carriage return-linefeed.

PRINT CLOSE

KEDIT closes your printer device.

PURGE

Format

PURge *macroname1* [*macroname2* ...]

Description

Use the PURGE command to remove in-memory macros (put into memory via the DEFINE command) from memory. This allows you to free up the memory occupied by in-memory macros that you have finished using.

If you purge a macro assigned to a key that by default enters a character (such as the A or SHIFT+A macros), the key will revert to its default definition. If you purge a macro assigned to any other key, the key will then do nothing when you press it. Purging a macro assigned to these keys is equivalent to issuing the command

DEFINE *keyname* nop

If you purge any other in-memory macro, and then attempt to execute the macro, KEDIT will see that the macro is not in memory and will then search for it on disk.

If you attempt to purge a macro that is not in fact in memory, KEDIT takes no action.

Examples

PURGE F1 RETRY

KEDIT purges the definition of the F1 key; F1 will then be ignored if you press it. KEDIT also purges the definition of RETRY. Since this is not the name of a key, if you later try to run a macro called RETRY, KEDIT will look for RETRY.KEX on disk.

PUT, PUTD

Format

```
PUT [target [fileid]]
PUT LINE fileid [text]
PUTD [target [fileid]]
```

Description

The PUT command copies the contents of the specified *target* area of your file to the disk file specified by *fileid*. Text written to disk by the PUT command is not deleted from the file you are editing.

A second form of the PUT command lets you directly specify, as part of the PUT command, a line of *text* to be appended to the specified disk file.

The PUTD command does exactly the same thing as the first form of the PUT command except that, after text has been successfully written to disk, it is deleted from the file you are editing.

If you specify a *fileid* with no drive or directory, KEDIT uses the current drive and directory. The shortcuts for specifying a *fileid* that are discussed in connection with the SET FILEID command can also be used with the PUT and PUTD commands.

The text in the target area is appended to the specified disk file. If the file does not exist, it is created.

If no *fileid* is given, KEDIT creates a temporary file to hold the text. (If the temporary file already exists as a result of previous PUT or PUTD commands, it is completely replaced, and not appended to as would happen if you gave a fileid when issuing the PUT or PUTD command.) This temporary file can then be read back by issuing the GET command with no operands. If no target is given, the focus line is written to the temporary file.

PUT and PUTD do not create backup copies of the files you append to, regardless of the setting of BACKUP. The text written out will be affected by the settings of LRECL, RECFM, EOFOUT, EOLOUT, TRAILING, TABSOUT, and TRANSLATEOUT. To append to a file, KEDIT looks at the last byte of the file. If the last byte of the file is the DOS end-of-file character (character code 26), text is written starting at this last character of the file, overwriting the end-of-file character; otherwise, text is written after the last character of the file.

If STAY OFF is in effect, the line after the last line written to disk becomes the focus line when the PUT or PUTD command completes. If STAY ON is in effect, the focus line location does not change.

See also

GET

Examples

```
PUT 10 ABC.C
```

The focus line and the nine lines following it, for a total of ten lines, are appended to the file ABC.C. (If ABC.C does not yet exist, it is created.)

```
PUT 10
```

The focus line and the nine lines following it are placed into a temporary file that can later be retrieved by issuing the GET command with no operands.

```
PUT LINE ABC.C Hello there.
```

KEDIT adds the line ‘‘Hello there.’’ to end of the file ABC.C, creating the file if it did not already exist.

QUERY

Format

Query *option*

Description

The QUERY command allows you to determine the current value of any of the options that can be set via the SET command. You can also QUERY a number of values that cannot be directly set, such as the names of all files in the ring and the current time.

The QUERY command takes one operand, the name of the option whose value you want to see. The name of the option can be abbreviated using the same minimal truncations as the corresponding SET command allows. The value of the option is then displayed on the message line.

For a full discussion of all the options that you can QUERY, see Chapter 5, “QUERY and EXTRACT”.

See also

MODIFY, STATUS, Chapter 4, “The SET Command”, Chapter 5, “QUERY and EXTRACT”

Examples

```
QUERY TIME
```

The date and time are displayed in the message area.

```
QUERY MACRO F1
```

The definition of the macro assigned to the F1 key is displayed in the message area.

```
QUERY ZONE
```

The current ZONE settings are displayed in the message area.

QUIT, QQUIT

Format

```
QUIT
QQuit
```

Description

Use the QUIT command when you have finished working with the current file and have made no changes to it. KEDIT removes the file from memory and, if you are editing multiple files, makes the previous file in the ring become the current file. The file is not written to disk.

You will frequently use KEDIT to simply look at a file, with no intention of making changes to it. When you are finished with such a file, it is not necessary to write it back to your disk with the FILE command, since an unchanged copy of your file is already on disk. Instead, it makes sense in this situation to use the QUIT command to save time.

The QUIT command is assigned by default to function key F3. You can also use the File Close menu item to remove an unmodified file from the ring. (If you use File Close on a file that has been modified, KEDIT will ask you if you want to save the file before removing it from memory.)

If you have made changes to your file, you usually wouldn't want to use the QUIT command, since your changes would not be written to disk. So that accidental use of the QUIT command cannot cause you to lose valuable work, the QUIT command will work only if the file has not been changed since you began editing it or last used the SAVE command to save the file to disk. (KEDIT checks whether the count it maintains of alterations since the last SAVE is greater than 0.) If you try to QUIT from a file that has been changed, KEDIT will not QUIT but will instead give you an error message. If you really do want to QUIT from a file that has been changed, you can use the QQUIT command. QQUIT does the same thing as QUIT, except that it will exit from your file even if changes have been made.

See also

CANCEL, FILE, SAVE

READV

Format

```
READV Cmdline [initial]
READV EDITfield [initial]
READV KEY [Ignoremouse|NOIgnoremouse]
```

Description

The READV command, valid only if issued from a macro, causes KEDIT to read information from the user of the macro. The results are passed back through assignments to macro variables, in much the same way as the results of the EXTRACT command are passed back to a macro.

READV CMDLINE

READV CMDLINE lets your macro obtain a line of input from the user. READV CMDLINE allows optional specification of initial contents of the line to be read in. KEDIT reads a line of input from the command line, which is then passed back to your macro.

You can set the initial contents of the line to be read in by specifying the optional *initial* text.

READV CMDLINE sets these variables:

`readv.0` 1

`readv.1` Text read from the dialog box

READV EDITFIELD

READV EDITFIELD is like READV CMDLINE, except that input is read from an edit field that is displayed within a dialog box, and not from the command line.

READV KEY

READV KEY reads a single keystroke from the keyboard, waiting if necessary until a keystroke has been entered. Your macro can then determine the key that was pressed and act accordingly.

READV KEY has options that let you control the effect of mouse events that occur while waiting for a keystroke. With IGNOREMOUSE, the default, mouse events are ignored (except that, if MOUSEBEEP ON is in effect, the speaker beeps). With NOIGNOREMOUSE, mouse events cause READV KEY to terminate with a return code of 2, but they remain queued up for processing after the macro that issued READV KEY terminates.

READV KEY gives you the name of the key that is entered (as described in Chapter 7, "Built-in Macro Handling"), the character associated with the key (or a null string for function keys and other keys without assigned characters), the scan code (a decimal number from 0 to 255) of the key, and the Shift Status at the time KEDIT reads the key.

When a key is entered, READV KEY sets these variables:

`readv.0` 5 under Windows XP/2000/Vista; 4 under Windows 98/Me

`readv.1` Key name (in uppercase, with possible "C-", "S-", "A-", "S-C-", or "A-C-" prefix); "-" is always used in the prefix rather than "+" for compatibility with earlier versions of KEDIT.

`readv.2` Character value (or null)

`readv.3` Scan code (in decimal)

`readv.4` Shift Status (see below)

`readv.5` Extended Shift Status (see below; not set under Windows 98/Me)

Shift Status is an eight character string of 0's and 1's where characters are set to 1 as follows:

Position	Set to 1 if
1	INSERTMODE ON
2	CapsLock ON
3	NumLock ON
4	ScrollLock ON
5	Either Alt key down
6	Either Ctrl key down
7	Left Shift key down (XP/2000/Vista); either Shift key down (Windows 98/Me)
8	Right Shift key down (XP/2000/Vista); either Shift key down (Windows 98/Me)

For example, if INSERTMODE ON is in effect and a Ctrl key is down, READV.4 will be set to ''10000100''.

Extended Shift Status, not available under Windows 98/Me, is also an eight character string of 0's and 1's. Characters are set to 1 as follows:

Position	Set to 1 if
1	Always 0 (SysReq key status is not available)
2	CapsLock key down
3	NumLock key down
4	ScrollLock key down
5	Right Alt key down
6	Right Ctrl key down
7	Left Alt key down
8	Left Ctrl key down

Notes

Key combinations involving Alt, Ctrl, and a character key (that is, A-C-*x*) are ignored except when a macro for the key combination involved has been defined.

For keys normally processed by the ASCII macro (that is, special characters found on non-U.S. keyboards but not on U.S. keyboards, digits entered via the numeric pad, and characters entered via the Alt key-numeric pad method), READV.1 is set to ''ASCII *ddd*'', where *ddd* is the decimal value of the character code involved. The scan code in READV.3 is 0 for digits entered via the numeric pad, 56 (the scan code of the Alt key) for Alt key-numeric pad input, and the scan code of the key involved for special characters on non-U.S. keyboards.

See also

DIALOG, KEXX PULL and PARSE PULL instructions

RECOVER

Format **RECover [*n*|*]**

Description

The RECOVER command, normally assigned to Alt+R, allows you to recover lines of text that you have deleted or changed. KEDIT tries to save the last 100 deleted or changed lines in an internal buffer.

Note that the UNDO command provides a more powerful facility for undoing the effects of changes to your file. RECOVER is available mainly for compatibility with XEDIT and with earlier versions of KEDIT.

If you issue the RECOVER command and give a number as an operand, KEDIT tries to recover that many lines. The most recently deleted or changed line is inserted above the focus line, the next most recently deleted or changed line is inserted above that, etc. The last line recovered becomes the new focus line.

If you issue the RECOVER command with no operand, KEDIT tries to recover the last line deleted or changed. Issuing the RECOVER command again causes KEDIT to try to recover the line before that, etc. Issuing

```
RECOVER *
```

causes KEDIT to recover all the changed or deleted lines that it can.

See also UNDO

Examples

```
RECOVER
```

KEDIT tries to recover the last changed or deleted line of text.

```
RECOVER 4
```

KEDIT tries to recover the last four changed or deleted lines of text.

REDO

Format **REDO**

Description Use the REDO command, normally assigned to Ctrl+Bksp and to Ctrl+Y, if you have used the UNDO command to undo some changes to your file and then decide to redo those changes, reversing the effect of the UNDO command. If you have used the UNDO command repeatedly to undo multiple levels of changes, you can use the REDO command repeatedly to redo those changes. REDO will work only if no changes have been made to the current file since the corresponding UNDO command. After an UNDO command you can move around in the current file, can save it to disk, and can make changes to other files in the ring, and you can still use the REDO command. But REDO is not available once you make any further changes to the contents of the current file, change the selection levels of any line (for example, by using the ALL command), change the lineflags of any line (for example, by using the TAG command), add or delete any line names (for example, by using the SET POINT command). You can also REDO the last UNDO action using the Edit Redo menu item or using the Redo button on the toolbar.

Whenever it is possible to REDO previous UNDO commands, the available undo level count (the third number following ''Alt='' on the status line) will be followed by an asterisk (''*'').

See also User's Guide Chapter 3, "Using KEDIT for Windows", UNDO, SET UNDOING

REFRESH

Format **REFRESH**

Description Use the REFRESH command to cause KEDIT to update the display while a macro is active. Normally, KEDIT doesn't redisplay its windows after every command issued from within a macro. The windows are only updated when the macro finishes or when it needs keyboard input. The REFRESH command allows you to have the windows redisplayed at additional times during the execution of your macro. You might, for example, wish to refresh the windows to indicate to the user the progress of a long-running macro. Note, however, that frequent refreshes can slow your macro down significantly.

REGUTIL

Format

REGUTIL SAVE|CLEAR STATE|HISTory|SETTINGS
REGUTIL SAVE SET *option*
REGUTIL CONVERT SETTINGS
REGUTIL BACKUP CONFIG|HISTory
REGUTIL GET CONFIG|HISTORY *section name*

Description

Use the REGUTIL command to help manage the contents of the information saved by KEDIT in the Windows registry. Most KEDIT users will not need to use the REGUTIL command, since the registry is normally updated automatically by KEDIT at the appropriate times.

REGUTIL SAVE|CLEAR STATE

KEDIT normally saves certain information about the STATE of your KEDIT session, such as the position of your frame window, whether it is maximized, and the screen and printer fonts you are using, in the Windows registry at the end of each session, and refers to this information at the start of the next KEDIT session. With REGUTIL SAVE STATE, KEDIT immediately saves the current state of a session, even though the session has not ended. With REGUTIL CLEAR STATE, KEDIT removes this state information from the registry; if no state information is found in the registry at the start of the next KEDIT session, KEDIT will use internal defaults for your frame window position, fonts, etc.

REGUTIL SAVE|CLEAR HISTory

Similarly, KEDIT normally saves certain HISTORY information in the Windows registry at the end of each session. This includes information about commands you have recently issued from the command line, files you have recently edited, strings you have recently worked with in the Edit Find dialog box, etc. REGUTIL SAVE HISTORY will immediately update the history information in the registry. REGUTIL CLEAR HISTORY will clear out all of the KEDIT history information in the registry.

REGUTIL SAVE|CLEAR SETTINGS

KEDIT also saves the values of most SET options in the Windows registry. This is not done automatically at the end of a KEDIT session, but is instead done when you use Options Save Settings to save the value of all such options or when you use the Save Setting button in the Options SET Command dialog box to update the value of an individual setting in the registry. REGUTIL SAVE SETTINGS updates the value of all savable options in the registry, just as Options Save Settings does. REGUTIL CLEAR SETTINGS removes all saved settings from the registry; if no saved settings are present at the start of a KEDIT session, KEDIT will use default values for all of your SET options.

REGUTIL SAVE SET *option*

This command lets you update the value of an individual SET option in the Windows registry, and is equivalent to using the Save Setting button in the Options SET Command dialog box to save an individual setting.

REGUTIL CONVERT SETTINGS

This command adds a new file, REGSET.KEX, to the ring of files that you are editing. KEDIT takes each of the settings saved in the Windows registry, converts it into the equivalent KEDIT SET command, puts quotes around the SET command to ensure that the command is valid from within a KEDIT macro, and adds it to REGSET.KEX. REGSET.KEX is therefore a valid KEDIT macro containing SET commands corresponding to each of the settings saved in the registry.

REGUTIL BACKUP CONFIG|HISTory

REGUTIL BACKUP adds a file called KEDIT.CONFIG.REG or KEDIT.HISTORY.REG to the KEDIT ring, containing a copy (for CONFIG) of the KEDIT configuration settings saved in the registry or (for HISTORY) of recent command lines, files edited, etc. from the registry.

You can use these files to transfer your KEDIT configuration settings and/or history to a new machine. Save the .REG file to disk, copy it to your new machine, and double-click on it to install it in the registry of the new machine.

REGUTIL GET CONFIG|HISTORY *section name*

REGUTIL GET is valid only within macros. It reads a value from KEDIT's section of the registry, returning information in these macro variables::

regutil.0 0 if the requested information was not found in the registry or there was a problem reading it in, otherwise 1.

regutil.1 Not set when REGUTIL.0 is 0. Otherwise, it contains the value of the information read from the registry entry. (KEDIT uses escape sequences to store certain special characters in the registry; in REGUTIL.1 these escape sequences have been replaced by the actual special characters involved)

For example, "REGUTIL GET CONFIG STATE32 PrinterFaceName" might return:

regutil.0 1

regutil.1 "Courier New" (quotes aren't part of the value returned)

Notes

The exact details of how KEDIT stores configuration and history information in the registry are undocumented and are more likely than most other aspects of KEDIT to be subject to incompatible changes between versions of KEDIT. You should not use REGUTIL GET, or the ability to edit the contents of the KEDIT.CONFIG.REG or KEDIT.HISTORY.REG files unless you are aware of and are comfortable with this possibility.

With the default of REGSAVE STATE HISTORY in effect, KEDIT will automatically update the state and history information in the Windows registry at the end of your KEDIT session, overriding the changes made to the registry by any REGUTIL SAVE|CLEAR STATE|HISTORY commands earlier in the session. If you want to avoid this, you can put REGSAVE NOSTATE NOHISTORY into effect.

The Windows registry does not contain values for all of the KEDIT SET options that can be saved. Instead, it only contains values that differ from KEDIT's defaults. Therefore the REGSET.KEX file created by the REGUTIL CONVERT SETTINGS command will only contain SET commands corresponding to saved settings whose values differ from KEDIT's defaults.

For historical reasons the INIUTIL command, which does the same thing as the REGUTIL command, is also available.

See also HISTUTIL, SET REGSAVE

RENAME

Format **REName *fileid1 fileid2***

Description Use KEDIT's RENAME command to change the name of a file on disk. *Fileid1* specifies the disk file to be renamed and *fileid2* specifies the file's new name.

Both fileids can contain path specifications; if the path specification for the second file is different from the path specification for the first file, the RENAME command will move the file from one directory to another. You cannot use RENAME to move a file from one drive to another.

The KEDIT RENAME command changes the name of a file on disk. It does not affect the fileid associated with any of the files currently being edited. Use the SET FILEID command to change the fileid associated a file currently being edited.

Unlike the DOS RENAME command, KEDIT's RENAME command does not allow wildcard characters in either fileid. You can use KEDIT's DOS command to issue the ''real'' DOS RENAME command.

See also DIR, DOS, ERASE, SET FILEID

Examples

```
RENAME ABC.TXT DEF.TXT
```

The file ABC.TXT in the current directory is renamed to DEF.TXT.

REPEAT

Format **REPEat [*target*]**

Description Use the REPEAT command to cause the command in the equal buffer (which is normally the most recently completed command issued from the command line) to be repeatedly executed, affecting additional lines in your file.

When you issue the REPEAT command with no operands, KEDIT moves the focus line pointer down one line in your file and then repeats the last command entered (that is, the last command entered prior to the REPEAT command). REPEAT with no operands is equivalent to

```
=DOWN 1
```

(The ‘‘=’’ command causes KEDIT to re-execute the command in the equal buffer.)

When you specify a target with REPEAT, KEDIT determines the number of lines *n* from the focus line to the target line. KEDIT then repeatedly moves down a line and re-executes the command in the equal buffer, continuing until it has either done this *n* times, or the command in the equal buffer gives a nonzero return code. For example, if you enter

```
CAPPEND X
```

KEDIT will place an ‘‘X’’ at the end of the focus line. If you then enter

```
REPEAT 3
```

KEDIT will, three times, move down one line and repeat the CAPPEND X command. So,

```
CAPPEND X
REPEAT 3
```

is equivalent to

```
CAPPEND X
=DOWN 1
=DOWN 1
=DOWN 1
```

Both of these sequences are also equivalent to

```
CAPPEND X
DOWN 1
CAPPEND X
DOWN 1
CAPPEND X
DOWN 1
CAPPEND X
```

All three of the above sequences have the same effect: the focus line and the three lines following it have an ‘‘X’’ appended to them.

If the target that you specify is located above the focus line, KEDIT moves repeatedly up one line, towards the target line, before repeating the last command.

Using REPEAT in a macro

The command re-executed by the REPEAT command is the command in the equal buffer. This is normally the most recently completed command issued from the command line. The equal buffer is not automatically updated by commands issued from macros, but you can use the SET = command from within a macro to directly set the

contents of the equal buffer. So to use REPEAT in a macro, you will probably precede it with

```
'SET = command'
```

where *command* is the command you want to REPEAT.

Examples

```
REPEAT *
```

KEDIT moves down one line and reissues the last command, repeating this process through the end of the file.

```
REPEAT -1
```

KEDIT moves up one line and reissues the last command.

REPLACE

Format

```
Replace [text]
```

Description

The REPLACE command causes the *text* that you specify to replace the focus line in your file.

If no *text* is specified (the command line consists only of the word "REPLACE"), KEDIT replaces the focus line with a blank line and the blank line becomes the new focus line. With INPUTMODE OFF, the default, the cursor then moves to the left margin column of the focus line. With INPUTMODE LINE or INPUTMODE FULL, you enter KEDIT's Input Mode.

See also

INPUT, SET INPUTMODE

Examples

```
R Hello there.
```

A line consisting of "Hello there." is put in your file, replacing the focus line.

```
REPLACE
```

The focus line is replaced with a blank line. This line becomes the focus line, and KEDIT then acts according to the current INPUTMODE setting.

RESET

Format

```
RESet [Block|CMDSEL|FIELD|Prefix|THIGHlight|UNDO]
```

Description

The RESET command lets you reset various aspects of KEDIT's status. By default, Alt+U issues the RESET BLOCK and RESET THIGHLIGHT commands and the Esc key issues the RESET FIELD command.

RESet Block
KEDIT unmarks any currently marked block.

RESet CMDSEL
KEDIT unmarks any currently marked command line selection.

RESet FIELD
Resets the contents of the cursor field to what it was when the cursor entered the field.

RESet Prefix
Resets the prefix area. All text in the prefix area is ''blanked out'', as if you had typed blanks over all pending prefix commands.

RESet THIGHlight
Displays the currently highlighted target in its normal color.

RESet UNDO
Forces KEDIT to start a new undo level from within a macro. If, for example, you run a macro that changes 10 lines, KEDIT normally groups all 10 changes together into a single undo level that can only be undone as a unit. If your macro changed 5 lines, then issued the RESET UNDO command, and then changed 5 more lines, KEDIT would create 2 undo levels, and you could undo the second group of changes separately from the first group.

RESet
RESET with no operands resets marked blocks and prefix commands, and is equivalent to issuing RESET BLOCK and RESET PREFIX.

RESTORE

Format **RESTore**

Description The RESTORE command is used only in connection with the PRESERVE command. PRESERVE saves the current values of most SET options so that you can temporarily change them, execute some commands with the new settings, and then restore their values with the RESTORE command.

RESTORE restores the settings most recently preserved for the current view of the current file. If you issue the RESTORE command without having first issued the PRESERVE command, you will get an error message. If you issue the RESTORE command twice in a row without an intervening PRESERVE command, you will also get an error message; the preserved settings are saved only until the next RESTORE.

See also PRESERVE

RGTLEFT

Format **RGTLEFT [*n*]**

Description Use the RGTLEFT command, most useful when assigned to a key, to work with text that is a bit too wide to be completely displayed within a window. Issuing RGTLEFT repeatedly will alternate between scrolling text in the window to the right and then to the left, allowing you to view text that won't fit into the window, and then to return to your normal view of the text.

Text scrolls *n* columns to the right or left. If *n* is not specified, text scrolls by three quarters of the number of columns displayed in the window. (If you have used the SET VERIFY command to display multiple sets of columns in the window, text scrolls by three quarters of the width of the first set of columns.)

The RGTLEFT command scrolls the window to the right if the current VERSHIFT value is less than or equal to 0; otherwise, the RGTLEFT command scrolls the window to the left. RGTLEFT works by adjusting the value of VERSHIFT.

You can also scroll right and left in your file by using the mouse to manipulate the horizontal scroll bar.

See also LEFT, RIGHT

RIGHT

Format **RIght [*n*|HALF]**

Description The RIGHT command scrolls your view of the file *n* columns to the right. The RIGHT command does not affect the contents of your file; it only affects which columns of your file are displayed in the document window.

RIGHT with no operands scrolls one column to the right. RIGHT HALF scrolls one half the width of the document window to the right.

For example, assume that you have issued the command

```
SET VERIFY 40 *
```

so that columns 40 through 119 of your file are visible in a window 80 columns wide. Using the RIGHT and LEFT commands, you can scroll the window right or left.

```
RIGHT 10
```

would scroll 10 columns to the right, showing you columns 50 through 129 of your file.

```
RIGHT 20
```

would then scroll an additional 20 columns to the right, showing you columns 70 through 149 of your file.

The RIGHT and LEFT commands affect the value of VERSHIFT. VERSHIFT is a number that represents the number of columns that the display window has been scrolled to the right or left of the columns specified by the VERIFY setting. For example, after the commands RIGHT 10 and RIGHT 20, VERSHIFT will be 30, since the window has been scrolled a total of 30 columns to the right. If you then issued a LEFT 65 command, VERSHIFT would be set to -35 (that is, 30 - 65). (You can QUERY VERSHIFT but cannot directly set it. Its value is controlled by the RIGHT and LEFT commands, and by the RGTLEFT command. KEDIT's AUTOSCROLL facility also works by adjusting the value of VERSHIFT.) The commands RIGHT 0 and LEFT 0 are special cases. They reset the value of VERSHIFT to 0. VERSHIFT is also reset to 0 whenever you issue a SET VERIFY command.

You can also scroll right and left in your file by using the mouse to manipulate the horizontal scroll bar.

See also LEFT, RGTLEFT, SET VERIFY

RIGHTADJUST

Format **`RIGHTAdjust [`*`target`*`]`**

Description Text in the *target* area is right-adjusted, with the text of each line shifted so that the last nonblank character within each line is in the right margin column.

You can use RIGHTADJUST BLOCK, or the Rightadjust Block button on the bottom toolbar, to right-adjust line blocks, box blocks, and one-line stream blocks. Box blocks and one-line stream blocks are given special handling: KEDIT right-adjusts the text within the block boundaries, and text outside the block is not affected. You cannot right-adjust a multi-line stream block.

See also CENTER, LEFTADJUST

Examples

```
RIGHTADJUST
```

The focus line is right-adjusted. This command is assigned by default to Ctrl+R.

```
RIGHTA 6
```

The focus line and the five lines following it, for a total of six lines, are right-adjusted according to the current margin settings.

SAVE, SSAVE

Format

SAVE [*fileid*]
SSave [*fileid*]

Description

The SAVE command causes KEDIT to write the current file to disk. Unlike the FILE command, which writes the file to disk and then removes it from memory, the SAVE command writes the file to disk but does not remove it from memory, and you can continue to edit it.

Any changes you have made to the file are saved and you can continue to edit the copy of the file in the PC's memory. If you then mistakenly delete important parts of your file, or the PC's memory is wiped out by a power failure, all of your work through the last SAVE will have been safely written to disk.

You will normally issue the SAVE command without specifying the optional *fileid* operand. In this case, the file will be written to disk under its current fileid. The current fileid is displayed in the title bar of the document window and, unless you have changed it with a command like SET FILEID, is the fileid you originally used when you began editing the file. You can use the optional *fileid* operand to write the file to disk under a different fileid; the current in-memory fileid does not change. When specifying the *fileid* operand, you can make use of the shortcuts discussed in connection with the SET FILEID command.

You can also save your file using the File Save menu item and the Save button from the toolbar. You can use the File Save As menu item to set a new fileid for the current file and save the file under that fileid.

Like the SAVE command, the SSAVE command also writes the current file to disk. The difference between the commands is that there are conditions under which the SAVE command will give you an error, to warn you that you may be inadvertently overwriting some data, while the SSAVE command will write your file to disk regardless of the possible problem. The FILE and FFILE commands have the same relationship to each other as the SAVE and SSAVE commands; see the description of the FILE and FFILE commands for a discussion of the situations where these possible problems can arise.

See also

FILE, FFILE, QUIT, QQUIT, SET AUTOSAVE, SET BACKUP

Examples

```
SAVE
```

Saves the current file on disk under its current fileid.

```
SAVE A:SAMP1.PAS
```

Saves the file on disk under the name A:SAMP1.PAS.

```
SAVE C:\TEST\
```

Saves the current file in the \TEST directory of the C: drive (which must already exist), using the file's current name and extension.

SCHANGE

Format

SCHange /*string1*/*string2*/ [*target*] [*n*] [*m*]

Description

The SCHANGE (''selective change'') command is similar to the CHANGE command, but it allows you to make changes selectively. To understand this description of the SCHANGE command, you should first read the description of the CHANGE command.

Note that the function of the Edit Replace dialog box is similar to that of the SCHANGE command. The Edit Replace dialog box is more frequently used and in many situations is more convenient.

Where the CHANGE command simply changes all occurrences of *string1* into *string2* in the specified area of the file, SCHANGE highlights each occurrence of *string1* and asks you if you want to make the change. You have three choices:

You can press function key F6. This causes the change to be made. SCHANGE then gives you a chance to undo the change by pressing F6 again before moving to the next occurrence of *string1*.

Your second choice is to press function key F5. This causes SCHANGE to move on to the next occurrence of *string1*.

Your third choice is to press the Esc key. This cancels the SCHANGE command immediately. The current occurrence of *string1* is not changed, and SCHANGE doesn't look for any further occurrences of *string1*.

The operands to SCHANGE are the same as the operands for the CHANGE command and have the same meanings. The only differences are that if you issue the command without giving a target line, CHANGE will only affect the first occurrence of *string1* in the focus line, while SCHANGE will ask you about all occurrences of *string1* in all lines from the focus line through the bottom of the file. If you issue the command with a *target* but without saying how many occurrences in each line are to be processed, the CHANGE command will only affect the first occurrence in each line, while the SCHANGE command will ask you about all occurrences.

The last line scanned becomes the new focus line after the SCHANGE command finishes.

If you enter the SCHANGE command with no operands, KEDIT re-executes the last SCHANGE command that you issued from the command line.

When issued from a macro, the SCHANGE command sets the macro variable SCHANGE.0 to 3, returns the number of occurrences changed in SCHANGE.1, returns the number of lines changed in SCHANGE.2, and returns the number of lines truncated (because the changed text would have extended beyond the truncation column) in SCHANGE.3.

See also CHANGE

SET

Format **`[Set] option value`**

Description The SET command allows you to control how KEDIT carries out many of its functions. You can decide, for example, whether the wordwrap feature is enabled, what the left and right margins should be, and what colors to use when displaying text on your screen.

You can use the Options SET Options dialog box to set the values of most of the SET options.

The SET command is fully described in Chapter 4, "The SET Command".

See also EXTRACT, MODIFY, PRESERVE, RESTORE, QUERY, STATUS

SHIFT

Format **`SHift Left|Right [n [target]]`**

Description The SHIFT command moves the text in all lines in the *target* area LEFT or RIGHT *n* columns. If no *target* is specified, only the text in the focus line shifts. If *n* is not specified either, KEDIT shifts the text in the focus line one column to the left or right.

If STAY ON (the default) is in effect, the focus line location is unchanged after the shift. If STAY OFF is in effect, the last line shifted becomes the new focus line.

The SHIFT command affects text from the left zone column through the truncation column. Text that would be shifted left beyond the left zone column is lost, as is text that would be shifted right beyond the truncation column.

You can use the SHIFT command with line blocks, box blocks, and one-line stream blocks. Box blocks and one-line stream blocks get special handling: text from the left boundary of the box block through the truncation column shifts; the right boundary of the block is ignored. The SHIFT command cannot handle multi-line stream blocks.

Note that you can use Shift+F7 and Shift+F8 to shift the currently marked block left or right by one character, and you can use the Shift Block Left and Shift Block Right buttons on the bottom toolbar for the same purpose.

Do not confuse the SHIFT command, which changes your file by moving text to the left or right, with the LEFT and RIGHT commands, which affect which columns of your file are displayed, but do not change the contents of your file.

Examples

```
SHIFT RIGHT 4 ALL
```

All text in your file is shifted four columns to the right.

```
SHIFT LEFT
```

Text in the focus line is shifted one column to the left.

```
SHIFT RIGHT 1 4
```

Text in the focus line and the three lines following it, for a total of four lines, is shifted one column to the right.

SHOWDLG

Format

SHOWDLG *dialog*

Description

The SHOWDLG command displays and processes one of KEDIT for Windows' built-in dialog boxes. It is used mainly by the menu-handling macros that are activated when you select an item from one of KEDIT's menus. For example, when you use File Print, KEDIT runs the macro MENU_FILE_PRINT, and this macro uses the command

```
SHOWDLG PRINT
```

to show KEDIT's Print dialog box.

The dialog boxes controlled by SHOWDLG, all of which take their name from the KEDIT menu item involved, are:

ABOUTKEDITFORWINDOWS
ARRANGE
BOOKMARK
DIRECTORY
FILL
FIND
GOTO
INTERFACE
OPEN [*defaultdir*]
PRINT
PRINTSETUP
REPLACE

SAVE
SAVEAS
SAVESETTINGS
SCREENFONT
SELECTIVEEDITING
SETCOMMAND [*option*]
SORT
STATUS

Notes

SHOWDLG OPEN takes an optional operand specifying the directory that you would like the File Open dialog box to display when it opens. For example:

```
SHOWDLG OPEN "C:\My Directory"
```

SHOWDLG SETCOMMAND takes an optional operand specifying which SET option should initially be selected when the SET Command dialog box is displayed. For example:

```
SHOWDLG SETCOMMAND WRAP
```

SHOWDLG SAVE is used in the processing of File Save, and usually does not actually display a dialog box; unless an Untitled file is involved, it immediately saves your file to disk under its current name. It is handled by the SHOWDLG command because its processing is similar to that of SHOWDLG SAVEAS.

SORT

Format

SORT *target* [[Ascending|Descending] *n1 m1*] ...

Description

The SORT command sorts the text of the specified *target* area.

KEDIT decides what order to put lines in by comparing characters in the columns specified by your sort fields. Each sort field is expressed as a pair of numbers giving the leftmost and rightmost columns of the field. (You can use an asterisk ("*") instead of the second number; in that case, KEDIT will use the right zone column as the rightmost column.) There can be up to ten sort fields.

If you don't specify any sort fields at all, KEDIT uses the current ZONE settings to determine a sort field, except that if the target area is a box block, KEDIT uses the leftmost and rightmost columns of the block.

You can precede a sort field specification with ASCENDING or DESCENDING, which tells KEDIT to sort that field (and all following fields until you specify otherwise) in ascending or descending order. By default, KEDIT sorts all fields in ascending order.

Unlike most other KEDIT commands, the SORT command processes all lines in the target area, regardless of their selection level, so that even excluded lines are sorted.

You can also access KEDIT's sort facility by using the Actions Sort dialog box.

If STAY ON is in effect, the focus line pointer does not move after the sort is complete. With STAY OFF, the sorted line that ends up closest to the top of your file becomes the focus line.

If the SORT|NOSORT operand of SET INTERNATIONAL has the default value of NOSORT, KEDIT orders text according to the character codes of the characters involved. If the first IGNORE|RESPECT operand of SET CASE is set to IGNORE, uppercase and lowercase versions of the same letter (for example, "c" and "C") are treated as if they were both lowercase. Only the 26 letters from "A" to "Z" and from "a" to "z" are treated as alphabetic. With CASE RESPECT in effect, uppercase and lowercase versions of the same letter are treated as different characters.

If the SORT|NOSORT operand of SET INTERNATIONAL has the value SORT, the sort order used by KEDIT is determined by your Windows language driver. If the first IGNORE|RESPECT operand of SET CASE is set to IGNORE, uppercase and lowercase versions of the same letter (for example, "c" and "C") are treated as if they were both lowercase, with lowercasing also determined by your Windows language driver. With CASE RESPECT in effect, uppercase and lowercase versions of the same letter are treated as different characters.

The SORT command's handling of international characters (that is, alphabetic characters other than the 26 letters of the English alphabet, such as the accented letters found in many European languages) is discussed in connection with the SET INTERNATIONAL command.

See Also

SET INTERNATIONAL

Examples

```
SORT ALL D
```

Sort all lines in the file into descending order, using the current zone settings to define the sort field.

```
SORT 6 4 5 1 3
```

This tells KEDIT to sort six lines, beginning with the focus line, in ascending order according to the data in columns 4 through 5 (the first sort field) and then columns 1 through 3 (the second sort field). If the lines to be sorted looked like this:

```
ABCDE
ABCED
ABCDF
ABEDE
AABBC
ZZZBC

KEDIT would sort them into the following order:

AABBC
ZZZBC
ABCDE
ABEDE
ABCDF
ABCED
```

```
SORT 48 1 12 D 14 30 34 56 A 62 84
```

Sort the focus line and the 47 lines following it in ascending order based on columns 1 through 12, descending order based on columns 14 through 30 and 34 through 56, and ascending order based on columns 62 through 84.

SOS

Format

SOS *action1* [*action2* ...]

Description

The SOS (“screen operation simulation”) command, used primarily in macros, handles a number of specialized editing actions.

Many of these actions involve cursor placement and editing of text within the field containing the cursor. For example, the SOS command lets you delete the character at the cursor position, move the cursor to the next tab position, or move the cursor to the last nonblank character of the field.

SOS also handles a number of miscellaneous actions, such as executing pending prefix commands and beeping the PC's speaker, that aren't handled elsewhere in KEDIT.

SOS ADDline

Adds a blank line after the focus line. KEDIT positions the line in the document window according to the NEWLINES setting. The cursor is moved to the newly-added line.

SOS Alarm
SOS BEEP

The speaker beeps.

SOS BLANKDown
SOS BLANKUp

KEDIT searches for the next blank line. KEDIT searches up (SOS BLANKUP) or down (SOS BLANKDOWN) from the focus line. The search continues until a blank line (that is, a line with no nonblank characters at or to the left of the truncation column), or the top-of-file or end-of-file line, is encountered.

If the cursor is on the command line, the blank line becomes the current line. Otherwise, the cursor moves to the blank line. This line becomes the focus line.

SOS BLOCKEnd
SOS BLOCKStart

If the cursor is on the command line, the first line of the marked block (with SOS BLOCKSTART) or the last line of the marked block (with SOS BLOCKEND) becomes the current line.

Otherwise, the cursor moves to the first or last line of the marked block, with the cursor repositioned to the starting or ending column of a box block or stream block. An error occurs if there is no marked block in the current file.

SOS BOTTOMEdge
The cursor moves to the bottommost line of the file area. If this is below the end-of-file line, the cursor moves to the end-of-file line.

SOS CDn
SOS CLeft
SOS CRight
SOS CUp
The cursor moves one column down, left, right, or up, according to the rules used by the CURSOR ESCREEN command.

SOS CHECK
SOS CHECK displays a message giving a checksum for all characters in the cursor line and for all characters in the cursor line up to the cursor column. It also reports on mismatched quotes and parentheses in the cursor line. SOS CHECK is provided mainly to help Mansfield Software Group verify correct data entry while providing telephone support for KEDIT.

SOS CURRent
The cursor moves to the current line.

SOS CURSORAdj
Text in the cursor field is adjusted so that the first nonblank character of the text is located at the cursor position. The cursor does not move.

SOS DELBAck
The cursor moves one character to the left and then deletes the character at the new cursor position.

SOS DELBEGin
All text in the cursor field from column 1 up to (but not including) the cursor column is deleted, with text starting at the cursor column shifting to column 1 to fill the gap. The cursor's position is unchanged.

SOS DELChar
The character at the cursor position is deleted.

SOS DELEnd
All text in the cursor field, from the character at the cursor position through the end of the field, is deleted.

SOS DELLine
The focus line is deleted, with the line below it becoming the new focus line.

SOS DELSEL
If DELSEL() is true (that is, INTERFACE CUA is in effect, there is an anchored block or command line selection, and the cursor has not moved and the file has not changed since the block or selection was marked), KEDIT deletes the block or selection. SOS DELSEL is used in the default definitions of Del and Bksp when INTERFACE CUA is in effect to delete a block or selection that you have just marked.

SOS DELWord
The "word" (as defined with the SET WORD command) at, or to the right of, the cursor position is deleted.

SOS DOPREfix
KEDIT executes any prefix commands pending for the current file.

SOS ENDChar
The cursor moves to the blank character following the last nonblank (or significant trailing blank) character in the cursor line.

If the cursor line is empty, the cursor moves to the first column of the line.

SOS ENDWord
The cursor moves to the last character of the current ‘‘word’’ (as defined with the SET WORD command).

If the cursor is not in a word, it moves to the end of the next word; if there are no more words in the field, it moves to the end of the last word in the field; if the field is blank the cursor does not move.

SOS ERRORBEEP
If BEEP ON is in effect, the speaker beeps.

SOS EXecute
The cursor moves to the command line, and KEDIT executes any command on the command line.

SOS FIRSTCHar
The cursor moves to the first nonblank character of the cursor line.

If the cursor line is blank, the cursor moves to the first column of the line.

SOS FIRSTCOl
The cursor moves to the first column of the cursor line.

SOS INSTAB
KEDIT inserts blanks into the current field from the cursor position to the next tab position. The cursor moves to the next tab position.

SOS LEFTEdge
If the cursor is on the command line, it moves to column 1 of the command line. Otherwise, it moves to the leftmost column of the file area.

SOS LINEAdd
Same as SOS ADDLINE. Adds a line below the focus line.

SOS LINEDel
Same as SOS DELLINE. Deletes the focus line.

SOS MAKECURR
If the cursor is on the command line, nothing happens. Otherwise, the cursor line becomes the current line.

SOS MARGINL
The cursor moves to the left margin column of the cursor line.

SOS MARGINR
The cursor moves to the right margin column of the cursor line.

SOS MOUSEBEEP
If MOUSEBEEP ON is in effect, the speaker beeps.

SOS PARINDent
The cursor moves to the paragraph indent column of the cursor line.

SOS PREfix
If the cursor line has a prefix area, the cursor moves to the first column of the prefix area.

SOS QCmnd
KEDIT moves the cursor to the first column of the command line and clears the contents of the command line.

SOS QUICKFINDACT
KEDIT activates the Quick Find toolbar item, so that you can edit the string that it contains.

SOS QUICKFINDB
SOS QUICKFINDf
KEDIT searches forward (SOS QUICKFINDF) or backward (SOS QUICKFINDB) in your file for the Quick Find string. That is, KEDIT searches forward or backward for the string currently displayed in the Quick Find toolbar item.

SOS RESTORE
SOS RESTORECol
SOS RESTORELine
KEDIT restores the cursor position to the location it had in the current window when the position was last saved with SOS SAVE, SOS SAVECOL, or SOS SAVELINE. If no cursor position had previously been saved, KEDIT restores the cursor to the first column of the command line.

With SOS RESTORE, both the line and column position within the window of the cursor are restored.

With SOS RESTORECOL, the column position is restored, but the cursor remains on the same line of the document window.

With SOS RESTORELINE, the line position is restored, but the cursor stays in the same column.

SOS RETRIEVEb
SOS RETRIEVEF
KEDIT cycles backward or forward through lines of text previously entered on the command line, and redisplays them on the command line. If the command line is empty, then all previous commands will be displayed. If you have entered any text on the command line, then only commands beginning with that text will be displayed.

SOS RIGHTEdge
If the cursor is on the command line, it moves to the rightmost column of the command line. Otherwise, it moves to the rightmost column of the file area.

SOS SAVE
SOS SAVECol
SOS SAVELine

KEDIT saves the cursor position within the current window so that SOS RESTORE can later be used to restore the cursor position.

SOS SAVE saves both the line and column position of the cursor.

SOS SAVECOL saves only the column position, without changing any previously saved line position.

SOS SAVELINE saves only the line position, without changing any previously saved column position.

SOS SETCOLPtr

If the cursor is in the prefix area, no action is taken. Otherwise, the column pointer is set to point to the cursor column.

SOS SETLeftm

If the cursor is in the prefix area, no action is taken. Otherwise, the left margin column is set to the cursor column.

SOS SETTAB

If the cursor is in the prefix area, no action is taken. Otherwise, the cursor column is added to the list of tab columns.

SOS STARTWord

The cursor moves to the first character of the current ‘‘word’’ (as defined with the SET WORD command).

If the cursor is not in a word, it moves to the start of the preceding word; if there are no preceding words, it moves to the start of the first word on the line; if the line is blank the cursor does not move.

SOS TABB

The cursor moves to the nearest tab position to the left of its current location.

If the cursor is on the command line and there are no preceding tab positions, the cursor moves to column 1 of the command line. Otherwise, if there are no tab positions to the left of the cursor column, the cursor moves to the last tab position on the line of the file above the cursor line.

SOS TABCmd

The cursor moves to the first column of the command line of the current window.

SOS TABCMDB
SOS TABCMDF

The cursor moves to the previous or next document window, to the first column of the new window’s command line.

SOS TABf

The cursor moves to the next tab position to the right of its current location.

If the cursor is on the command line and there are no more tab positions, the cursor moves to the end of the command line. Otherwise, if there are no tab positions to the right of the cursor column, the cursor moves to the first tab position on the line of the file below the cursor line.

SOS TABFIELDB
The cursor moves to the first character of the current field.

If the cursor is already in the first character of the current field, it moves to the first character of the previous field, wrapping from the first field of the document window to the last field of the document window if necessary.

SOS TABFIELDf
The cursor moves to the first character of the next field, wrapping from the last field of the window to the first field of the document window if necessary.

SOS TABWORDB
The cursor moves to the beginning of the first "word" (as defined with the SET WORD command) to the left of its current position.

If there are no words to the left of the cursor, the cursor moves to the first column of the line.

SOS TABWORDf
The cursor moves to the beginning of the first "word" (as defined with the SET WORD command) to the right of its current position.

If there are no words to the right of the cursor, the cursor does not move.

SOS TOPEdge
The cursor moves to the topmost line of the file area. If this is above the top-of-file line, the cursor moves to the top-of-file line.

See also CURSOR, TEXT

Examples

```
SOS BEEP
```

KEDIT beeps the PC's speaker.

```
SOS CDN DELCHAR TAB
```

The cursor moves down one line, and the character at which the cursor is then positioned is deleted. The cursor then moves to the next tab column.

SPLIT

Format `SPlit [ALigned]`

Description The SPLIT command splits a line into two lines. The SPLIT command is usually issued from a macro assigned to a key. (It is assigned to Alt+S by default.) Text to the left of the focus column remains in the focus line, while text in and to the right of the focus column is split off to form a new line.

SPLIT with no operands positions the split-off text to begin in column one of the new line. SPLIT ALIGNED (which Alt+S normally uses) adds as many leading blanks to the new line as there are in the focus line. This is useful when you are working with

indented text, and you would like the text in the new line to be indented to the same column as the focus line.

The SPLIT command does not affect the position of the focus line or focus column.

See also JOIN, SPLTJOIN

Examples Assume that the focus line looks like the following, with the first nonblank character of the focus line in column 5 and the cursor positioned at the "S" of the word "Split":

```
____This is the text to Split into two
```

Pressing a key to which SPLIT is assigned would start the split-off text in column one, yielding

```
____This is the text to
Split into two
```

A key with SPLIT ALIGNED, on the other hand, would preserve the existing indentation, giving

```
____This is the text to
____Split into two
```

SPLTJOIN

Format **SPLTJOIN**

Description The SPLTJOIN command does either a SPLIT ALIGNED or a JOIN ALIGNED, depending on the cursor position. SPLTJOIN must be issued from a macro with the cursor positioned in the file area; it is not valid if issued from the command line. SPLITJOIN is assigned to function key F11 by default. If you issue the SPLTJOIN command with the cursor at or to the left of the end of a line, SPLTJOIN splits the line into two lines. When the cursor is past the last nonblank character of a line, SPLTJOIN joins together the contents of that line and the line below it.

When SPLTJOIN splits a line, it splits it at the cursor position, as if you had issued a SPLIT ALIGNED command. When SPLTJOIN joins a line with the line below it, it joins at the cursor column, as if you had issued a JOIN ALIGNED command.

See also JOIN, SPLIT

STATUS

Format **STATus**

Description Using the STATUS command is equivalent to using the Options Status menu item: KEDIT displays a dialog box with the current values of most SET options.

See also Chapter 4, "The SET Command", Chapter 5, "QUERY and EXTRACT"

SYNEX

Format **SYNEX *command***

Description KEDIT usually checks each command issued from the command line to see if you have used the SET SYNONYM command to redefine its behavior. If so, KEDIT processes the command as specified in the synonym definition. This synonym processing is normally bypassed for commands issued from macros. The SYNEX (''synonym execute'') command specifically requests that (unless SYNONYM OFF is in effect) synonym processing apply to a command, even when it is issued from a macro.

SYNEX is useful only when issued from a macro, because synonym processing applies by default to commands issued from the command line.

See also COMMAND, SET SYNONYM

Examples

```
SYNEX DELETE 3
```

KEDIT checks to see if you have defined a synonym for the DELETE command. If so, KEDIT processes the synonym. Otherwise, KEDIT deletes three lines from your file.

TAG

Format **TAG [*target*]**

Description The TAG command lets you highlight all lines in your file that match a specified target.

The TAG command sets the tag bit of all lines matching a specified target, clears the tag bit of lines that do not match the target, and then puts HIGHLIGHT TAGGED into effect so that the matching lines are highlighted on your display.

Issuing the TAG command with no operands clears the tag bits of all lines in the file.

See also User's Guide Chapter 8, "Selective Line Editing and Highlighting", LESS, MORE, SET HIGHLIGHT

Examples

```
TAG /yesterday/
```

Tags and highlights all lines in your file that contain the string ''yesterday''.

```
TAG BLANK
```

Tags and highlights all blank lines in your file.

```
TAG
```

Turns off the tag bits of all lines in your file, so that no lines are highlighted.

TEXT

Format **TEXT** ***text***

Description The TEXT command allows you, normally from within a macro, to simulate typing the specified *text* at the cursor position. KEDIT processes the text just as if you had entered it from the keyboard, with the handling of the text affected by the cursor position, the settings of INSERTMODE and WORDWRAP, etc.

In fact, whenever you type text into KEDIT from the keyboard, the text is actually being passed to KEDIT by the TEXT command. For example, the macro normally assigned to the A key is ''text a'', which enters a lowercase ''a'' at the cursor position. The default macro for Shift+A is ''text A'', which enters an uppercase ''A'' at the cursor position.

If HEX ON is in effect, you can specify the *text* using hexadecimal or decimal notation, as discussed in the description of SET HEX.

Examples

```
TEXT abcd
```

The text ''abcd'' is processed by KEDIT as if you had typed the text at the cursor position.

```
TEXT x'61626364'
```

Assuming HEX ON is in effect, this example also causes KEDIT to process the text ''abcd'' (specified by giving the character codes for the characters in hexadecimal).

TFIND

Format **TFind [*target*]**

Description

The TFIND command (''target find'') makes the line referred to by the specified target become the new focus line. If the target is a string target, however, the string must start in the left zone column of the line.

The TFIND command is used when you want to look for a string that starts in a particular column. Since the left zone column is usually set to column 1, TFIND is most often used to search for a string that starts in column 1 of a line. This can be useful, for example, when working with assembler language programs, where labels are normally defined beginning in column 1 of a line.

You can specify any type of target with the TFIND command, but for targets other than string targets, the TFIND command works exactly like the LOCATE command. With a string target, the TFIND command only finds occurrences of the string starting in the left zone column, while the LOCATE command will locate occurrences anywhere between the left zone and right zone columns.

If you issue the TFIND command with no operands, KEDIT will re-execute the last TFIND command you issued, looking again for the same target.

The action of the TFIND command is affected by the settings of HEX, WRAP, STAY, VARBLANK, ARBCHAR, CASE, ZONE, and THIGHLIGHT.

See also FIND, LOCATE

Examples

```
TFIND /ABC/
```

The next line that has, beginning in the left zone column, ''ABC'', becomes the focus line.

```
LOCATE /ABC/
```

The next line that contains an ''ABC'' anywhere within the current zone becomes the focus line.

```
TFIND ~/9/
```

The next line that does not have a ''9'' in the left zone column becomes the focus line.

```
LOCATE ~/9/
```

The next line that does not contain, anywhere within the current zone, a ''9'' becomes the focus line.

TOP

Format **TOP**

Description The TOP command makes the top-of-file line become the focus line. With INTERFACE CUA in effect, you can also press Ctrl+Home to get to the top of your file. With INTERFACE CLASSIC you can instead press Ctrl+Page Up.

See also BOTTOM

UNDO

Format **UNDO**

Description The UNDO command, normally assigned to Alt+Bksp and to Ctrl+Z, will undo one level of changes to the current file. You can issue the UNDO command repeatedly to undo additional changes.

You can also undo an action using the Edit Undo menu item or using the Undo button on the toolbar.

Whenever it is possible to UNDO previous actions, the available undo level count (the third number following "Alt=" on the status line) will be nonzero.

See also User's Guide Chapter 3, "Using KEDIT for Windows", REDO, RESET, SET UNDOING

UNLOCK

Format **UNLOCK**

Description Use the UNLOCK command to allow other users to access the disk copy of a locked file that you are editing.

The UNLOCK command causes KEDIT to unlock the current file. That is, KEDIT closes the file handle associated with the disk copy of the current file.

Attempting to UNLOCK a file that is not locked causes an error message. If the current file is locked, KEDIT displays "Locked" to the right of the status line. Additionally, if the current file is locked and IDLINE ON is in effect, its fileid will be preceded on the idline with an asterisk.

See also User's Guide Chapter 12, "File Processing", LOCK, SET LOCKING

UP

Format **`Up [n]`**

Description The line *n* lines above the focus line becomes the new focus line. If *n* is not specified, the line above the focus line becomes the focus line.

See also DOWN

Examples

```
UP
```

The line above the focus line becomes the new focus line.

```
UP 4
```

The line four lines above the focus line becomes the new focus line.

UPPERCASE

Format **`UPPercase [target]`**

Description Use the UPPERCASE command to convert lowercase characters in a specified portion of your file to uppercase characters.

If you give no operands, the UPPERCASE command converts the focus line to uppercase. Otherwise, all text within the specified *target* area that falls within the current ZONE columns is converted. If the target area is a box block, its entire contents are converted, regardless of the ZONE settings. If STAY OFF is in effect, the last line uppercased becomes the focus line. Otherwise, the focus line location does not change.

With the default of INTERNATIONAL NOCASE in effect, LOWERCASE treats only the 26 letters from ''A'' to ''Z'' and from ''a'' to ''z'' as alphabetic. With INTERNATIONAL CASE in effect the characters to be treated as alphabetic, and what their lowercase equivalents are, are determined by your Windows language drivers.

Note that you can use the default definition of Shift+F5, the Actions Uppercase menu item, or the Uppercase Block button on the bottom toolbar to uppercase a block of text.

See also LOWERCASE, SET INTERNATIONAL

Examples

```
UPPERCASE
```

All lowercase characters in the focus line are converted to uppercase.

UPPER -4

All lowercase characters in the focus line and in the three lines above it, for a total of four lines, are converted to uppercase.

WINDOW

Format

```
WINdow MINimize|MAXimize|RESTORE [DOCument|FRAME]
WINdow CASCADE
WINdow TILE [Horizontally|Vertically]
WINdow ARRANGE [Horizontally|Vertically]
WINdow CLOSE [DOCument|FRAME|FILE]
WINdow NEWwindow
WINdow NEXTwindow|PREVwindow
WINdow ARRANGEIcons
WINdow MENUmode [menuname]
```

Description

The WINDOW command performs assorted operations related to the sizing and positioning of KEDIT's frame window or of your document windows. The WINDOW command is most often used in macros and is not usually issued directly from the command line. For example, the macro that handles the Window Cascade menu item issues the WINDOW CASCADE command to tell KEDIT to cascade your document windows.

WINdow MINimize|MAXimize|RESTORE [DOCument|FRAME]
KEDIT minimizes a window, maximizes a window, or restores a window to the normal (non-minimized, non-maximized) state. The operation can affect the current DOCUMENT window (this is the default) or it can affect the FRAME window.

WINdow CASCADE
KEDIT cascades its document windows. This is equivalent to using the Window Cascade menu item.

WINdow TILE [Horizontally|Vertically]
KEDIT tiles its document windows either HORIZONTALLY (this is the default) or VERTICALLY, as it does when you use the Window Tile Horizontally or Window Tile Vertically menu items.

WINdow ARRANGE [Horizontally|Vertically]
KEDIT arranges its document windows either HORIZONTALLY (this is the default) or VERTICALLY, as it does when you use the Window Arrange... menu item.

WINdow CLOSE [DOCument|FRAME|FILE]
KEDIT closes either the current DOCUMENT window (this is the default), the FRAME window, or the current FILE. Closing the document window is equivalent to selecting Close from the document window's system menu. Closing the frame window is equivalent to selecting Close from the frame window's system menu, or to using File Exit. Closing the file is equivalent to using the File Close menu item.

WINdow NEWwindow
KEDIT creates a new document window, giving you an additional view of the current file. This is equivalent to using the Window New Window menu item.

WINdow NEXTwindow|PREVwindow
KEDIT makes the next or previous document window become the current document window. This is the default behavior of Ctrl+F6 (which moves to the next document window) or Shift+Ctrl+F6 (which moves to the previous document window).

WINdow ARRANGEIcons
KEDIT rearranges any minimized document windows so that they are lined up neatly near the bottom of the frame window; this is equivalent to using the Window Arrange Icons menu item.

WINdow MENUmode [*menuname*]
KEDIT activates its menu bar. To display and activate a specific menu, you can specify *menuname*, which can be File, Edit, Actions, Options, Window, Help, DOCument (to activate the document window's system menu), or FRAME (to activate the frame window's system menu).

WINEXEC

Format

WINEXEC [WAIT|NOWAIT] [NORMal|MAXimize|MINimize] ***command***

Description

Use the WINEXEC command to run an external Windows or DOS program.

Specify the *command* involved in the same way as you would when using the Run option from the Windows Start Menu: give the name of the module involved (possibly including a drive and/or path specifier) followed by any command line parameters that you want to pass to the module.

You can precede the command with an option specifying either WAIT (this is the default) or NOWAIT. With the WAIT option, KEDIT is inactive and can accept no further mouse or keyboard input until the external command completes. With the NOWAIT option, KEDIT remains active after the external command starts and you can interact with KEDIT without waiting for the external command to complete.

You can also precede the command with an option that specifies whether the program that you execute should start out with a NORMAL window (that is non-maximized, non-minimized window; this is the default), a MAXIMIZED window, or a MINIMIZED window.

Notes

While you can run both Windows and text mode Command Prompt-style programs via the WINEXEC command, it is usually preferable to run text mode programs via KEDIT's DOS command or the related DOSNOWAIT or DOSQUIET commands, and to use the WINEXEC command for running Windows programs. This is because the DOS command will automatically handle commands that are internal to CMD.EXE, will insure that the text mode program's output remains on

the screen until you have had a chance to see it, and will remove KEDIT for Windows from the screen while waiting for your text mode program to complete, while the WINEXEC command does none of these.

When issued from a macro WINEXEC WAIT (but not WINEXEC NOWAIT) returns information about the exit code set by the command that was executed. Details on this are given in the discussion of the DOS command.

See also DOS

Examples

```
WINEXEC NOTEPAD
```

Starts the Windows Notepad program. Since WAIT and NORMAL are in effect by default, Notepad starts up with its default window size and KEDIT is inactive until Notepad has finished running.

```
WINEXEC NOWAIT NOTEPAD C:\AUTOEXEC.BAT
```

Starts the Windows Notepad program, with its default window size, and tells it to edit the file C:\AUTOEXEC.BAT. Because the NOWAIT option is specified, KEDIT does not wait for Notepad to finish execution. Both programs are active at once and you can switch between the two programs and do work in either of them.

WINHELP

Format **WINHELP *fileid* [*topic*]**

Description Use the WINHELP command to invoke the Windows Help program so that you can view a Windows Help file.

The WINHELP command is available mainly for compatibility with previous versions of KEDIT. It is useful only with older, Windows 3.1-style, .HLP Help files and does not work with the .CHM HTML Help files more commonly used by current Windows applications. In particular, it does not work with the current KEDIT for Windows Help file, which uses the newer HTML Help format.

Specify the *fileid* of the Help file that you want to view. You should include the .HLP file extension and (unless the Help file is in the current directory, in the Windows directory, or in your PATH), the drive and path specifier for the Help file.

You can optionally supply a *topic* to search for in the Help file. The Windows Help program will then jump to that topic in the Help file or, if there are either no matching topics or more than one matching topic, will display its Search dialog box. If you do not specify a *topic*, Windows Help will display the Help file's Contents page.

See also HELP

Examples `WINHELP C:\SDK31\WIN31WH.HLP ShowCaret`

Assuming that the Help file WIN31WH.HLP is installed in your C:\SDK31 directory, this command would tell Windows Help to display that Help file and to search for help on the topic ShowCaret.

WMSG

Format **WMSG *command***

Description WMSG, used mainly in macros, executes a command and displays any error message that the command generates in a Windows message box instead of on the message line.

See also CMSG, DMSG, EMSG, MSG

Examples `WMSG LOCATE /Tomorrow/`

This example, which would be in quotes if it were included in a macro, looks for the string ''Tomorrow''. If the string is found, KEDIT makes the line that contains the string become the focus line, in the normal way. But if an error is encountered, the error message is not displayed within the document window, on the message line, but is instead displayed in a pop-up Windows message box.

XEDIT

Format **Xedit [*fileid* ...] [(*options* [)]]**

Description Use the XEDIT command to begin editing one or more additional files. See the description of the KEDIT command for details on the XEDIT command. The XEDIT command performs exactly the same functions as the KEDIT command.

See also KEDIT

&

Format ***&commandline***

Description When you precede a command line with an ampersand (''&''), KEDIT handles all the commands on the command line in the normal way and then redisplays the commands (preceded by an ''&'') on the command line. This allows you to easily re-execute the command line repeatedly, and lets you make changes to the command before re-entering it.

Examples

```
&LOCATE /12.4/
```

KEDIT locates the next line containing "12.4", then redisplays

```
&LOCATE /12.4/
```

on the command line.

?

Format

```
? [+]
```

Description

The ? command, normally assigned to function key F6, causes KEDIT to redisplay on the command line the last command entered on the command line. You can then edit the command and re-enter it.

You can issue the ? command repeatedly to get back the second-to-the-last command entered, then the third-to-the-last, etc., eventually cycling back to the most recent command. KEDIT saves the last 40 command lines entered. As a shortcut, you can enter multiple question marks on the same command line. If you enter two consecutive question marks on the same command line, KEDIT shows you the second-to-the-last command entered. Three question marks give you the third-to-the-last, etc.

KEDIT normally moves backward through the set of saved command lines, from the most recent to the least recent. If you enter one or more question marks followed by a plus sign (for example, "?+"), KEDIT will cycle forward through the saved command lines, from least recent to most recent.

Any text on the command line following the question marks (and possible plus sign) is ignored.

See also

SOS RETRIEVEB, SOS RETRIEVEF

Examples

```
DELATE 4
?
```

In this example, you want to issue the command DELETE 4 but you have spelled the word "DELETE" wrong. Instead of retyping the entire command line, you can simply issue the ? command (either by typing it in or by pressing F6). KEDIT will then redisplay "DELATE 4" on the command line, and you can simply change the "A" in DELATE to an "E" and press Enter.

```
??
```

KEDIT redisplays the second-to-the-last command line entered.

=

Format **= [*command*]**

Description

The = command, assigned by default to function key F9, causes KEDIT to re-execute the command in the equal buffer, which is normally the most recently completed command issued from the command line. For example, if you type in

```
DOWN 3
```

to cause KEDIT to move down three lines in the file and you then enter

```
=
```

KEDIT will move down three more lines.

You can enter several ‘‘=’’s in a row to cause KEDIT to re-execute a command several times.

```
===
```

would cause the last command to be re-executed three times.

The ‘‘=’’ can optionally be followed by a command that you want KEDIT to process before the last command is re-executed. For example, assume that you enter these commands:

```
DOWN 3
=TOP
```

DOWN 3 causes KEDIT to move down three lines in your file. =TOP tells KEDIT to re-execute the DOWN 3 command after first moving to the top of your file, so the net effect of this sequence is to make line 3 of your file become the focus line.

Commands issued from macros, as opposed to the command line, are not automatically put into the equal buffer and made available for the = command, but you can use the SET = command to set the contents of the equal buffer from within a macro.

Commands issued via menu and toolbar operations do not affect the contents of the equal buffer.

See also REPEAT, SET =

Chapter 4. The SET Command

The SET command lets you control how KEDIT carries out many of its functions. You can decide, for example, whether the wordwrap feature is enabled, what the left and right margins should be, and what colors to use to display text on your screen. The general format of the SET command is

```
[Set] option value
```

For example,

```
SET MARGINS 1 60 5
SET TABS 1 10 16 30
SET WRAP OFF
```

You are allowed, however, to leave out the word "SET" and simply specify the option and its value. (SET ALT and SET = are exceptions; SET must be specified for these options to avoid confusion with the ALTER and = commands.) So the previous examples could in fact have been entered like this:

```
MARGINS 1 60 5
TABS 1 10 16 30
WRAP OFF
```

Format

For each SET option, the discussion involves:

The format of the corresponding SET command, including the operands involved and the minimal truncations that you can specify.

The default value of the option.

The level at which the option takes effect; this can be the Global, File, or View level, and is discussed in more detail below.

The dialog box that you can use, as an alternative to issuing the SET command from the command line, to control the value of the option. In most cases, this is the Options SET Command dialog box. Some of the more specialized options do not have a corresponding dialog box and can only be changed via the SET command.

Information on whether the value of the SET option can be saved in the Windows registry so that it will take effect in future KEDIT sessions.

The bulk of the documentation for most SET options consists of a description of the option and its operands, along with examples of their use.

Default value

The following pages contain a discussion of each of the options that can be set. Each option has a default value, which is in effect until you change it. There are several ways in which the default value of an option can be changed:

At the start of each session KEDIT processes the KEDIT section of the Windows registry. Among other things, KEDIT's registry section contains settings that you

changed in previous sessions and then saved, using the Options Save Settings menu item or related facilities, so that they would take effect in future KEDIT sessions.

At the start of each session KEDIT also processes your profile macro, which is normally called WINPROF.KEX. Your profile is processed after the settings in the registry, and changes made to your settings by SET commands issued from your profile override any changes made to those same settings via the registry.

Finally, you can change the values of SET options at any time during a KEDIT session by issuing SET commands from the command line or from macros, or by using the Options SET Command dialog box.

Level

The description of each SET option indicates 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. 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.

Saved settings

The values of most SET options can be saved in KEDIT's section of the Windows registry. These saved settings are then put back into effect at the start of each future KEDIT session.

The values of the following SET options can be saved: ARBCHAR, ARROW, AUTOEXIT, AUTOINDENT, AUTOSAVE, AUTOSCROLL, BACKUP, BEEP, BOUNDMARK, CASE, CLOCK, CMDLINE, COLMARK, COLOR, CURLINE, CURRBOX, CURSORSIZE, CURSORTYPE, DEFEXT, DEFPROFILE, DEFSORT, DIRFORMAT, DOCSIZING, ECOLOR, EOFIN, EOFOUT, EOLIN, EOLOUT, FCASE, FORMAT, HELPDIR, HEX, HEXDISPLAY, HIGHLIGHT, IDLINE, IMPMACRO, INITIALDIR, INITIALDOCSIZE, INITIALFRAMESIZE, INITIALINSERT, INITIALWIDTH, INPUTMODE, INTERFACE, INTERNATIONAL, KEYSTYLE, LINEND, LOCKING, MACROPATH, MARGINS, MARKSTYLE, MONITOR, MOUSEBEEP, MSGLINE, NEWLINES, NOVALUE, NUMBER, OFPW, PATH, PCOLOR, PREFIX (but not PREFIX SYNONYM), PREFIXWIDTH, PRINTCOLORING, PRINTER, PRINTPROFILE, RECENTFILES, RIGHTCTRL, SCALE, SCROLLBAR, SHADOW, SHARING,

STATUSLINE, STAY, STREAM, SYNONYM ON|OFF, TABLINE, TABS, TABSIN, TABSOUT, THIGHLIGHT, TIMECHECK, TOFEOF, TOOLBAR, TRAILING, UNDOING, VARBLANK, WINMARGIN, WORD, WORDWRAP, and WRAP.

You can use Edit Save Settings or the command REGUTIL SAVE SETTINGS to update the values of all of these options in the Windows registry. You can update individual values by using the Save Setting button within the Options SET Command dialog box, or by using the REGUTIL SAVE SET *option* command. Not all of these option values are actually written to the 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 Windows 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.

Unsupported SET options

The following SET options, supported in earlier versions of KEDIT, are not used by this version of KEDIT for Windows: BLINK, BORDER, CURSORSHAPE, EAPRESERVE, FILEOPEN, KEYBOARD, LOGO, MOUSE, MOUSEBAR, PSCREEN, RETRACE, REXXIO, SHIFTSTATE, SWAP, SYSRC, and TOPVIEW. Any SET commands issued from the command line for these options will yield an error message. SET commands for these options issued from macros will yield a return code of 4 and will be otherwise ignored; avoiding an error message in this situation means that many existing macros that use these options will continue to work. Additionally, QUERY and MODIFY commands for these options will yield an error message, while EXTRACT commands and implied EXTRACT functions involving these options will return default information.

SET ATTRIBUTES is supported in KEDIT for Windows for compatibility with earlier versions of KEDIT, but it is not documented here because new users are encouraged to use SET COLOR, which is now the preferred alternative. QUERY, MODIFY, and EXTRACT ATTRIBUTES are also still available.

SET MOUSETEXT is supported in KEDIT for Windows for compatibility with earlier versions of KEDIT, but it is not documented here because new users are encouraged to use SET TOOLBUTTON and SET TOOLSET. QUERY, MODIFY, and EXTRACT MOUSETEXT are not supported. KEDIT for Windows handles SET MOUSETEXT commands by internally converting them to the equivalent SET TOOLSET BOTTOM commands, and your text mode mousebar text is displayed as on KEDIT's bottom toolbar. The bottom toolbar is not displayed by default, but can be controlled via the SET TOOLBAR command.

See also EXTRACT, MODIFY, PRESERVE, RESTORE, STATUS, Chapter 5, "QUERY and EXTRACT"

SET ALT

Format **`Set ALT`** ***`n1`*** **`[`*** ***`n2`*** **`]`**

KEDIT default: 0 0

Level: File

Dialog box: None

Save Settings handling: Not savable

Description SET ALT, used mainly in macros, lets you change KEDIT's alteration counts. KEDIT keeps track of two types of ''alteration count''. The first is the number of changes since your file was last saved or autosaved. (Your file is saved when you issue the SAVE command or use the File Save or File Save As menu items.) The second is the number of changes since the last save, regardless of intervening autosaves. Both alteration counts are incremented whenever you issue a command that changes the contents of your file. A single command may affect many lines of your file, but will only add 1 to the alteration counts. A macro that issues a large number of commands may increment the alteration counts many times.

The first alteration count is used by the autosave facility. Whenever enough changes have been made to your file since the last save or autosave (you set the threshold for this with the SET AUTOSAVE command), KEDIT will autosave your file.

The second alteration count is used by the QUIT command. If you try to quit from a file which has been altered since the last save (that is, its second alteration count is not zero), KEDIT will not quit the file but will give you a message warning you that the file has been changed. The second alteration also affects, when you use File Close, whether or not KEDIT asks if you want to save the file.

After every successful autosave, the value of the first alteration count is reset to zero. After every successful save, the values of both alteration counts are reset to zero. The UNDO and REDO commands also affect the alteration count.

KEDIT displays the alteration counts on the status line as the first two numbers after ''Alt=''; the third number following ''Alt='' indicates how many levels of changes are available to KEDIT's undo facility.

See also SET AUTOSAVE

Examples

```
SET ALT 100
```

This sets the first alteration count for your file to 100. The word "SET" is required for the ALT option to avoid confusion with the ALTER command. If AUTOSAVE is set to some number less than 100 when this command is issued, an autosave will immediately take place and the alteration count will be reset.

```
SET ALT 0 0
```

This sets both alteration counts for the current file to 0. You could then QUIT from the file even if it had been changed, since the QUIT command is allowed if the second alteration count is zero.

SET ARBCHAR

Format

[Set] ARBchar ON|OFF [*char1*] [*char2*]

KEDIT default: OFF $?

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description

When ARBCHAR ("arbitrary character") is ON, the characters *char1* and *char2* (known as the ARBCHAR characters) have a special meaning when used in string targets, string column targets, with the CHANGE and SCHANGE commands, and with the Edit Find, Edit Replace and Edit Selective Editing dialog boxes.

When ARBCHAR is on and KEDIT is looking for a string, the ARBCHAR characters act as wildcard characters. The first ARBCHAR character, usually a dollar sign ("$"), will match any group of zero or more characters. The second ARBCHAR character, usually a question mark ("?"), will match any single character. For example, assume that

```
ARBCHAR ON $ ?
```

is in effect and you enter

```
/ab$c/
```

Then any of the following strings would be matched:

```
abc
abbc
abfgfghc
```

They would be matched because the search string you gave will match any string of characters containing an "a" followed immediately by a "b" followed by any group of zero or more characters followed by a "c". So in the first string above, the "$"

matched the empty string. In the second string above the ‘‘$’’ matched the second ‘‘b’’. In the third string, the ‘‘$’’ matched ‘‘fgfgh’’.

If you enter

```
/ab?c/
```

then ‘‘?’’, the second ARBCHAR character, will match any single character. So, of the three strings above, ‘‘/ab?c/’’ will only match the second string, ‘‘abbc’’, since that is the only one consisting of an ‘‘a’’, a ‘‘b’’, exactly one other character, and then a ‘‘c’’.

You can use more than one arbitrary character in the same search string. For example, the target

```
/The$saw$boy./
```

would match

```
The man saw the small boy.
```

Note that it would not match

```
The man the boy saw.
```

When ARBCHAR is OFF, the ARBCHAR characters have no special meaning, so they match only themselves.

When you use the first ARBCHAR character (usually a ‘‘$’’) as the first character in the string to be changed by a CHANGE or SCHANGE command, or with the Edit Replace dialog box, it will match all characters starting from the left zone column. If it is the last character, it will match all characters through the right zone column. If it is the only character in the string to be changed, it will match all characters from the left zone column through the right zone column.

Assume that ZONE 1 8 is in effect and that the focus line is

```
12345678
```

The command

```
C/$3/A/
```

would change the focus line to

```
A45678
```

The command

```
C/3$/A/
```

would give you

```
12A
```

while

```
C/$/A/
```

would give you

```
A
```

You can also use the first ARBCHAR character, normally "$", on the right side of a CHANGE or SCHANGE command, or in the Edit Replace dialog box. The first "$" on the right side of a CHANGE command is replaced by all characters matched by the first "$" on the left side, the second "$" on the right is replaced by characters matching the second "$" on the left, etc. There must be at least as many "$"s on the left as on the right. (The second ARBCHAR character, normally "?", works similarly.)

Again, assume that ZONE 1 8 is in effect and the focus line contains

```
12345678
```

The command

```
C/1$8/A$Z/
```

gives

```
A234567Z
```

With ZONE 1 8 still in effect and "12345678" on the focus line

```
C/$4$5$/$A$B$C/
```

gives

```
123AB678C
```

Examples

```
ARB ON
```

This sets ARBCHAR ON, so that the current ARBCHAR characters take on the special meanings discussed above when used in string searches. Unless you say otherwise, "$" and "?" are the ARBCHAR characters.

```
ARB OFF
```

This sets ARBCHAR OFF, so that the ARBCHAR characters have no special meaning in string searches.

```
ARB ON * +
```

This turns ARBCHAR ON and sets "*" up as the character that matches any group of zero or more characters and "+" as the character that matches any single character.

```
ARB ON [
```

This turns ARBCHAR ON and sets "[" as the character that matches any group of zero or more characters. The character that matches any single character is unchanged from the value already in effect.

SET ARROW

Format **`[Set] ARRow ON|OFF`**

KEDIT default: ON

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description With ARROW ON, the default, KEDIT displays an arrow (“====>”) at the beginning of the command line. This makes it easier for you to keep track of whether the cursor is on the command line or in the file area. With ARROW OFF, the arrow does not appear.

SET AUTOCOLOR

Format **`[Set] AUTOCOLOR`** ***`.ext parser`***

KEDIT default: See the table below

Level: Global

Dialog box: None

Save Settings handling: Not savable

Description Use SET AUTOCOLOR to determine the syntax coloring parser used for files with a specified extension.

When the default of COLORING ON AUTO is in effect for a file, KEDIT decides which parser to use for that file by examining the file’s extension. If SET AUTOCOLOR has been used to specify a parser for that extension, KEDIT uses that parser to control syntax coloring for the file. The NULL parser, which does no syntax coloring, is used whenever a file has an extension for which no parser has been specified.

For example, the command

```
SET AUTOCOLOR .LNG LANG
```

tells KEDIT to use the LANG parser (which must already have been defined via the SET PARSER command) for files with an extension of .LNG.

Parsers referred to in SET AUTOCOLOR commands must already be defined, either by being built into KEDIT or via the SET PARSER command.

SET AUTOCOLOR is automatically put into effect for the following extensions during KEDIT initialization:

Extension	Parser
.BAS	BASIC
.FRM	BASIC
.C	C
.COB	COBOL
.COBOL	COBOL
.CBL	COBOL
.CPP	C
.CXX	C
.CS	CSHARP
.DLG	RESOURCE
.FOR	FORTRAN
.FORTRAN	FORTRAN
.F90	FORTRAN
.F	FORTRAN
.H	C
.HPP	C
.HXX	C
.HTM	HTML
.HTML	HTML
.INI	INI
.JAV	JAVA
.JAVA	JAVA
.KEX	REXX
.KML	REXX
.REX	REXX
.KLD	KLD
.PAS	PASCAL
.DPK	PASCAL
.DPR	PASCAL
.PRG	XBASE
.RC	RESOURCE

See also SET COLORING, SET ECOLOR, SET PARSER, Chapter 8, "KEDIT Language Definition Files"

SET AUTOEXIT

Format **`[Set] AUTOEXIT ON|OFF`**

KEDIT default: OFF

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET AUTOEXIT determines whether KEDIT automatically exits when the last file in the ring is closed.

With the default of AUTOEXIT OFF, KEDIT keeps running even if there are no files in the ring and all of its document windows have been closed. You can then begin editing other files, or you can use File Exit to close KEDIT's frame window and end your editing session.

With AUTOEXIT ON, your editing session ends whenever the last file is removed from the ring. Some KEDIT users prefer this behavior, because it is more like the behavior of text mode KEDIT and because it saves the extra step of closing KEDIT's frame window to exit KEDIT after closing the last file in the ring.

SET AUTOINDENT

Format **`[Set] AUTOIndent ON|OFF`**

KEDIT default: OFF

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET AUTOINDENT controls where the default F2 and (if INTERFACE CUA is in effect) Enter key definitions position the cursor after they add a line to your file.

With the default of AUTOINDENT OFF, the cursor is positioned in the left margin column of the newly-added line.

With AUTOINDENT ON, the cursor is lined up with the first character of the line above the new line. If that line is blank, the cursor stays in the column that it was in.

AUTOINDENT ON is useful when you are entering text in a programming language like C, where different portions of the file are indented to different levels, but where you most often want each line indented to the same level as the line above it.

SET AUTOSAVE

Format

`[Set] AUtosave n|OFF`

KEDIT default: OFF

Level: File

Dialog box: Options SET Command

Save Settings handling: Savable

Description

Use SET AUTOSAVE (''automatic save'') to cause KEDIT to automatically save your file to disk after a specified number of changes have been made to it.

After a power failure, a system crash, or inadvertent changes to your file during a KEDIT session, you may be able to use the autosaved version of your file to recover some of your work.

If AUTOSAVE is set to some number *n*, which must be greater than one, then whenever that many alterations have been made to your file (see SET ALT for a discussion of KEDIT's alteration count), KEDIT will automatically save your file to disk. While KEDIT is autosaving your file, you will see the message ''***Autosave***''. If no errors occur while saving the file, the first alteration count will be reset to zero. If errors occur (such as drive not ready or disk full errors), you will get an error message and the PC's speaker will beep.

When KEDIT writes a copy of your file to disk during an AUTOSAVE, it uses the same drive, directory, and name as the file you are editing, but it uses a file extension of .AUS (for ''AUtoSave''), as if you had issued the command ''SAVE =.AUS''. This causes the file to be autosaved under its own name, but with an extension of .AUS. The ''real'' version of your file on disk is not overwritten. KEDIT will not create a .BAK file during an autosave, regardless of the setting of BACKUP. Any existing .AUS file is simply overwritten when the autosave occurs.

Whenever you successfully write your file to disk by using the SAVE or FILE commands, or by using the File Save menu item, KEDIT erases the .AUS file to avoid cluttering up your disk. If you QQUIT your file, or your KEDIT session ends abnormally, the .AUS file will remain on your disk for possible recovery of lost work. You can periodically clean up your disk by erasing these .AUS files.

If you use the File Close or File Exit menu items to close a file that has been modified, the .AUS file is erased if you tell KEDIT to save the modified file; it is not erased if you tell KEDIT not to save the modified file.

The DIR.DIR and MACROS.KML files created by the DIR and MACROS commands are treated as special cases. Regardless of the setting of AUTOSAVE, they are never autosaved.

See also SET ALT, SET BACKUP

Examples

```
AUTOSAVE 30
```

The current file will be autosaved after every thirty alterations to a file with an extension of .AUS.

SET AUTOSCROLL

Format **`[Set] AUTOSCroll n|Half|OFF`**

KEDIT default: HALF

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description The AUTOSCROLL setting controls how far KEDIT will scroll when automatic horizontal scrolling is invoked.

Suppose, for example, that your window is 80 columns wide and VERIFY 1 80 is in effect (so that columns 1 through 80 of the file are displayed in the window) and that the cursor is in column 80. If you try to move the cursor 1 column to the right, KEDIT invokes automatic horizontal scrolling to bring column 81 of the file into the window. With AUTOSCROLL HALF (the default) in effect, KEDIT scrolls by half the width of the document window, which is in this case 40 columns. So columns 41 through 120 of the file are displayed in the window, with the cursor in column 81 of the file.

Note that with PREFIX ON, as discussed in User's Guide Chapter 7, "The Prefix Area", the cursor keys that normally invoke automatic horizontal scrolling instead move the cursor in and out of the prefix area.

If AUTOSCROLL *n* is in effect, KEDIT autoscrolls *n* columns at a time:

```
AUTOSCROLL 5
```

would cause KEDIT to autoscroll five columns at a time. In the above example, moving the cursor right from column 80 would cause columns 6 through 85 of your file to display, with the cursor at column 81 of the file.

With AUTOSCROLL OFF, KEDIT does not do automatic scrolling. If you attempt to move the cursor beyond the left or right edge of the document window, KEDIT places the cursor at the left or right edge of the document window and cannot move to the intended column of your file.

KEDIT autoscrolls by adding or subtracting the appropriate number of columns from the current VERSHIFT value, as if you had used the LEFT or RIGHT command to adjust the VERSHIFT value by that many columns.

See also LEFT, RIGHT, RGTLEFT, SET VERIFY

SET BACKUP

Format **`[Set] BACKup OFF|TEMP|KEEP`**

KEDIT default: OFF

Level: File

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET BACKUP controls whether KEDIT creates a backup copy of your file when it is written to disk. It affects the FILE and SAVE commands and the File Save menu item. It also has an effect when you use the File Close menu item with a modified file that you then tell KEDIT to save.

The default is BACKUP OFF, but unless you have very limited disk space you will probably want to run with BACKUP KEEP in effect, to create an automatic backup copy of files that you edit. You can periodically clean up your disk by erasing these .BAK files.

With BACKUP OFF, when a file that you are writing to disk will replace an existing file, KEDIT first erases the old version of the file and then writes out the new version. If something goes wrong while the file is being written (such as a disk error or power failure), you may end up with neither the old nor the new version of your file on the disk.

With BACKUP TEMP or BACKUP KEEP, when a file will replace an existing file, KEDIT first renames the existing file to have the same name but an extension of .BAK (for ‘‘BAcKup’’). (If this .BAK file already exists, KEDIT erases it.) KEDIT then

writes the new version of the file out to disk. If there is a problem while the new version is being written, you will at least still have the old version of the file on your disk (with the .BAK extension). If BACKUP TEMP is in effect, the .BAK file is erased after the new version of the file has been successfully written to disk. With BACKUP KEEP, the .BAK file is not erased, but is left on disk.

Note that if BACKUP is set to TEMP or KEEP, there must be enough room on your disk to hold both versions of your file: the new version that you are writing out and the old version, which is being kept as a .BAK file. If BACKUP TEMP is in effect, this .BAK file may need to be there for only a few seconds, while your new version is being written to disk. Nevertheless, there must still be enough room on your disk to hold both versions of your file or you will get disk full error messages.

See also SET AUTOSAVE

SET BEEP

Format **`[Set] BEEP ON|OFF`**

KEDIT default: OFF

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET BEEP controls whether the speaker beeps when error messages are displayed, when SOS ERRORBEEP is issued, and when ALERT commands are executed.

See also SET MOUSEBEEP, SOS BEEP, SOS ERRORBEEP, SOS MOUSEBEEP

SET BOUNDMARK

Format

```
[Set] BOUNDMark Zone|TRunc|MARgins|TABs|Verify|WINMARgin
[Set] BOUNDMark OFF
```

KEDIT default: ZONE TRUNC

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description

SET BOUNDMARK (“boundary mark”) controls KEDIT’s drawing of vertical lines in your document window that indicate important column locations in your file.

With the default of BOUNDMARK ZONE TRUNC, KEDIT draws boundary marks at the beginning and end of the current zone columns, and after the truncation column. For example, if ZONE 20 30 and TRUNC 80 are in effect, KEDIT will draw a line between columns 19 and 20 of your file, between columns 30 and 31 of your file, and between columns 80 and 81 of your file.

With SET BOUNDMARK, you can specify one or more of the following operands:

ZONE
Markers are drawn before the left zone column and after the right zone column.

TRUNC
A marker is drawn after the truncation column.

MARGINS
A marker is drawn before the left margin column and after the right margin column.

TABS
A marker is drawn before each tab column.

VERIFY
A marker is drawn after each set of verify columns. For example, if VERIFY 1 20 30 40 is in effect, KEDIT will display columns 1 through 20 of your file followed by columns 30 through 40, and boundary marks will appear between columns 20 and 30 and after column 40.

WINMARGIN
When WINMARGIN ON is in effect, a margin area (used for marking line blocks with the mouse) is displayed at the left of the document window. BOUNDMARK WINMARGIN tells KEDIT to draw a marker line at the right edge of this margin area, just before the first column of text.

Alternatively, you can specify BOUNDMARK OFF, which turns off any existing boundary markers. Note that a separate set of vertical lines, controlled by the SET COLMARK command, may still be displayed.

Notes

You would normally only use one or two of the BOUNDMARK operands at a time, since a large number of vertical lines could clutter your screen and be confusing.

The lines drawn by SET BOUNDMARK are automatically adjusted if one of the settings involved changes. For example, if BOUNDMARK ZONE is in effect and you issue a SET ZONE command, the position of the zone markers will be updated.

You can use a related command, SET COLMARK, to draw marker lines at specific columns of your file, independent of the settings of ZONE, TRUNC, etc.

The color of the boundary markers is determined by the foreground color that you specify for the SET COLOR BOUNDMARK; the background color for SET COLOR BOUNDMARK is ignored.

To avoid drawing an extra line where there is already a clearly understood boundary, boundary markers are not drawn for the left zone or left margin columns when these are set to column 1 of your file.

See also SET COLMARK, SET MARGINS, SET TABS, SET TRUNC, SET WINMARGIN, SET VERIFY, SET ZONE

Examples

```
BOUNDMARK WINMARGIN MARGINS
```

Draw boundary markers between the window margin area and the first column of the document window, before the left margin column, and after the right margin column.

```
BOUNDMARK OFF
```

Turn off any boundary markers being displayed for the current view of your file.

SET CASE

Format

```
[Set] CASE Mixed|Upper [Respect|Ignore] [Respect|Ignore]
```

KEDIT default: MIXED IGNORE RESPECT

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description The CASE option controls how KEDIT handles uppercase and lowercase text.

If CASE UPPER is in effect, all lowercase letters input directly into the document window are treated as if their uppercase equivalent had been entered. Note that only text entered directly into the document window is uppercased; CASE UPPER has no effect on text entered into dialog boxes.

For example, suppose that CASE UPPER is in effect and you try to type on the command line

```
INPUT abcABC123
```

The characters will actually appear on the command line in uppercase, and the command will cause ‘‘ABCABC123’’, not ‘‘abcABC123’’, to be input to your file. If CASE MIXED, the default, is in effect, lowercase letters are treated as lowercase letters, and not as their uppercase equivalents.

The first RESPECT or IGNORE setting affects how KEDIT searches for string targets and string column targets. If RESPECT is in effect, strings must match exactly. With

IGNORE, the default, strings containing exactly the same letters, regardless of whether they are both uppercase, both lowercase, or of different cases, will match. For example, with IGNORE,

```
/the/
```

would locate ‘‘the’’, ‘‘The’’, ‘‘THE’’, and several other variations of ‘‘the’’. With RESPECT, only the exactly matching ‘‘the’’ would be located.

IGNORE is useful since it allows you to search for words without worrying about whether they are or are not capitalized. A word may, for example, be capitalized in some of its occurrences where it appears as the first word of a sentence, but not in occurrences in the middle of sentences.

The first RESPECT or IGNORE setting also affects comparisons between uppercase and lowercase versions of the same letter during execution of the SORT command.

The second RESPECT or IGNORE controls comparisons made by the CHANGE, SCHANGE, and COUNT commands. With RESPECT, the default, the string specified on the left side of a CHANGE or SCHANGE, and the single string specified with the COUNT command, must match exactly for a change to occur; with IGNORE, the strings can differ in case.

If you ignore case when searching for strings in a change operation, KEDIT tries to determine the case of the result string placed back into your file in an intelligent manner. For example, with the second RESPECT|IGNORE operand set to IGNORE, the following command:

```
CHANGE /the/a/ 1 *
```

will change the line

```
The boy saw the girl.
```

into what you probably intended:

```
A boy saw a girl.
```

while blindly substituting lowercase ‘‘a’’ for each occurrence of ‘‘the’’ found by a case-insensitive search would have given

```
a boy saw a girl.
```

KEDIT’s internal rules for handling cases will very often, but not always, give the ‘‘right’’ result. However, if you want complete control of exactly which strings are matched and what they are changed into, you will need to set the second CASE RESPECT|IGNORE operand to RESPECT and specify the case of the strings involved exactly.

With INTERNATIONAL NOCASE, the default, in effect, KEDIT treats only the 26 letters from ‘‘A’’ to ‘‘Z’’ and ‘‘a’’ to ‘‘z’’ as alphabetic and does not treat accented characters as alphabetic. With INTERNATIONAL CASE in effect, KEDIT uses the

Windows language drivers installed on your system to determine which characters are considered alphabetic and how to uppercase, lowercase, and compare them.

See also SET INTERNATIONAL

Examples

```
CASE M R
```

This sets CASE to MIXED RESPECT, leaving the value of the second RESPECT|IGNORE setting unchanged.

```
CASE U
```

This sets CASE UPPER and leaves the RESPECT|IGNORE settings unchanged.

SET CLOCK

Format `[Set] CLOCK ON|OFF`

KEDIT default: ON

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description With CLOCK ON, the default, KEDIT displays a time-of-day clock on the status line. With CLOCK OFF, the clock is not displayed.

See also SET STATUSLINE

SET CMDLINE

Format `[Set] CMDline ON|OFF|Top|Bottom`

KEDIT default: BOTTOM

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET CMDLINE controls the display of KEDIT's command line.

[Set] CMDline Bottom
[Set] CMDline ON
The command line is displayed at the bottom of the editing window.

[Set] CMDline Top
The command line is displayed at the top of the editing window.

[Set] CMDline OFF
Turns off the display of KEDIT's command line. It is normally important to have the command line available, since many aspects of KEDIT are controlled from the command line. CMDLINE OFF is provided mainly for use in specialized macros where this is not an issue. CMDLINE OFF is not recommended for general use.

SET COLMARK

Format

[Set] COLMark *n1* [*n2 n3 ...*]
[Set] COLMARK OFF

KEDIT default: OFF

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description

SET COLMARK ("column mark") controls KEDIT's drawing of vertical lines that indicate column locations in your file.

With COLMARK OFF, the default, no column markers are drawn. Note that a separate set of vertical lines, controlled by the SET BOUNDMARK command, may still be displayed.

But if you give a list of one or more columns, KEDIT will draw a vertical line to the left of where each specified column of your file appears in the document window. The color of the vertical lines is determined by the foreground color of the SET COLOR COLMARK setting; any background color specified for COLOR COLMARK is ignored. You can specify a maximum of 20 columns.

SET COLMARK draws lines at fixed column positions in your file. A related command, SET BOUNDMARK, lets you draw lines whose locations vary depending on the boundaries of your zone columns, your margin columns, etc.

See also

SET BOUNDMARK

Examples

```
COLMARK 81
```

This tells KEDIT to draw a vertical line preceding column 81 of each line of your file. This can be very useful, since on many displays you can have windows much wider than 80 columns and you can mistakenly enter more text on a line than you intend to.

```
COLMARK 10 20 30 40
```

This gives you vertical lines preceding columns 10, 20, 30, and 40 of your file.

SET COLOR

Format

[Set] COLOR *field foreground* [ON *background*]
[Set] COLOR *field* DEFAULT

KEDIT default: See the table below
Level: File
Dialog box: Options SET Command
Save Settings handling: Savable

Description

SET COLOR allows you to control the colors and highlighting that KEDIT uses on your display. You can individually control the color used by KEDIT for each of the different types of information displayed by KEDIT by specifying the type of *field* involved and the *color* to use for it.

Color Defaults

Here is a list of the fields whose colors you can control, along with the color KEDIT uses by default for each field, and a description of each field:

Field	Default	Description
Arrow	blue on white	command line arrow
Block	white on black	text within marked block
BOUNDMark	gray on white	ZONE markers, etc. controlled by SET BOUNDMARK
Cmdline	black on white	command line
COLMark	gray on white	column markers controlled by SET COLMARK
CURRBox	gray on white	box drawn around current line
Filearea	black on white	file area
HIghlight	black on yellow	lines highlighted by SET HIGHLIGHT
Idline	blue on white	ID line
Msgline	red on white	message lines
Pending	dark red on white	pending prefix commands

Field	Default	Description
PRefix	dark blue on white	prefix area
Scale	dark blue on white	scale line
SHadow	dark blue on white	shadow lines
Tabline	dark blue on white	tab line
THIghlight	black on green	highlighted target
TOfeof	dark blue on white	top-of-file and end-of-file lines
TOOLtip	black on yellow	pop-up toolbar help (no longer has any effect - system tooltip color is used instead)

Available colors

You can choose from the following sixteen colors, although not all will display properly on all adapters. For example, on a gray scale display, the colors would map into shades of gray.

BLAck
White
BLUe
DARK BLUe
Cyan
DARK Cyan
GRAy
DARK GRay
Green
DARK Green
Magenta
DARK Magenta
Red
DARK Red
Yellow
DARK Yellow

Specifying colors

The color to use for a field can be specified in two ways:

`[Set] COLOR` ***`field foreground`*** **`[ON`** ***`background`*****`]`**

Specify the name of the field involved, followed by the foreground color for the field and, optionally, the background color for the field. If you do not specify a background color, a background of white is used. You can use an asterisk (“*”) for the field name to indicate that you want to set all fields to the same color.

[Set] COLOR *field* DEFAULT

If you have made changes to the color of a field, you can tell KEDIT for Windows to switch back to using its default color for the field. A table of KEDIT for Windows' default colors is given above.

Notes

SET COLOR is only used to control colors used within a document window. It is not used for scrollbars, dialog boxes, window title bars and borders, etc. These colors are controlled via the Windows Control Panel.

For BOUNDMARK, COLMARK, and CURRBOX, KEDIT draws lines on the screen using the foreground color involved, and the background color specified for these items has no effect.

When COLORING ON is in effect and you are using KEDIT's syntax coloring facility to show keywords, comments, etc. in your text in different colors, SET ECOLOR determines the colors involved.

If MONITOR COLOR and MONITOR MONO, settings supplied for compatibility with text mode KEDIT, are in effect, SET COLOR works as it does in text mode KEDIT, and not as described here.

See also SET ECOLOR, SET MONITOR, SET PCOLOR

Examples

SET COLOR FILEAREA BLUE ON YELLOW

The file area of the document window will be displayed as blue characters on a yellow background.

COLOR * DEFAULT

All fields are reset to their default colors.

SET COLORING

Format **[Set] COLORING ON|OFF AUTO|*parser***

KEDIT default: ON AUTO

Level: File

Dialog box: Options SET Command

Save Settings handling: Not savable

Description Use SET COLORING to enable or disable KEDIT's syntax coloring facility and to determine which parser KEDIT will use to handle the syntax coloring.

When syntax coloring is active, KEDIT uses different colors to highlight different types of text. Syntax coloring is controlled by a language-specific parser. The parser scans the text in your file, decides which characters are parts of keywords, comments, strings, etc., and displays the text in the appropriate color; the specific colors used are determined by the SET ECOLOR command.

Syntax coloring parsers for several languages are built into KEDIT, and you can use KEDIT Language Definition files in connection with the SET PARSER command to load your own parser definitions.

The first operand to SET COLORING turns syntax coloring ON or OFF for the current file. When COLORING OFF is in effect, your file is displayed without syntax coloring. When COLORING ON is in effect, your file is displayed using the colors determined by the parser that you specify.

The second operand to SET COLORING determines the parser to use for the current file:

AUTO

With the AUTO operand, KEDIT uses a parser that is determined by the extension of the file you are editing. The parser that is used for a given extension is controlled via the SET AUTOCOLOR command. By default, the following extensions are handled:

Extension	Parser
.BAS	BASIC
.FRM	BASIC
.C	C
.COB	COBOL
.COBOL	COBOL
.CBL	COBOL
.CPP	C
.CS	CSHARP
.CXX	C
.DLG	RESOURCE
.FOR	FORTRAN
.FORTRAN	FORTRAN
.F90	FORTRAN
.F	FORTRAN
.H	C
.HPP	C
.HXX	C

Extension	Parser
.HTM	HTML
.HTML	HTML
.INI	INI
.JAV	JAVA
.JAVA	JAVA
.KEX	REXX
.KML	REXX
.REX	REXX
.KLD	KLD
.PAS	PASCAL
.DPK	PASCAL
.DPR	PASCAL
.PRG	XBASE
.RC	RESOURCE

If no parser is defined for a particular extension, KEDIT uses the NULL parser, which is a special dummy parser that doesn't actually apply any syntax coloring.

parser

The *parser* operand gives the name of the language-specific parser to use. You can choose one of the parsers that is built into KEDIT or you can use a parser of your own that you have loaded via the SET PARSER command. Here are the parsers that are built into KEDIT:

Parser	Used With
C	C and C++ programs
CSHARP	C# programs
REXX	KEXX and REXX programs
HTML	HTML documents
JAVA	Java programs
COBOL	COBOL programs
FORTRAN	FORTRAN programs
PASCAL	Pascal and Delphi programs
KLD	the KEDIT Language Definition files described in Chapter 8, "KEDIT Language Definition Files"
INI	INI files
BASIC	BASIC programs

Parser	Used With
XBASE	xBase programs
RESOURCE	the RC and DLG files used in Windows program development
NULL	dummy parser that doesn't actually apply syntax coloring

Notes

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 all normal situations correctly, but accepting that in some unusual cases, especially in files that contain syntax errors, text may be colored incorrectly.

Most KEDIT users will be able to leave the default of COLORING ON AUTO unchanged. Files with the extensions used in several common languages will automatically get syntax coloring, and other files will not.

See also

SET AUTOCOLOR, SET ECOLOR, SET PCOLOR, SET PARSER, Chapter 8, "KEDIT Language Definition Files"

SET CURLINE

Format

`[Set] CURLine` *`line`*

KEDIT default: M (Middle of the document window)

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description

Use SET CURLINE to control the *line* of the document window in which KEDIT displays the current line.

You can specify the *line* in three ways: relative to the top of the document window, relative to the middle of the document window, and relative to the bottom of the document window.

`[Set] CURLine` *`n|+n`*
This tells KEDIT to use line *n* of the document window.

[Set] CURLine M
This tells KEDIT to use the middle line of the document window. In a document window that is 24 lines high, KEDIT would use line 12. (In a document window that is 25 lines high, KEDIT would round up and use line 13.)

[Set] CURLine M+*n*|M-*n*
This tells KEDIT to use the line *n* lines above or below the middle line of the document window.

[Set] CURLine -*n*
This tells KEDIT to use the line *n* lines from the bottom of the document window, where the last line of the document window is line -1.

See also SET CURRBOX

Examples

```
CURLINE 2
```

Display the current line in the second line of the document window.

```
CURLINE -2
```

Display the current line in the second line from the bottom of the document window.

```
CURLINE M+2
```

Display the current line in the line two lines below the middle of the window.

SET CURRBOX

Format **[Set] CURRBox ON|OFF [ON|OFF]**

KEDIT default: ON OFF

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET CURRBOX determines whether KEDIT draws a box around the current line to make that line stand out.

The first operand determines whether the box is drawn when the cursor is on the command line. This is ON by default, since most commands issued from the command line act relative to the current line, and it is useful to be sure which line this is.

The second operand determines whether the box is drawn when the cursor is in the file area. This is OFF by default, because the current line is usually of less interest when you are not working from the command line.

The box's color is determined by the foreground color of the SET COLOR CURRBOX setting; any background color specified for COLOR CURRBOX is ignored.

SET CURSORSIZE

Format **`[Set] CURSORSIze vovr vins hovr hins`**

KEDIT default: 10 25 15 30

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET CURSORSIZE controls the size of KEDIT's text cursor.

If CURSORTYPE VERTICAL is in effect, or if CURSORTYPE INTERFACE and INTERFACE CUA are in effect, KEDIT uses a vertical cursor. The first and second operands for SET CURSORSIZE control the width of this vertical cursor: the first operand controls the width of the cursor in Overtype Mode, and the second operand controls the width in Insert Mode.

If CURSORTYPE HORIZONTAL is in effect, or if CURSORTYPE INTERFACE and INTERFACE CLASSIC are in effect, KEDIT uses a horizontal cursor. The third and fourth operands for SET CURSORSIZE control the height of this horizontal cursor: the third operand controls the height of the cursor in Overtype Mode, and the fourth operand controls the height in Insert Mode.

The values used with SET CURSORSIZE are expressed as a percentage of the width of a character, and can range from 1 to 100. For example, with the default settings in effect, a vertical cursor in Overtype Mode is 10% of the width of a character in the current font.

Examples

```
SET CURSORSIZE 20 40 20 40
```

This example makes the cursor sizes somewhat thicker than the default settings. For example, the vertical cursor is 20% of the character width in Overtype Mode instead of the default of 10%.

```
SET CURSORSIZE 25 10 30 15
```

This example makes the Insert Mode cursor thinner than the Overtype Mode cursor, reversing KEDIT's normal convention of using a thicker cursor to indicate Insert Mode.

SET CURSORTYPE

Format **`[Set] CURSORType Vertical|Horizontal|Interface`**

KEDIT default: INTERFACE

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET CURSORTYPE determines the shape of KEDIT's text cursor.

With CURSORTYPE VERTICAL, KEDIT uses a vertical cursor, displayed at the left of the current character.

With CURSORTYPE HORIZONTAL, KEDIT uses a horizontal cursor, displayed beneath the current character.

With the default of CURSORTYPE INTERFACE, the shape of KEDIT's cursor is dependent on the setting of the INTERFACE option. With INTERFACE CUA, KEDIT will use a vertical cursor; with INTERFACE CLASSIC, KEDIT will use a horizontal cursor.

Most Windows programs use a vertical text cursor, as KEDIT for Windows does by default when INTERFACE CUA is in effect. This fits well with the way KEDIT marks blocks when INTERFACE CUA is in effect. With INTERFACE CUA, blocks marked with the mouse and with CUA-compatible keys extend up to, but do not include, the current character. Visually, this means that they extend up to, but not beyond, the vertical cursor displayed at the left of the current character.

Text mode KEDIT and other text mode applications use a horizontal text cursor, as KEDIT for Windows does when INTERFACE CLASSIC is in effect. This also fits well with the way KEDIT marks blocks when INTERFACE CLASSIC is in effect. With INTERFACE CLASSIC, stream and box blocks always include the character at the cursor position. Visually, the block extends to the edge of the horizontal cursor displayed under the current character.

SET DEBUGGING

Format **`[Set] DEBUGGing ON|OFF`** ***`height tracing`***

KEDIT default: OFF 15 +R

Level: Global

Dialog box: Options SET Command

Save Settings handling: Not savable

Description

SET DEBUGGING controls whether the debugging window (a separate window used by the macro debugger) is on or off, the height of the debugging window, and the default tracing level for macros executed via the DEBUG command.

When the debugging window is on, trace output is written to the debugging window and interactive trace input is read from a line at the bottom of the debugging window. When the debugging window is off, the KEXX debugger is inactive and all KEXX TRACE instructions are ignored.

The *height* of the debugging window is initially set to 15 lines. The debugging window must be at least 6 lines high and can be no more than 25 lines high. *Tracing* controls the initial level of tracing in effect for macros run via the DEBUG command. The default value of +R specifies that interactive trace will be active and that all clauses and expression results will be traced.

See also

User's Guide Chapter 10, "Using Macros", PROFDEBUG initialization option, DEBUG, KEXX TRACE instruction

Examples

```
DEBUGGING ON 10 +I
```

Turns on the debugging window, and sets it to occupy ten lines. Also sets the default level of tracing for macros run via the DEBUG command to +I, which means interactive tracing of all clauses and intermediate and final expression results.

SET DEFEXT

Format

```
[Set] DEFEXT ON|OFF
```

KEDIT default: OFF

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description

With DEFEXT (''default extension'') ON, the KEDIT, GET, FILE, PUT, and SAVE commands use the extension of the current fileid if you do not explicitly specify one. For example, assume you are editing the file SAMPLE1.PAS and you want to save it under the name SAMPLE2.PAS. With DEFEXT OFF, you must say either

```
SAVE SAMPLE2.PAS
```

or

```
SAVE SAMPLE2.=
```

With DEFEXT ON, you can simply say

```
SAVE SAMPLE2
```

and the file extension of .PAS will be assumed.

With DEFEXT ON, if you want to specify a fileid that has no extension, you must give a period after the name of the file. So, with DEFEXT ON,

```
SAVE SAMPLE2
```

saves the file as SAMPLE2.PAS, and

```
SAVE SAMPLE2.
```

saves the file as SAMPLE2, with no extension.

Note that SET DEFEXT does not affect how file names are handled when you use the File Open or File Save As dialog boxes, where you must explicitly supply any desired extension.

See also NODEFEXT initialization option

SET DEFPROFILE

Format **[Set] DEFPROFile *fileid***

KEDIT default: WINPROF.KEX

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET DEFPROFILE (“default profile”) lets you change the name of the profile that KEDIT runs by default to something other than WINPROF.KEX.

When KEDIT needs to run your profile, because the first file is being added to the ring at the start of a KEDIT session or because REPROFILE ON is in effect and new files are being added to the ring later in a KEDIT session, it normally runs the profile specified via the DEFPROFILE option. which defaults to WINPROF.KEX.

You can override this behavior by using the NOPROFILE option on the KEDIT command line, in which case no profile is executed, or by using the PROFILE option to specify a different profile to be executed.

When you change the value of DEFPROFILE, the change stays in effect for the rest of the KEDIT session and, if you use Options Save Settings, will affect future KEDIT

sessions. In contrast, the PROFILE and NOPROFILE options only affect the profile executed while processing the current KEDIT command line, and do not affect the profile executed for additional files that you later begin editing.

You can also specify DEFPROFILE as a KEDIT initialization option on the command line used to invoke KEDIT. If you do this, the value that you specify will replace the current value of the DEFPROFILE option and, at the start of a KEDIT session, will override any DEFPROFILE option saved in the Windows registry previous sessions.

See also DEFPROFILE initialization option, PROFILE initialization option, NOPROFILE initialization option, SET REPROFILE

Examples

```
SET DEFPROFILE C:\MYMACROS\MYPROF.KEX
```

KEDIT will use C:\MYMACROS\MYPROF.KEX as your default profile, instead of WINPROF.KEX.

SET DEFSORT

Format

```
[Set] DEFSORT Date|Extension|Name|Path|Size ...
[Set] DEFSORT OFF
```

KEDIT default: NAME EXTENSION PATH

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET DEFSORT (''default sort'') controls the order in which the DIR and DIRAPPEND commands normally sort the list of files they create.

[Set] DEFSORT Date
Causes DIR and DIRAPPEND to sort the list of files according to the date and time of each file, with the newest files listed first.

[Set] DEFSORT Extension
Tells DIR and DIRAPPEND to sort the list of files in alphabetical order according to the file extension.

[Set] DEFSORT Name
Sorts the list of files in alphabetical order according to the file name.

[Set] DEFSORT Path
Sorts the list of files in alphabetical order according to the drive letter and subdirectory of each file.

[Set] DEFSORT Size
Sorts the list of files according to the size of each file, with the largest files listed first.

These operands can be combined. For example, the default setting of DEFSORT, which is NAME EXTENSION PATH, tells DIR and DIRAPPEND to sort directory listings by name and, if several files have the same name, to sort these by extension and then path.

With DEFSORT OFF, DIR and DIRAPPEND do not sort directory listings; files are listed in the order they appear in your disk's directory.

After using DIR or DIRAPPEND to create a DIR.DIR file sorted according to your DEFSORT setting, you can use the DIRSORT command to re-sort the DIR.DIR file into a different order.

See also DIR, DIRAPPEND, DIRSORT, SET DIRFORMAT

SET DIRFORMAT

Format **[Set] DIRFORMat *fname fext year***

KEDIT default: 30 10 2

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET DIRFORMAT controls the output format used by KEDIT's DIR command.

The first two operands control the amount of space set aside in DIR.DIR files for file names and for file extensions. To accommodate long filenames, you can use the SET DIRFORMAT command to have KEDIT set aside more columns for file names and for file extensions (that is, everything after the last period in a fileid). By default, KEDIT sets aside 30 columns in DIR.DIR files for filenames and 10 columns for file extensions.

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.

The third operand of SET DIRFORMAT controls the number of digits used to display the year in DIR.DIR files. It can be set to either 2 (the default, which yields two-digit years, such as 96 or 08, in DIR.DIR listings) or 4 (which yields four-digit years, such as 1996 or 2008).

SET DIRFORMAT does not have any effect on a DIR.DIR file that you have already created; it only affects subsequent DIR commands. So you should use SET DIRFORMAT before issuing a DIR command whose output you want to affect.

See also DIR, DIRAPPEND

SET DISPLAY

Format **[Set] DISPlay *n1* [*n2*|*]**

KEDIT default: 0 0

Level: View

Dialog box: Options SET Command

Save Settings handling: Not savable

Description SET DISPLAY causes KEDIT to select for display lines whose selection level falls into the range *n1* through *n2*. Lines whose selection level falls outside this range are excluded from the display.

Each line of your file has a number, called its selection level, associated with it. Selection levels can range from 0 to 255. You can set the selection level of a line by using the SET SELECT command. The ALL command, the X and S prefix commands, and the Edit Selective Editing dialog box also affect selection levels of lines in your file.

Initially, all lines in your file have a selection level of 0, and DISPLAY is set to 0 0, so all lines in your file are selected.

SET DISPLAY *n1* *

is equivalent to

SET DISPLAY *n1* 255

and

SET DISPLAY *n1*

is equivalent to

SET DISPLAY *n1 n1*

With SHADOW ON, the default, excluded lines are represented in your document window by a shadow line, which indicates the number of excluded lines. With SHADOW OFF, excluded lines are not represented in your document window at all.

With SCOPE DISPLAY, the default, most KEDIT commands will operate only on lines that are selected, and will act as if excluded lines are not present in your file. With

SCOPE ALL, KEDIT commands operate on excluded lines as well as selected lines, even though excluded lines are not shown on your screen. A special case is the current line, which is always displayed with SCOPE ALL, regardless of its selection level.

See also User's Guide Chapter 8, "Selective Line Editing and Highlighting", ALL, SET SCOPE, SET SELECT, SET SHADOW

Examples

```
SET DISPLAY 5 10
```

All lines in your file with a selection level of 5 through 10 are shown, while all lines in your file whose selection level is less than 5 or greater than 10 are excluded.

```
SET DISPLAY 1 *
```

All lines in your file whose selection level is greater than zero are selected.

```
SET DISPLAY 1
```

This is equivalent to SET DISPLAY 1 1—only lines with a selection level of 1 are selected.

SET DOCSIZING

Format **`[Set] DOCSIZing Standard|EXTENded [`*`n`*`]`**

KEDIT default: EXTENDED 80

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET DOCSIZING controls the method used to determine the size of new non-maximized document windows, and of existing document windows that are rearranged via the Window Cascade menu item.

When DOCSIZING STANDARD is in effect, KEDIT lets window handling code built into Windows determine the size. Each window is given a standard default size that is some fixed percentage of the size of the frame window. With DOCSIZING STANDARD, SET DOCSIZING's second operand has no effect.

When DOCSIZING EXTENDED is in effect, the window sizes are controlled by KEDIT. The idea is that rather than making all windows the same size, KEDIT tries to make each window large enough to hold as much useful information as possible. KEDIT makes each new document window extend to the bottom of the frame window, and tries to make each window wide enough to display *n* columns of text, where *n* defaults to 80 and is the second operand specified for SET DOCSIZING.

DOCSIZING EXTENDED makes a difference only when the frame window is at least wide enough for the width of one of the document windows that you will create or cascade, plus enough extra room to allow four cascaded document windows, each slightly offset from the others, to fit. In a typical case, you will have document windows 80 characters wide and the extra room to allow for cascading will be about 10 characters wide, so the frame window needs to be wide enough for approximately 90 columns of text. If the frame window is narrower than this, KEDIT uses the standard document window sizing, as if DOCSIZING STANDARD had been in effect.

Note that for DOCSIZING EXTENDED to have an effect on the initial document window created at the start of a KEDIT session, it must have been set during a previous KEDIT session and saved to the Windows registry via Options Save Settings.

SET DRAG

Format

```
[Set] DRAG Box|Line|Stream [PERSISTent|SELection]
      [Anchor|Word] [RESET]
[Set] DRAG CMDline [SELection] [Anchor|Word] [RESET]
[Set] DRAG DRAGDROP
[Set] DRAG NONE
```

KEDIT default: NONE

Level: Global

Dialog box: None

Save Settings handling: Not savable

Description

SET DRAG, a specialized command used only in the macros that process mouse clicks, controls what will happen when you drag the mouse after clicking a mouse button.

[Set] DRAG Box|Line|Stream
[Set] DRAG CMDline

To mark a block when you drag the mouse, KEDIT's internal mouse handling routines repeatedly issue MARK commands. This form of SET DRAG determines the operands used for these MARK commands. You can mark a BOX, LINE, or STREAM block or, if INTERFACE CUA is in effect, you can mark a command line selection.

PERSISTent|SELection

Determines whether dragging the mouse marks persistent blocks or selections. For LINE, STREAM, and BOX blocks, PERSISTENT is the default and SELECTION may only be specified when INTERFACE CUA is in effect. For command line selections, which are available only when INTERFACE CUA is in effect, SELECTION is the default and is the only legal choice.

Anchor|Word
Determines whether the ANCHOR operand or the WORD operand (which is valid only for stream blocks and command line selections) is used in the MARK commands issued while dragging to mark a block. All blocks marked by dragging the mouse are anchored blocks; ANCHOR is the default for SET DRAG if neither ANCHOR nor WORD is specified explicitly. See the MARK command for discussion of these operands.

RESET
If RESET is specified, the first MARK command issued when the mouse is dragged will include the RESET operand, unmarking any existing block. Otherwise, any existing block in the current file will be extended from its anchor point as you drag with the mouse.

[Set] DRAG DRAGDROP
This form of SET DRAG specifies that dragging the mouse should invoke a drag-and-drop operation, either moving or copying the currently-marked block to the location at which the mouse button is released. If the Ctrl key is down at the end of the drag operation, the block is copied; otherwise it is moved.

[Set] DRAG NONE
Specifies that no block mark, move, or copy will occur when the mouse is dragged.

See also EXTEND, MARK

SET ECOLOR

Format

[Set] ECOLOR *a foreground* [ON *background*]
[Set] ECOLOR *a* DEFAULT

KEDIT default: See the table below

Level: File

Dialog box: Options SET Command

Save Settings handling: Savable

Description

SET ECOLOR (“emphasis color”) controls the colors used by KEDIT’s syntax coloring facility.

When syntax coloring, enabled via the SET COLORING command, is active, KEDIT uses different colors to highlight different types of text. KEDIT includes a simple parser for each language with syntax coloring support. The parser scans the text in your file, decides which characters are parts of keywords, comments, strings, etc. and based on this decides which of 35 emphasis types, referred to by the letters A through Z and the numbers 1 through 9, to use for those characters. For example, numbers used in most programming languages are given emphasis type C, which is by default shown in dark red.

Here are the emphasis colors that KEDIT uses by default when MONITOR WINDOWS is in effect:

Letter	Color	Language Element
A	dark green	comments
B	dark cyan	strings
C	dark red	numbers
D	blue	keywords
E	dark red	labels
F	dark red	preprocessor keywords
G	red	header lines
H	black	extra right paren, matchable keyword
I	blue	level 1 paren
J	blue	level 1 matchable keywords
K	dark red	level 1 matchable preprocessor keywords
L	dark green	level 2 paren, matchable keyword
M	red	level 3 paren, matchable keyword
N	dark cyan	level 4 paren, matchable keyword
O	dark magenta	level 5 paren, matchable keyword
P	gray	level 6 paren, matchable keyword
Q	dark blue	level 7 paren, matchable keyword
R	magenta	level 8 or higher paren, matchable keyword
S	magenta	incomplete strings
T	blue	HTML markup tags
U	red	HTML character/entity references
V—Z	black	not currently used
1	red	alternate keyword color 1
2	dark blue	alternate keyword color 2
3	dark red	alternate keyword color 3
4	dark magenta	alternate keyword color 4
5	dark green	alternate keyword color 5
6	dark cyan	alternate keyword color 6
7	red	alternate keyword color 7
8	black	alternate keyword color 8
9	blue	alternate keyword color 9

[Set] ECOLOR *a foreground* [ON *background*]

Specify the emphasis type involved (in the range A—Z or 1—9), followed by the foreground color to use and, optionally, the background color. If you do not specify a background color, a background of white is used. You can use an asterisk (''*'') instead of a letter as the emphasis type to indicate that you want all emphasis types to use the same color. The foreground and background colors that you can choose from are the same as the colors used with the SET COLOR command.

[Set] ECOLOR *a* DEFAULT

If you have made changes to an emphasis color, you can tell KEDIT for Windows to switch back to using the default color. A table of KEDIT for Windows' default emphasis colors is given above.

Notes

Even when syntax coloring is active, some of the text in your file may not be given an emphasis color. For example, most parsers do not give an emphasis color to ordinary variables. KEDIT uses the color controlled by SET COLOR FILEAREA, which is normally black on white, for such text.

For text that is part of a block or part of a highlighted target, syntax coloring is overridden. KEDIT instead uses the colors determined by SET COLOR BLOCK (normally white on black) or SET COLOR THIGHLIGHT (normally black on green).

For text that is highlighted because you have used the SET HIGHLIGHT command or the TAG command, and for which syntax coloring also applies, KEDIT attempts to merge the two colors. KEDIT uses the background color from the SET COLOR HIGHLIGHT setting (this is normally yellow), and the foreground color from the appropriate ECOLOR setting. You need to take this into account when you decide on the colors you will use with SET COLOR HIGHLIGHT and SET ECOLOR, since not all combinations of foreground and background colors work well together.

See also

SET AUTOCOLOR, SET COLOR, SET COLORING, SET PARSER, SET PCOLOR, Chapter 8, "KEDIT Language Definition Files"

Examples

```
ECOLOR A RED
```

Emphasis type A (used for comments) is displayed in red.

```
ECOLOR * DEFAULT
```

All emphasis colors are reset to their default values.

SET EOFIN

Format **`[Set] EOFIN ALLOW|PREVENT`**

KEDIT default: ALLOW

Level: File

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET EOFIN controls how KEDIT handles end-of-file characters (character code 26) when reading in files as you begin to edit them and during execution of the GET command.

End-of-file characters are sometimes used to signal that the end of all useful data in a file has been reached. Many programs (such as the DOS TYPE command) will stop processing a file when an end-of-file character is encountered. With EOFIN PREVENT, KEDIT also behaves this way: when an end-of-file character is encountered, KEDIT assumes that it has reached the end of the useful data in the file and stops processing it.

With EOFIN ALLOW, the default, KEDIT does not stop processing a file when it encounters an end-of-file character, but instead processes the entire file and allows end-of-file characters as data in your file.

Existing files sometimes have one or more end-of-file characters at the very end of the file, because some older programs require it. So KEDIT always ignores any end-of-file characters at the very end of an input file, regardless of the setting of EOFIN.

See also User's Guide Chapter 12, "File Processing", SET EOFOUT, SET EOLIN

SET EOFOUT

Format **`[Set] EOFOUT EOL|EOLEOF|EOF|NONE`**

KEDIT default: EOL

Level: File

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET EOFOUT controls the sequence of characters that KEDIT adds to the last line of files written to disk by the FILE, SAVE, and PUT commands, by the autosave facility, and by the File Save and related menu items.

[Set] EOFOUT EOL
Tells KEDIT to add the same end-of-line sequence, controlled by SET EOLOUT and normally a carriage return-linefeed pair, to the last line of the file as it adds to every other line of the file. This is the default.

[Set] EOFOUT EOLEOF
Causes KEDIT to add the end-of-line sequence defined by SET EOLOUT followed by an end-of-file character (character code 26) to the last line of a file. This was the default behavior in text mode KEDIT 4.0 and earlier.

[Set] EOFOUT EOF
Tells KEDIT to add only an end-of-file character, with no end-of-line sequence, to the last line of the file.

[Set] EOFOUT NONE
Tells KEDIT not to add anything to the last line of the file.

See also User's Guide Chapter 12, "File Processing", SET EOFIN, SET EOLOUT

SET EOLIN

Format **[Set] EOLIN CRORLF|LF|CR|NONE**

KEDIT default: CRORLF

Level: File

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET EOLIN controls how KEDIT determines where each line ends when reading files as you begin to edit them and during execution of the GET command.

Most PC files use a carriage return-linefeed sequence (character codes 13 and 10) to mark the end of each line. Some files, especially those created under UNIX, have only a linefeed at the end of each line, and a few files have only a carriage return.

[Set] EOLIN CRORLF
With EOLIN CRORLF (''CR or LF''), the default, carriage returns, linefeeds, and carriage return-linefeed pairs all signal the end of a line to KEDIT. Most text files are handled correctly with EOLIN CRORLF.

[Set] EOLIN CR
Carriage returns and carriage return-linefeed pairs signal the end of a line, but linefeed characters do not. This allows you to edit files that use linefeed characters as data within a line, instead of as an end-of-line signal.

[Set] EOLIN LF
Linefeeds and carriage return-linefeed pairs signal the end of a line, but carriage return characters do not, so you can edit files that use carriage return characters as data.

[Set] EOLIN NONE
KEDIT does not look for an explicit end-of-line sequence. Instead, KEDIT assumes that all lines have a fixed length, equal to the value of the LRECL setting, and KEDIT reads in exactly that many bytes per line. EOLIN NONE is useful in specialized situations and is discussed further in User's Guide Chapter 12, "File Processing".

See also User's Guide Chapter 12, "File Processing", SET EOFIN, SET EOLOUT

SET EOLOUT

Format **[Set] EOLOUT CRLF|LF|CR|NONE**

KEDIT default: CRLF

Level: File

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET EOLOUT controls which characters KEDIT uses to mark the end of each line written to disk by the FILE, SAVE, and PUT commands, by the autosave facility, and by the File Save and related menu items.

[Set] EOLOUT CRLF
This is the default and causes KEDIT to use a carriage return-linefeed (ASCII codes 13 and 10) to mark the end of each line. This is the standard end-of-line sequence for text files on PC systems.

[Set] EOLOUT LF
KEDIT adds a linefeed character to each line. This is the standard way of ending lines on UNIX systems.

[Set] EOLOUT CR
KEDIT adds a carriage return character to each line.

[Set] EOLOUT NONE
Tells KEDIT not to write any explicit end-of-line sequence after each line. This is useful in specialized situations and is discussed in User's Guide Chapter 12, "File Processing".

Depending on the value of EOFOUT, a special end-of-line sequence may be written after the last line of the file; see SET EOFOUT for a discussion of the options involved.

See also User's Guide Chapter 12, "File Processing", SET EOFOUT, SET EOLIN

SET FCASE

Format

```
[Set] FCASE ASIS|LOWER
```

KEDIT default: ASIS

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description 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.

SET FILEID, FMODE, FPATH, FNAME, FEXT, FTYPE

Format

```
[Set] FILEId d:path\name.ext
[Set] FMode d[:]
[Set] FPath path
[Set] FName name
[Set] FExt ext
[Set] FType ext
```

KEDIT default: Based on fileid used to begin editing the file

Level: File

Dialog box: Options SET Command (SET FILEID only)

Save Settings handling: Not savable

Description

SET FILEID allows you to change the fileid of the file you are currently editing. The fileid is displayed in the title bar of the document window and, unless you override it, is the fileid under which your file will eventually be written to disk by the FILE or SAVE commands, or by menu items like File Save.

Fileids are made up of four components: the drive specifier, the directory path, the file name, and the file extension. SET FILEID lets you control all of these. Related commands let you control the individual components: SET FMODE sets the drive specifier, SET FPATH sets the directory path, SET FNAME sets the file name, and SET FEXT and the equivalent SET FTYPE set the file extension.

If you omit the drive specifier from the operand for SET FILEID, KEDIT uses the drive letter of the current fileid. If you omit the directory path, the directory path of the current fileid will be used, unless the drive specifier is changing, in which case KEDIT uses the current directory of the specified drive.

Shortcuts

For the drive specifier, you can give a specific drive letter or you can use a period (“.”) to indicate that the current default system drive letter should be used.

For the directory path, you can give a specific directory path, or you can use the same types of relative path specifications as are accepted in Command Prompt windows.

For the file name, give the name to use. You cannot omit the file name from the fileid specification.

For the file extension, you can give a specific file extension. If you omit the file extension, what happens depends on the setting of DEFEXT. If DEFEXT OFF is in effect, the new fileid will have no extension. If DEFEXT ON is in effect, and the new fileid specification does not end in a period, the current file extension will be used. If DEFEXT ON is in effect but the new fileid specification does end in a period, the new fileid will have no extension. (FEXT NONE or FTYPE NONE is handled as a special case, providing another way to assign a fileid with no extension.)

You can also use an equal sign (“=”) for any of the four components of the fileid to specify that the corresponding component of the existing fileid should be used without change.

You can optionally enclose fileid operands in double quotes, and you must do so for fileids that contain blanks.

UNC names

You can also specify fileids using UNC (Universal Naming Convention) names. This is 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.

Examples

Each of the following examples assumes that the current fileid is C:\PROJ\PROG.TXT, and that the current drive and directory is D:\WORK.

```
FILEID DEF.XYZ
```

Changes the fileid to C:\PROJ\DEF.XYZ.

```
FILEID SUB\DEF.XYZ
```

Changes the fileid to C:\PROJ\SUB\DEF.XYZ.

```
FILEID "SUB\TESTFILE.DATA"
```

Changes the fileid to C:\PROJ\SUB\TESTFILE.DATA. (Double-quotes are required with fileids that contain blanks, but are also allowed with fileids that don't contain blanks.)

```
"D:My file.txt"
```

Changes the fileid to D:\WORK\My file.txt. No path specification was given but the drive specifier changed, so the current directory of the new drive is used. Fileid specifications that include blanks must be enclosed in double quotes.

```
FILEID =.TMP
```

Changes the fileid to C:\PROJ\PROG.TMP.

```
FILEID \NEW\=
```

This changes the fileid to C:\NEW\PROG if DEFEXT OFF is in effect, and to C:\NEW\PROG.TXT if DEFEXT ON is in effect.

```
FMODE .
```

Changes the drive to the current default drive, which is D:, and since the drive is changing, the current directory of the new drive is used, yielding D:\WORK\PROG.TXT.

```
FNAME FRED
```

Changes the fileid to C:\PROJ\FRED.TXT.

```
FPATH \KEEP
```

Changes the fileid to C:\KEEP\PROJ.TXT.

SET FORMAT

Format

```
[Set] FORMAT Justify|NOJustify [BLANK|EXTENded
      [SIngle|DOuble]]
```

KEDIT default: NOJUSTIFY BLANK DOUBLE

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description

SET FORMAT affects how paragraphs are reformatted when you use the FLOW command or press Shift+Ctrl+F (or, with INTERFACE CLASSIC in effect, press Ctrl+F). It also affects what KEDIT considers a paragraph when you use PARAGRAPH targets.

JUSTIFY or NOJUSTIFY controls whether paragraphs are justified within the margins or are left with ragged right margins when they are reformatted.

BLANK or EXTENDED controls what KEDIT considers to be a paragraph when you reformat text or when you use PARAGRAPH as a target. With BLANK, the default, paragraphs are separated by completely blank lines. With EXTENDED, blank lines can still separate paragraphs, but the start of a new paragraph is also recognized if the left margin is in column 1 and a line is encountered that starts with a blank, a tab character, a period (“.“), a less than (“<”), a greater than (“>“), or a colon (“:”). This allows paragraphs that are indented but not separated by blank lines. It also treats lines that start with HTML tags, with the “dot” commands used with some text formatting programs, or with the greater than (“>”) that is often used to quote text in e-mail messages, as marking the start of a paragraph.

SINGLE or DOUBLE controls the number of blanks placed after a sentence during paragraph reformatting. With DOUBLE, the default, KEDIT will insert two spaces after each sentence. (Unfortunately, KEDIT is sometimes wrong about when it has reached the end of a sentence, because it can be confused by certain acronyms, abbreviations, and short words beginning with a capital letter.) If SINGLE is in effect, KEDIT will only insert one space after a sentence. (Note that in either case, additional spaces may be added when FORMAT JUSTIFY is in effect and KEDIT must add extra spaces to move text into the right margin column.)

Example

```
FORMAT JUSTIFY EXTENDED
```

This tells KEDIT that you want paragraphs to be justified when you format them, and that paragraph boundaries may be determined by lines starting with a period, colon, less than, greater than, blank, or tab.

SET HELPDIR

Format **[Set] HELPDIR *directory***

KEDIT default: *PROGRAM

Level: Global

Dialog box: None

Save Settings handling: Savable

Description SET HELPDIR is a specialized command needed only by users who cannot access the KEDIT Help file because it is stored on a network device.

KEDIT's Help file, KEDITW.CHM, is in HTML Help format and is normally installed in the KEDIT program directory. Because of a Microsoft security fix issued in 2005, HTML Help files stored on a network drive don't work without special changes to the registry. If your copy of KEDIT is installed on a network drive, you can copy KEDITW.CHM to a local directory and use SET HELPDIR to tell KEDIT where it is.

The default value of HELPDIR is *PROGRAM, which means that KEDIT will look for KEDITW.CHM in the KEDIT program directory and nowhere else.

If HELPDIR is set to the name of a directory, for example C:\MyKeditHelp, KEDIT will look for KEDITW.CHM in that directory. If KEDITW.CHM is not found there, KEDIT will then look for it in the KEDIT program directory.

Notes

SET HELPDIR is most often issued from within the WINPROF.KEX macro that is automatically executed when KEDIT starts up.

The KEDIT Setup program does not look for or update copies of KEDITW.CHM that are stored outside of the KEDIT program directory. So if you copy the Help file into a different directory and use SET HELPDIR to access it, you will need to recopy it if you later install an updated version of KEDIT that has a newer version of the Help file.

SET HEX

Format **[Set] HEX ON|OFF**

KEDIT default: OFF

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description

When you set HEX ON, you can enter hexadecimal and decimal codes for strings used in target searches and with a number of KEDIT commands.

With HEX ON, any string used in a string target that starts with X' (that is, "X", in uppercase or lowercase, followed by a single quote) is assumed to contain the character codes in hexadecimal for the characters involved. Strings that start with D' (that is, "D", in uppercase or lowercase, followed by a single quote) are assumed to contain the character codes in decimal for the characters involved. Both types of strings must also end with a single quote. Decimal numbers must be in the range 0 to 255 and are separated by blanks. Hexadecimal numbers must be in the range 00 to FF; each character is represented by exactly two hexadecimal digits. Pairs of hexadecimal digits may or may not be separated by blanks.

For example, the character codes for "1", "2", "3", and "4" are 49, 50, 51, and 52 in decimal and 31, 32, 33, and 34 in hexadecimal. With HEX ON, the following string targets are equivalent:

```
/1234/
/d'49 50 51 52'/
/x'31323334'/
/x'31 32 33 34'/
```

In addition to string targets, you can take advantage of HEX ON for the string operands of CHANGE, COUNT, and SCHANGE, and for the text operands of CAPPEND, CINSERT, COVERLAY, CREPLACE, FILLBOX, FIND, FINDUP, INPUT, NFIND, NFINDUP, OVERLAY, PRINT, PUT, REPLACE, and TEXT. It also affects strings entered into the Edit Find, Edit Replace, and Edit Selective Editing boxes.

SET HEXDISPLAY

Format

`[Set] HEXDISPlay ON|OFF`

KEDIT default: OFF

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description

With HEXDISPLAY ON, KEDIT will display on the status line the hexadecimal and decimal values of the character code for the character at which the cursor is located.

For example, if the cursor is positioned at a lowercase "e", whose character code is 65 in hexadecimal and 101 in decimal, KEDIT will display, in the status line:

```
'e'=65/101
```

See also SET STATUSLINE

SET HIGHLIGHT

Format **[Set] HIGHlight OFF|ALTered|TAGged|SELect *n* [*m*]**

KEDIT default: OFF

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET HIGHLIGHT controls KEDIT's highlighting facility, determining which types of lines are highlighted on your display.

[Set] HIGHlight OFF
This is the default, and it means that no lines are highlighted.

[Set] HIGHlight ALTered
Tells KEDIT to highlight all lines in your file that have been altered during the current editing session. (KEDIT determines whether a line has been altered by looking at the flag bits associated with each line to see if the new bit or change bit are set. See SET LINEFLAG for a discussion of these flag bits.)

[Set] HIGHlight TAGged
Tells KEDIT to highlight all lines whose tag bits are set. Tag bits are most often set via the TAG command, which automatically puts HIGHLIGHT TAGGED into effect whenever it is issued.

[Set] HIGHlight SELect *n* [*m*]
Highlights all lines with a given selection level, or whose selection levels fall in a specified range.

SET COLOR HIGHLIGHT determines the colors KEDIT uses to display highlighted lines. With highlighted text for which syntax coloring also applies, KEDIT attempts to merge the two colors. KEDIT uses the background color from the SET COLOR HIGHLIGHT setting, and the foreground color from the appropriate ECOLOR setting. You need to take this into account when you decide on the colors you will use with SET COLOR HIGHLIGHT and SET ECOLOR, since not all combinations of foreground and background colors work well together.

See also User's Guide Chapter 8, "Selective Line Editing and Highlighting", TAG, SET ECOLOR, SET LINEFLAG

SET IDLINE

Format **[Set] IDline ON|OFF**

KEDIT default: OFF

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description Use SET IDLINE to tell KEDIT whether to display the ID line.

The ID line is an optional line of information displayed at the top of the file area in a document window. It has information about the fileid, your position in the file, the size of the file, etc. The ID line is usually off in KEDIT for Windows because the information contained there is normally displayed elsewhere: the fileid is on the title bar, and the current position in the file, the size of the file, and the number of alterations to the file are on the status line. SET IDLINE is provided mainly for compatibility with the text mode version of KEDIT.

SET IMPMACRO

Format **[Set] IMPMACro ON|OFF**

KEDIT default: ON

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description You can set IMPMACRO (“implied macro”) ON (the default) or OFF. With IMPMACRO ON, when you enter a command that KEDIT does not recognize, KEDIT checks to see if what you entered is instead the name of a macro, and if so, KEDIT runs the macro. With IMPMACRO OFF, when you enter a command that KEDIT does not recognize, KEDIT issues an error message.

The advantage of IMPMACRO ON is that you can invoke most KEDIT macros directly, without having to issue the MACRO command, and can treat the macros as if they were built-in KEDIT commands. The disadvantage of IMPMACRO ON is that, when you really do make a typing error and enter an invalid command, there will be a slight delay while KEDIT checks your disk to see if the invalid command is really a macro.

IMPMACRO ON only helps you when the macro name involved consists entirely of alphabetic characters. If you want to invoke a macro whose name contains numeric characters, or you want to give a drive or path specification for a macro name, you must use the MACRO command to run the macro. For example, even with IMPMACRO ON,

```
MACRO DEL5
```

would run the macro DEL5, while

```
DEL5
```

would instead delete five lines from your file. And

```
MACRO C:TEST
```

would run the macro TEST.KEX on your C: drive, while

```
C:TEST
```

would not run your macro and is, in fact, a valid CHANGE command.

See also MACRO

SET INISAVE

Format **`[Set] INISAVE STATE|NOSTATE [HISTory|NOHISTory]`**

KEDIT default: STATE HISTORY

Level: Global

Dialog box: None

Save Settings handling: Not savable

Description SET INISAVE does the same thing as SET REGSAVE. See the discussion of SET REGSAVE for a full description of the operands involved.

(KEDIT for Windows now stores its configuration information in the Windows registry, but KEDIT for Windows 1.5 and earlier stored this information in the KEDITW.INI file. So SET REGSAVE is the newer name for this option, but for compatibility reasons SET INISAVE remains available.)

See also NOREG initialization option, REGUTIL, SET REGSAVE

SET INITIALDIR

Format `[Set] INITIALDIR PRESERVE|RECALL [PRESERVE|RECALL|FIRSTFile]`

KEDIT default: PRESERVE PRESERVE

Level: Global

Dialog box: Options SET Command

Save Settings handling: Automatically saved

Description

SET INITIALDIR controls the current directory put into effect at the start of a KEDIT session.

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.

When KEDIT begins execution, the current directory is normally the Windows Documents or My Documents folder. If you invoke KEDIT via a desktop shortcut, you can override this by specifying a different "Start In" directory in the shortcut's Properties dialog box. With the default setting of INITIALDIR PRESERVE PRESERVE, KEDIT leaves this current directory in effect.

It is sometimes more useful to have the directory of the files that you are actually working with as the current directory, rather than the directory from which KEDIT was loaded. So, SET INITIALDIR lets you make the directory of the first file edited at the start of a KEDIT session become the current directory. But in the case where you double-click on the KEDIT for Windows icon to run KEDIT, there is no initial fileid specified and KEDIT starts out by editing the file UNTITLED.1. In this situation, SET INITIALDIR provides a way to set the current directory to whatever was the current directory the last time you ran KEDIT.

The first operand of SET INITIALDIR determines what the current directory will be if no fileid is used to start a KEDIT session. There are two choices:

PRESERVE
: This is the default; KEDIT stays with the directory already put into effect by Windows.

RECALL
: KEDIT switches to whatever directory was the current directory at the end of the last KEDIT session.

The second operand of SET INITIALDIR determines what the current directory will be if a fileid is used to start a KEDIT session. This happens when you explicitly invoke KEDIT by specifying a fileid. It also happens if, in the Windows File Manager or

Explorer, you drag-and-drop a file on KEDIT's icon or double-click on a file whose filetype has been associated with KEDIT. There are three choices for the second operand:

PRESERVE
This is the default; KEDIT stays with the directory already put into effect by Windows.

RECALL
KEDIT switches to whatever directory was the current directory at the end of the last KEDIT session.

FIRSTFILE
KEDIT switches to the directory of the first file edited.

Notes

KEDIT determines the initial current directory while it is initializing, before you have a chance to enter any commands and before any macros, even your profile macro, are executed. So, by the time you issue the SET INITIALDIR command it has no effect at all on the current KEDIT session. The purpose of SET INITIALDIR is to determine the initial current directory for future KEDIT sessions, and to have any effect the value of INITIALDIR must be saved in the Windows registry. Therefore, whenever you issue the SET INITIALDIR command, the registry is automatically updated to reflect its new value; you do not need to save it by using Options Save Settings or by using the Save Setting button of the SET Command dialog box.

SET INITIALDIR only affects how KEDIT determines the current directory at the start of a KEDIT session. After KEDIT has determined the initial current directory, KEDIT makes no further automatic changes to your current directory, although you can change the current directory yourself by using the File Directory dialog box or the CHDIR and CHDRIVE commands.

If KEDIT is unable to switch to the directory specified by SET INITIALDIR (for example, if the directory in effect at the end of the last KEDIT session no longer exists), KEDIT will leave the current directory unchanged.

See also

CHDIR, CHDRIVE

Examples

```
SET INITIALDIR RECALL FIRSTFILE
```

If no fileid is specified in the command string used to start KEDIT, KEDIT switches to the current directory in effect at the end of the last KEDIT session. If a fileid is specified, KEDIT switches to the directory of that file.

```
SET INITIALDIR RECALL RECALL
```

KEDIT will always switch to whatever was the current directory at the end of the last KEDIT session.

SET INITIALDIR PRESERVE PRESERVE

This is the default; KEDIT will leave unchanged the current directory put into effect by Windows when KEDIT is loaded.

SET INITIALDOCSIZE

Format **[Set] INITIALDOCsize MAXimized|NORMal|RECALL**

KEDIT default: MAXIMIZED

Level: Global

Dialog box: Options SET Command

Save Settings handling: Automatically saved

Description

SET INITIALDOCSIZE determines whether the initial document window created when KEDIT starts up is maximized within KEDIT's frame window or is in the non-maximized state sometimes referred to as the ''normal'' or ''restored'' state.

INITIALDOCSIZE MAXIMIZED
This is the default. KEDIT starts with a maximized document window.

INITIALDOCSIZE NORMAL
KEDIT starts with a ''normal'' document window, neither minimized nor maximized, usually occupying only part of the area of the frame window.

INITIALDOCSIZE RECALL
KEDIT starts with the state, maximized or non-maximized, that was in effect for the last document window closed in your last KEDIT session.

Notes

KEDIT determines the initial document window size while it is initializing, before you have a chance to enter any commands and before any macros, even your profile macro, are executed. So, by the time you issue the SET INITIALDOCSIZE command it has no effect at all on the current KEDIT session. The purpose of SET INITIALDOCSIZE is to determine the initial document window size for future KEDIT sessions, and to have any effect the value of INITIALDOCSIZE must be saved in the Windows registry. Therefore, whenever you issue the SET INITIALDOCSIZE command, the registry is automatically updated to reflect its new value; you do not need to save it by using Options Save Settings or by using the Save Setting button of the SET Command dialog box.

SET INITIALDOCSIZE only affects how KEDIT determines the state of the initial document window. When additional document windows are created during a KEDIT session, the newly added windows are maximized if the current document window is maximized and otherwise they are non-maximized. If, in the course of a KEDIT session, you close all of your document windows and then create a new document window, that window will be maximized or non-maximized depending on the state of the last window closed.

See also SET DOCSIZING, SET INITIALFRAMESIZE

SET INITIALFRAMESIZE

Format **`[Set] INITIALFRAMEsize MAXimized|NORMal|RECALL`**

KEDIT default: RECALL

Level: Global

Dialog box: Options SET Command

Save Settings handling: Automatically saved

Description SET INITIALFRAMESIZE determines whether KEDIT's frame window is maximized or is in the non-maximized state sometimes referred to as the ''normal'' or ''restored'' state.

`INITIALFRAMESIZE MAXIMIZED`
KEDIT starts with a maximized frame window.

`INITIALFRAMESIZE NORMAL`
KEDIT starts with a ''normal'' frame window, neither minimized nor maximized, usually occupying only part of your screen.

`INITIALFRAMESIZE RECALL`
KEDIT starts with the frame window state, maximized or non-maximized, that was in effect at the end of your last KEDIT session.

Notes

KEDIT determines the initial frame window size while it is initializing, before you have a chance to enter any commands and before any macros, even your profile macro, are executed. So, by the time you issue the SET INITIALFRAMESIZE command it has no effect at all on the current KEDIT session. The purpose of SET INITIALFRAMESIZE is to determine the initial frame window size for future KEDIT sessions, and to have any effect the value of INITIALFRAMESIZE must be saved in the Windows registry. Therefore, whenever you issue the SET INITIALFRAMESIZE command, the registry is automatically updated to reflect its new value; you do not need to save it by using Options Save Settings or by using the Save Setting button of the SET Command dialog box.

To override the INITIALFRAMESIZE value for a particular KEDIT session, you can specify the FRAMESIZE initialization option on the command line used to invoke KEDIT.

See also FRAMESIZE initialization option, SET INITIALDOCSIZE

SET INITIALINSERT

Format **`[Set] INITIALINSert ON|OFF`**

KEDIT default: OFF

Level: Global

Dialog box: Options SET Command

Save Settings handling: Automatically saved

Description SET INITIALINSERT determines whether KEDIT starts out with Insert Mode in effect.

With INITIALINSERT OFF, the default, INSERTMODE OFF is put into effect when KEDIT is loaded, and you are in Overtype Mode.

With INITIALINSERT ON, INSERTMODE ON is put into effect when KEDIT is loaded, and you are in Insert Mode.

Notes

KEDIT determines the initial Insert Mode state while it is initializing, before you have a chance to enter any commands and before any macros, even your profile macro, are executed. So, by the time you issue the SET INITIALINSERT command it has no effect at all on the current KEDIT session. The purpose of SET INITIALINSERT is to determine the initial Insert Mode state in future KEDIT sessions, and to have any effect the value of INITIALINSERT must be saved in the Windows registry. Therefore, whenever you issue the SET INITIALINSERT command, the registry is automatically updated to reflect its new value; you do not need to save it by using Options Save Settings or by using the Save Setting button of the SET Command dialog box.

In contrast to SET INITIALINSERT, SET INSERTMODE immediately puts you into or takes you out of Insert Mode. A SET INSERTMODE command in your profile could therefore be used to determine whether KEDIT starts out in Insert Mode, and would override the state set by INITIALINSERT. INITIALINSERT is available as a separate command so that KEDIT users can control this aspect of KEDIT's behavior solely via Options SET Command and Options Save Settings, without the need for a profile.

See also SET INSERTMODE

SET INITIALWIDTH

Format **`[Set] INITIALWidth n`**

KEDIT default: 10000

Level: Global

Dialog box: Options SET Command

Save Settings handling: Automatically saved

Description SET INITIALWIDTH determines the WIDTH value that KEDIT puts into effect at the start of each KEDIT session. The WIDTH value determines the length of the longest line that you can edit during that KEDIT session. Its value can range from 1024 to 999999. The default is 10000.

Notes

KEDIT determines the WIDTH value while it is initializing, before you have a chance to enter any commands and before any macros, even your profile macro, are executed. So, by the time you issue the SET INITIALWIDTH command it has no effect at all on the current KEDIT session. The purpose of SET INITIALWIDTH is to determine the WIDTH used in future KEDIT sessions, and to have any effect the value of INITIALWIDTH must be saved in the Windows registry. Therefore, whenever you issue the SET INITIALWIDTH command, the registry is automatically updated to reflect its new value; you do not need to save it by using Options Save Settings or by using the Save Setting button of the SET Command dialog box.

To override the INITIALWIDTH value for a particular KEDIT session, you can specify the WIDTH initialization option on the command line used to invoke KEDIT.

Once the WIDTH value is established at the start of a KEDIT session, it remains in effect for the duration of that session and cannot be changed.

See also WIDTH initialization option

SET INPUTMODE

Format **`[Set] INPUTMode OFF|FUll|LIne`**

KEDIT default: OFF

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description

When you enter the INPUT command with no operands and INPUTMODE is OFF (the default), a blank line is added to your file and the cursor is positioned in the left margin column of the line.

If INPUTMODE FULL is in effect and you enter the INPUT command with no operands, KEDIT will simulate the Input Mode provided with IBM's XEDIT editor. INPUTMODE FULL has been added to KEDIT mainly for the convenience of frequent users of XEDIT. The prefix area will temporarily be turned off and blank lines will appear on your screen below the focus line. The cursor is positioned in the left margin column of the first of these blank lines and you can enter text, using the Enter key to move to the beginning of each new line. When you have entered enough lines to reach the bottom of the document window, KEDIT will automatically readjust the newly entered text to make room on the screen for more input. To leave Input Mode, move the cursor to the command line; you normally do this by pressing the F12 key (if you are using INTERFACE CUA) or the Home key (if you are using INTERFACE CLASSIC).

If INPUTMODE LINE is in effect and you enter the INPUT command with no operands, KEDIT adds one blank line to your file and positions the cursor in its left margin column. What is special about INPUTMODE LINE is that, until you leave Input Mode by moving the cursor to the command line (normally done in INTERFACE CLASSIC by pressing the Home key), you can add new lines to your file by pressing the Enter key, instead of the less convenient F2 key that is normally used with INTERFACE CLASSIC. This gives you an easy way to add lines to your file without having a large part of your screen filled with blank lines, as it is with INPUTMODE FULL. INPUTMODE LINE is primarily intended for users if INTERFACE CLASSIC, since by default the Enter key always adds new lines to a file when INTERFACE CUA is in effect.

The REPLACE command with no operands is similar to the INPUT command with no operands. INPUT with no operands adds a blank line, while REPLACE with no operands replaces the focus line with a blank line. Both can trigger KEDIT's Input Mode, depending on the setting of INPUTMODE.

XEDIT users should note that with INPUTMODE FULL and WORDWRAP ON, KEDIT's Input Mode is very close to XEDIT's Power Input Mode.

SET INSERTMODE

Format **`[Set] INSERTmode ON|OFF|TOGGLE`**

KEDIT default: Dependent on the value of INITIALINSERT

Level: Global

Dialog box: None

Save Settings handling: Not savable

Description With INSERTMODE ON, characters that you enter at the keyboard are inserted into the cursor line at the cursor position. Existing text shifts to the right to make room for the newly-inserted text. (If WORDWRAP ON is in effect, text shifted beyond the right margin column may be split off to a new line.) With INSERTMODE OFF, characters entered at the keyboard overtype existing text at the cursor location.

INSERTMODE TOGGLE, normally assigned to the Ins key, toggles the status of INSERTMODE, from ON to OFF or from OFF to ON.

The status of INSERTMODE is indicated on the status line, where ''INS'' is displayed if you are in Insert Mode and ''OVR'' is displayed if you are in Overtype Mode.

The cursor size also gives you an indication of the status of INSERTMODE, with a larger cursor normally displayed when INSERTMODE ON is in effect. The cursor's appearance is controlled by the SET CURSORTYPE and SET CURSORSIZE commands.

See also SET INITIALINSERT

SET INSTANCE

Format **`[Set] INSTANCE SINGLE|MULTIPLE`**

KEDIT default: SINGLE

Level: Global

Dialog box: Options SET Command

Save Settings handling: Automatically saved

Description SET INSTANCE controls whether multiple copies of KEDIT, or instead only a single copy of KEDIT, can be active at the same time.

With INSTANCE SINGLE, the default, only a single copy of KEDIT is active at a time. If a copy of KEDIT is already active when you double-click on the KEDIT for Windows icon in the Program Manager, double-click in the File Manager on a file that is associated with KEDIT, or otherwise try to run KEDIT, the existing instance of KEDIT is re-activated, with any new files to be edited added to the existing instance's ring of files. With INSTANCE MULTIPLE, a new instance of KEDIT, with a separate frame window and ring of files, is created in each of these cases, and it is possible to have several copies of KEDIT active at the same time.

INSTANCE SINGLE is the default because, when you want to edit several files at a time, it is normally more convenient to have one copy of KEDIT working with multiple files than it is to have multiple copies of KEDIT running simultaneously.

Notes

The value of the INSTANCE option is only used when you attempt to start a new instance of KEDIT for Windows. By the time you issue the SET INSTANCE command it therefore has no effect at all on the current KEDIT session, since either a new instance has already been started or an existing instance has already been re-activated. The purpose of SET INSTANCE is to influence the processing that takes place at the start of future KEDIT sessions, and to have any effect, the value of SET INSTANCE must be saved in the Windows registry. Therefore, whenever you issue the SET INSTANCE command, the registry is automatically updated to reflect its new value; you do not need to save it by using Options Save Settings or by using the Save Setting button of the SET Command dialog box.

To override the INSTANCE setting for a particular KEDIT session, you can specify the INSTANCE SINGLE or INSTANCE MULTIPLE initialization options on the command line used to invoke KEDIT.

See also INSTANCE initialization option

SET INTERFACE

Format **`[Set] INTERFACE CUA|CLASSIC`**

KEDIT default: CUA

Level: Global

Dialog box: Options SET Command, Options Interface

Save Settings handling: Savable

Description Use SET INTERFACE to control whether certain aspects of KEDIT for Windows' user interface 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 5.0.

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

Default behavior for most keys is based on CUA conventions. For example, the Ctrl+End key moves the cursor to the end of the file and Alt+W opens the Window menu.

Blocks marked with the mouse or with Shift+cursor key combinations 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.

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

By default, KEDIT uses a vertical text cursor.

You can use the SET KEYSTYLE and SET MARKSTYLE commands, or the Options Interface dialog box, to modify some aspects of KEDIT's behavior when INTERFACE CUA is in effect. These commands are primarily aimed at former users of text mode KEDIT. They let you take advantage of most aspects of INTERFACE CUA while letting you make a few aspects of KEDIT's behavior more like the behavior of text mode KEDIT. SET KEYSTYLE lets you adjust the behavior of some of KEDIT's keys, while SET MARKSTYLE controls whether the mouse marks selections or persistent blocks.

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

Default behavior for most keys is very close to the behavior in text mode KEDIT 5.0. For example, the Ctrl+End key deletes text from the cursor through the end of the line, and Alt+W deletes a word.

Blocks do not exhibit CUA-style behavior. All blocks in INTERFACE CLASSIC are persistent blocks: typing a character does not replace their contents, and cursor movement does not unmark them.

Command line selections are not available.

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

SET KEYSTYLE and SET MARKSTYLE have no effect.

See also SET KEYSTYLE, SET MARKSTYLE

SET INTERNATIONAL

Format **`[Set] INTERNATional CASE|NOCASE [SORT|NOSORT]`**

KEDIT default: NOCASE NOSORT

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET INTERNATIONAL controls some aspects of how KEDIT works with international characters.

CASE|NOCASE INTERNATIONAL CASE|NOCASE affects how text in your files is handled by the UPPERCASE and LOWERCASE commands, and in case-insensitive string searches.

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 string searches.

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.

SORT|NOSORT KEDIT's handling of the SORT command and international characters is somewhat complicated, because it depends on the value of SET CASE, and on both the first and second operands of SET INTERNATIONAL.

• If the second operand of SET INTERNATIONAL is set to NOSORT, the SORT command works like this:

- If SET CASE's first IGNORE|RESPECT operand is set to RESPECT, all characters are sorted according to the value of their character code, without regard to whether the characters are alphabetic and without regard to case.

- If SET CASE's first IGNORE|RESPECT operand is set to IGNORE, then the SORT command does pay attention to whether the characters are alphabetic and whether they are uppercase or lowercase. If INTERNATIONAL NOCASE is in effect, the SORT command lowercases the 26 letters of the English alphabet before making its comparisons. If INTERNATIONAL CASE is in effect, KEDIT uses your Windows language driver to decide which characters are alphabetic, and it lowercases any uppercase alphabetic characters, which may include accented characters, etc., before making its comparisons.

• If the second operand of SET INTERNATIONAL is set to SORT, the SORT command does not compare the lines to be sorted one character at a time, in the way that it

does when the second operand of SET INTERNATIONAL is NOSORT. Instead, when it compares text in the sort fields of two lines of your file, it uses your Windows language driver to compare the entire strings as a unit.

If SET CASE's first IGNORE|RESPECT operand is set to RESPECT, the comparison is case sensitive; if it is set to IGNORE, the comparison is case insensitive.

There are two reasons for using your Windows language driver to compare the strings to be sorted, and for comparing entire strings at a time, rather than on a character-by-character basis:

First, this allows the Windows language driver to determine how accented letters are sorted with respect to non-accented versions of the characters involved. With the second operand of SET INTERNATIONAL set to NOSORT, accented letters will always sort after all 26 of unaccented letters of the English alphabet, but most Windows language drivers sort the accented letters, using language-dependent rules, intermixed with, or immediately after, the unaccented versions of those letters.

Second, by doing the sort based on the entire strings involved, your Windows language driver can apply language-dependent rules that give special handling to some multi-character sequences. For example, the Windows English language driver gives special handling to the Latin character "Æ", sorting it between the two character sequences "AE" and "AF", yielding orderings like this: "ADC", "AEF", "ÆC", "AFC".

Notes

- SET INTERNATIONAL has no effect on the behavior of KEXX instructions or built-in functions, such as the UPPER() and LOWER() KEXX functions. Most KEXX instructions and built-in functions always consider only the 26 letters of the English alphabet as alphabetic; the exceptions are the ANSIUPPER(), ANSILOWER(), and ANSIDATATYPE() built-in functions, which depend on your Windows language driver to decide which characters are alphabetic, uppercase, and lowercase.

- 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; regardless of the actual value of SET INTERNATIONAL, international case and sort processing is bypassed in this situation.

See also

User's Guide Section 3.8, "International Support"

SET KEYSTYLE

Format

`[Set] KEYSTYLE` *`enter`* `[`*`home delete backspace alt`*`]`

where each value can be either

`Standard|ADJusted`

KEDIT default: STANDARD STANDARD STANDARD STANDARD STANDARD

Level: Global

Dialog box: Options SET Command, Options Interface

Save Settings handling: Savable

Description

The default behavior for many keys is very different when INTERFACE CUA is in effect than when INTERFACE CLASSIC is in effect. Users of text mode KEDIT who switch to KEDIT for Windows often want to use INTERFACE CUA so that KEDIT will act as much as possible like other Windows applications. But sometimes they are not comfortable with the new behavior of a few keys that are frequently used in text mode KEDIT. SET KEYSTYLE lets you change the default behavior of some of the keys that text mode KEDIT users have found hardest to get used to.

There are five keys whose behavior you can adjust. For each of the five keys, you can specify that, when INTERFACE CUA is in effect, their behavior is STANDARD (that is, they have the standard CUA behavior; this is the default for all five keys) or that their behavior is ADJUSTED (in which case they behave more like they would in text mode KEDIT). SET KEYSTYLE only makes a difference when INTERFACE CUA is in effect; when INTERFACE CLASSIC is in effect, all five of the keys have text mode compatible behavior, regardless of the value of the KEYSTYLE option.

1. The first operand controls the behavior of the Enter key. Its STANDARD behavior when the cursor is in the file area and INTERFACE CUA is in effect is to insert a new line following the character at the cursor position. The ADJUSTED behavior of the Enter key when the cursor is in the file area is to move the cursor to the left margin column of the next line.

2. The second operand controls the Home key. Its STANDARD behavior is to move the cursor to the start of the line that it is on. Its ADJUSTED behavior is to move the cursor to the command line and also to execute any pending prefix commands.

3. The third operand controls the Delete key. Its STANDARD behavior when the cursor is at the end of a line is to join the text from the following line to the cursor position. Its ADJUSTED behavior is to do nothing when the cursor is at the end of a line.

4. The fourth operand controls the Backspace key. Its STANDARD behavior when the cursor is at the start of a line is to join the text of that line to the end of the preceding line. Its ADJUSTED behavior is to do nothing when the cursor is at the start of a line.

5. The fifth operand controls the Alt key, when it is pressed alone and not in combination with any other key. Its STANDARD behavior is to activate the menu bar at the top of KEDIT's frame window. Its ADJUSTED behavior is to do nothing.

Notes

When you update the settings in the section of the Options Interface dialog box labeled Adjust CUA Keyboard Behavior, you are actually updating the value of the KEYSTYLE option.

The default key definitions for the five keys involved check the value of the KEYSTYLE option and act accordingly. If you supply your own definitions for these keys, SET KEYSTYLE will have no effect on how they behave unless you check the value of KEYSTYLE within your replacement key definitions.

SET KEYSTYLE is provided as an easy way to tailor the behavior of the keys that most often cause problems for users of text mode KEDIT who switch to KEDIT for Windows. You can do much more extensive tailoring of these keys, or of any other KEDIT keys, by using KEDIT's macro language to replace KEDIT's default key definitions with definitions of your own.

See also SET INTERFACE, SET MARKSTYLE

Examples

```
SET KEYSTYLE ADJUSTED ADJUSTED STANDARD STANDARD STANDARD
```

Set the Enter and Home keys to have their ADJUSTED behavior, with the Enter key moving to the left margin column of the next line and the Home key moving to the command line and processing any pending prefix commands. Set the Delete, Backspace, and Alt keys to have their standard CUA behavior.

SET LASTOP

Format **[Set] LASTOP *commandname text***

KEDIT default: Undefined

Level: Global

Dialog box: None

Save Settings handling: Not savable

Description

With SET LASTOP, most often used within macros, you can set the contents of KEDIT's operand memory for a particular command, controlling the operands that will be used the next time the command is issued without any operands.

The *commandname* operand specifies the command whose operand memory you want to set, and can be one of ALter, Change, CLocate, COUnt, Find, Locate, SCHange, or TFind. *Text* specifies the new contents of the command's operand memory.

KEDIT ‘‘remembers’’ the last operands of several KEDIT commands, so that you can reissue the same command again without having to type in the operands all over again. For example, if you issue the CHANGE command with no operands, the last CHANGE command that you issued will be repeated. KEDIT only remembers the operands of commands issued directly from the KEDIT command line; to avoid confusion, operands of commands issued from within a macro are not automatically remembered. Edit Find, Edit Replace, and searches done from the toolbar also have no effect on the value of LASTOP.

Examples

```
SET LASTOP LOCATE /1234/
```

If, when you next issue a LOCATE command, no operands are given, KEDIT will locate the string ‘‘1234’’.

```
SET LASTOP FIND 1234
```

Sets the operand memory for the FIND command. (The operand memory for the FIND command also affects the related FINDUP, NFIND, and NFINDUP commands.)

SET LINEFLAG

Format

[Set] LINEFLAG *flagbits* [*target*]

where *flagbits* can be one or more of the following:

```
CHAnge|NOCHAnge
NEW|NONEW
TAG|NOTAG
```

KEDIT default: NOCHANGE NONEW NOTAG

Level: File

Dialog box: Options SET Command (for focus line only)

Save Settings handling: Not savable

Description

Use the SET LINEFLAG command to control the values of the flag bits for lines in your file.

Associated with each line in your file are three flag bits, known as the change bit, the new bit, and the tag bit. You can use the SET LINEFLAG command to set or clear one or more of these bits for the lines in the specified *target* area.

When KEDIT initially reads a file in from disk, it sets all three bits off for all lines in the file. While you are editing your file, KEDIT sets the change bit for any line that is changed, added, moved, or sorted. For any line that is added to your file, KEDIT also sets the new bit. KEDIT sets or clears tag bits when processing the TAG, MORE, LESS, and SET LINEFLAG commands.

See also TAG, SET HIGHLIGHT

Examples

```
SET LINEFLAG TAG NOCHANGE 10
```

For each of the next 10 lines, beginning with the focus line, this command would turn the TAG bit on, turn the change bit off, and leave the new bit unaffected.

```
SET LINEFLAG NONEW NOCHANGE ALL
```

Clears the new and change bits for all lines in the file.

SET LINEND

Format **[Set] LINENd ON|OFF [*char*]**

KEDIT default: OFF #

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description You use SET LINEND to enable or disable the ability to enter multiple commands on the KEDIT command line. If the facility is enabled, the commands are separated by the LINEND character, which is "#" by default but can be changed. If LINEND is OFF, you can only enter one command at a time on the command line, and the LINEND character has no special meaning.

If LINEND is ON, the following will duplicate the focus line and then go down four lines:

```
DUP#DOWN 4
```

If LINEND is OFF, the "#" would have no special meaning and you would have an invalid command.

LINEND only affects commands entered from the command line It has no effect on commands issued from macros or on text entered into dialog boxes.

Examples

```
LINEND ON
```

Allow multiple commands on the same command line.

```
LINEND OFF
```

Disallow multiple commands on the same command line.

LINEND ON %

Allow multiple commands on the same command line, and set the LINEND character that separates them to be ‘‘%’’.

SET LOCKING

Format **[Set] LOCKING ON|OFF**

KEDIT default: OFF

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description Use SET LOCKING to tell KEDIT to lock files that are added to the ring, preventing access to these files by other users and programs.

If LOCKING ON is in effect when you begin editing a file, KEDIT will attempt to lock the file. SET LOCKING has no effect on the status of files that are already in the ring. KEDIT locks a file by keeping an open handle for the file with a sharing mode that prevents other programs from accessing the file; SET SHARING controls the sharing mode used.

There are three special cases where KEDIT does not lock a file, even if LOCKING ON is in effect when you begin editing the file:

1. When you use the DIR command to create a DIR.DIR file and when you use the MACROS command to create a MACRO.KML file, the lock processing is bypassed, since these are normally used as temporary in-memory files.

2. When you edit files that are marked on disk as read-only, lock processing is also bypassed. There is little need to protect against changes to such a file since, unless a utility program is used to change its read-only attribute, neither you nor any other user on your network can write to it.

3. When you edit files on your A: or B: drives, KEDIT does not automatically lock the files. This is because these are normally diskette drives, which are usually not shared over a network, and there can sometimes be problems if you inadvertently change diskettes in a drive that has open files. You can still use the LOCK command or initialization option to force KEDIT to lock files on your A: and B: drives.

KEDIT displays ‘‘Lock’’ on the status line to indicate when you are editing a locked file, displays ‘‘R/O’’ when you are editing a file whose read-only attribute bit is set, and displays ‘‘R/W’’ for file that is not locked and not read-only. Additionally, if IDLINE

ON is in effect, KEDIT displays an asterisk (“*”) on the ID line in front of the fileid for a locked file, and a plus sign (“+”) in front of the fileid for a read-only file.

You can override the current setting of SET LOCKING for a particular file by using the LOCK or NOLOCK options on the KEDIT command that you use to begin editing the file. You can also use the LOCK command to lock a file that you have already begun to edit, and the UNLOCK command to unlock a file that is locked.

See also User’s Guide Chapter 12, “File Processing”, LOCK and NOLOCK initialization options, LOCK, UNLOCK, SET SHARING

SET LRECL

Format **`[Set] LRecl n`**

KEDIT default: * (WIDTH value)

Level: File

Dialog box: Options SET Command

Save Settings handling: Not savable

Description The SET LRECL (“logical record length”) command affects the length of lines written to disk during a FILE, SAVE, and PUT commands, by the autosave facility, and by File Save and related menu items. SET LRECL also affects how lines are read in if EOLIN NONE is in effect.

SET LRECL * sets the logical record length equal to the value of the WIDTH initialization option, and is the default setting. This default does not normally need to be changed.

If RECFM FIXED is in effect, all lines written to disk will be *n* characters long, where *n* is the LRECL setting. If necessary, KEDIT will pad shorter lines with blanks or will truncate longer lines to the correct length when writing your file to disk.

If RECFM VARYING is in effect, lines will be a maximum of *n* characters long, and will be truncated if necessary. Shorter lines will not be padded with blanks. For example, if you set LRECL to 80 and some of the lines in your file are more than 80 characters long, only the first 80 characters of these lines would be written out. Characters beyond column 80 would not be written out. Note that KEDIT does not give you any warnings or messages about the text that is lost.

With the possible exception of the last line of the file (which is controlled by SET EOFOUT), KEDIT adds an end-of-line sequence, controlled by SET EOLOUT, to each line. The end-of-line sequence is not considered part of the logical record length. With the default of EOLOUT CRLF, a carriage return-linefeed pair is added to each line written to disk, for a total length of *n*+2 bytes per line.

If TABSOUT ON is in effect, tab compression is done first, and then the record length is examined.

The LRECL and RECFM settings have no effect on how KEDIT reads your file in or how KEDIT processes your file while you are editing it; they only affect how KEDIT writes your file to disk. An exception to this comes when EOLIN NONE is in effect; LRECL then controls how data read from disk will be broken into lines.

Except in special situations, RECFM and LRECL can be left at their default values and ignored. With the default settings of RECFM VARYING and LRECL equal to the value of the WIDTH initialization option, KEDIT will never need to pad or truncate lines when writing them to disk.

See also User's Guide Chapter 12, "File Processing", WIDTH initialization option, SET RECFM, SET TRAILING

SET MACROPATH

Format **`[Set] MACROPath ON|OFF|`*`envvar`*`|`*`dirlist`*

KEDIT default: PATH

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET MACROPATH controls the path search done by KEDIT when you want to run a macro that cannot be found in memory or in the current directory. It also controls the path search done by the DEFINE command when you load a macro or a .KML file into memory, by the SET TOOLBUTTON command when you load a bitmap file from disk, and by the SET PARSER command when you load a KEDIT Language Definition file.

`[Set] MACROPath ON`
The default. Tells KEDIT to search for macros in each of the directories specified via the PATH variable in the system environment.

`[Set] MACROPath OFF`
KEDIT will look for macros only in memory and in the current directory. In this special case, KEDIT will not do its normal path search for macros, and also won't do its usual search of the KEDIT Macros, KEDIT program directory, and USER and SAMPLES subdirectories.

**`[Set] MACROPath `*`envvar`*
Tells KEDIT to do a path search for macros, with the list of directories to search found in the environment variable *envvar*. Directories in the list are separated from each other by a semicolon, in the same format as is traditionally used with the PATH= environment variable.

[Set] MACROPath *dirlist*

Dirlist is a string with a list of semi-colon-delimited directories. It can also contain entries that begin with an asterisk, to indicate an indirect reference to an environment variable. ‘‘=’’ is also allowed, causing a search of the directory of the current file. For example,

```
SET MACROPATH C:\TEMP;E:\SOURCE;*INCLUDE;=
```

means search the C:\TEMP and E:\SOURCE directories, then the directories listed in the INCLUDE environment variable, and then the directory of the current file.

In order for the value of MACROPATH to affect KEDIT’s initial search for your profile, it needs to be set in an earlier KEDIT session and then saved to the Windows registry via the Options Save Settings dialog box. You can also specify MACROPATH as a KEDIT initialization option, and if you do so its value will override the value in the registry.

When searching for macros, .KML files, .BMP files, and .KLD files, KEDIT proceeds as follows: If a specific drive and/or directory is specified, KEDIT looks only there. Otherwise, KEDIT looks first in the current directory, and next does a path search, as controlled by SET MACROPATH. Finally, unless MACROPATH OFF is in effect, it looks in the “KEDIT Macros” subdirectory of your Windows Documents or My Documents folder, the directory from which KEDIT for Windows was loaded, and the USER and SAMPLES subdirectories of that directory. The search ends successfully as soon as KEDIT finds the file it is looking for, and ends in failure if the file cannot be located.

As discussed in User’s Guide Section 10.2.3, “Storing Your Macros”, we normally recommend that macros that you create be kept in the “KEDIT Macros” subdirectory of your Windows Documents folder (which is sometimes known as the My Documents folder).

See also MACROPATH initialization option, MACRO, SET PATH

SET MARGINS

Format

[Set] MARgins *left right* [[+|-]*parindent*]

KEDIT default: 1 72 +0

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description

Use SET MARGINS to control your left and right margin columns and the paragraph indent column (the indentation used for the first line of a paragraph).

The MARGINS setting affects paragraph reformatting by the FLOW command, the wordwrap facility, and the LEFTADJUST, RIGHTADJUST, and CENTER commands.

For the *left* and *right* margins, enter the column number of the margin. For the paragraph indent column, you can enter a specific column number, or you can specify a positive or negative offset from the left margin column.

The left margin must be less than or equal to the right margin. The right margin and paragraph indent column must be less than or equal to the truncation column. You can specify the right margin as an asterisk (''*''); this will set the right margin to the truncation column.

See also

User's Guide Section 3.11, "Word Processing Facilities"

Examples

```
MARGINS 10 60
```

The left margin is set to 10, the right margin is set to 60, and the paragraph indent column is unchanged.

```
MARGINS 10 65 +3
```

The left margin is set to 10, the right margin is set to 65, and the paragraph indent column is three columns to the right of the left margin column, at column 13. If you later issued the command

```
MARGINS 5 60
```

the paragraph indent specification would not change, and paragraphs would be indented three columns to the right of the new left margin, to column 8.

SET MARKSTYLE

Format

`[Set] MARKSTYLE` *`line`* `[`*`box stream`*`]`

where each value can be either

`PERSISTent|SELection`

KEDIT default: SELECTION SELECTION SELECTION

Level: Global

Dialog box: Options SET Command, Options Interface

Save Settings handling: Savable

Description

SET MARKSTYLE controls whether, when you use the mouse to mark text while INTERFACE CUA is in effect, you mark selections or persistent blocks.

SET MARKSTYLE has three operands, controlling how lines, boxes, and streams are marked. The default for all three operands is SELECTION, meaning that selections are marked; the other choice is PERSISTENT, meaning that persistent blocks are marked.

1. The first operand controls whether 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 or marks a line selection.

2. The second operand controls whether dragging with Alt+button 1 in the file area marks a persistent box block or marks a box selection.

3. The third operand controls whether dragging with button 1 in the file area marks a persistent stream block or marks a stream selection.

Notes

When you update the settings in the section of the Options Interface dialog box labeled Adjust CUA Mouse Behavior, you are actually updating the value of the MARKSTYLE option.

SET MARKSTYLE only affects blocks or selections marked with the mouse and does not affect blocks or selections marked from the keyboard. The default definitions for Shift+cursor pad key, which are the key combinations used in CUA-compatible applications to mark selections, always mark selections and never mark persistent blocks. The default definitions for Alt+L, Alt+B, and Alt+Z always mark persistent line, box, and stream blocks.

The default macros that handle mouse actions check the value of the MARKSTYLE option and act accordingly. If you supply your own definitions for these macros, SET MARKSTYLE will have no effect on how they behave unless you check the value of MARKSTYLE within your replacement definitions.

See also SET KEYSTYLE

Examples

```
SET MARKSTYLE PERSISTENT PERSISTENT SELECTION
```

Sets the mouse to mark lines and boxes as persistent blocks, but to mark stream selections when dragging with mouse button 1.

SET MONITOR

Format

```
[Set] MONitor WINDOWS|COLOR|MONO
```

KEDIT default: WINDOWS

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description

Use the SET MONITOR command to control certain aspects of how KEDIT for Windows handles your display.

With MONITOR WINDOWS, the normal KEDIT for Windows handling of your display applies, the normal KEDIT for Windows colors are used by default on your display, and the SET COLOR command works as described in this document.

MONITOR COLOR and MONITOR MONO are provided only for compatibility with text mode KEDIT, and cause KEDIT's display handling, default colors, and SET COLOR handling to work as they do in text mode KEDIT. In most situations we recommend that you leave the default of MONITOR WINDOWS in effect; MONITOR COLOR and MONITOR MONO may not be supported in future versions of KEDIT for Windows.

With MONITOR COLOR, KEDIT uses the default colors used on text mode color displays. With MONITOR MONO, KEDIT uses the colors it used with the original IBM monochrome display.

Whenever the SET MONITOR command is issued, KEDIT automatically resets all colors to the proper defaults for the type of display—WINDOWS, COLOR, or MONO—that you specified.

See also SET COLOR

SET MOUSEBEEP

Format **`[Set] MOUSEBEEP ON|OFF`**

KEDIT default: ON

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET MOUSEBEEP controls whether KEDIT sounds the speaker when mouse errors within a document window, such as a mouse click at an invalid location in the file area, are detected, and when the SOS MOUSEBEEP command is executed. Note that many mouse related errors, such as invalid mouse clicks while a dialog box is displayed, are handled directly by Windows and are not affected by SET MOUSEBEEP.

See also SET BEEP, SOS MOUSEBEEP

SET MSGLINE

Format **`[Set] MSGLine ON`** ***`line`*** **`[`*** ***`n`*** **`] [Overlay]`**

KEDIT default: ON 1 5 OVERLAY

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description Use SET MSGLINE to control where in your document window messages generated by KEDIT will be displayed.

You can specify what *line* of the document window will be the first line used for message display, the number of lines to use for message display, and whether the first message line should be reserved for messages or should OVERLAY a line normally used to display a line of your file.

The first *line* used for messages can be specified relative to the top of the document window, the middle of the document window, or the bottom of the document window; see SET CURLINE for a discussion of how to do this.

You can also specify the number of lines to use for message display. In most situations, KEDIT only needs to display one message at a time. Commands like QUERY can sometimes generate several messages, and a macro can issue several messages or cause several error messages to be generated. If KEDIT has more messages to display than

you have allowed for, KEDIT uses a special pop-up window for message display. Otherwise, KEDIT displays its messages in the area you specify with SET MSGLINE.

Finally, you can specify that the first message line is normally to be used for a line of your file, and that when there is a message to display, it will OVERLAY a line of your file on the display. (Message lines other than the first are always displayed in overlay mode.)

Even though you can specify a message area that overlaps the command line, KEDIT will never overlay the command line with a message. Instead, it will skip over the command line and put the message on the line below or above the command line.

Examples

```
SET MSGLINE ON 2 5 OVERLAY
```

This is the default MSGLINE setting, specifying that messages will display starting on line 2 of the document window, with a maximum of five messages displayed. When no message is displayed in line 2, line 2 will be used to display a line of your file.

```
SET MSGLINE ON -2 5 OVERLAY
```

The second-to-the-last line of your window will be used for messages. If KEDIT needs to display more than one message, the third-to-last line, etc., up to a total of five lines, will be used.

SET MSGMODE

Format

```
[Set] MSGMode ON|OFF
```

KEDIT default: ON

Level: View

Dialog box: None

Save Settings handling: Not savable

Description

With MSGMODE ON, KEDIT processes messages (including error messages) in the normal way, displaying them on your screen and saving the text of the last message so that QUERY LASTMSG can retrieve it. With MSGMODE OFF, KEDIT does not display messages and error messages on your screen, although the text of the last message is still saved for QUERY LASTMSG.

MSGMODE OFF is intended for use in specialized situations within macros, which sometimes need to issue commands that generate messages or error messages that need not be displayed. For example, you may want to issue a CHANGE command from within a macro, but may not want KEDIT to display its message indicating how many strings were changed.

The NOMSG command provides a better solution to this problem, and should normally be used in preference to SET MSGMODE OFF.

See also NOMSG initialization option, NOMSG

SET NEWLINES

Format **`[Set] NEWLines SAMELine|BELOW|BELOWCurr`**

KEDIT default: SAMELINE

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description When you add a new line to your file with the SOS LINEADD command (normally assigned to function key F2) or because WORDWRAP is ON and you go beyond the right margin, KEDIT normally places the newly added line in the line of the document window in which the cursor is located, scrolling the text above the new line up to make room for it. This is what happens with NEWLINES SAMELINE, the default.

With NEWLINES BELOW, the new line is placed one line below the cursor line, scrolling the text below the new line down to make room for it. When the cursor is on the bottom line of the document window, KEDIT can't add a new line below the cursor line, so the new line is added at the bottom of the document window, and text above the new line scrolls up to make room for it.

NEWLINES BELOWCURR is like NEWLINES BELOW, in that new lines are added below the cursor line. The difference is that, with NEWLINES BELOWCURR, when you are at the bottom of the document window and add a new line, KEDIT scrolls the window so that the new line is positioned at the current line location, and future lines can then be added below the current line.

SET NOVALUE

Format `[Set] NOVALUE ON|OFF`

KEDIT default: OFF

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET NOVALUE provides a debugging aid for developers of KEDIT macros. With NOVALUE ON, use of uninitialized variables in a KEXX macro will cause an error, much as SIGNAL ON NOVALUE does. This brings to your attention cases where you have forgotten to use quotes around literal values in macros.

You can get much the same effect by using SIGNAL ON NOVALUE within your KEXX macros to trap references to uninitialized variables. An advantage of SET NOVALUE ON is that it catches all uses of uninitialized variables in any macro that you run, without having to change the source of the macro. A disadvantage of SET NOVALUE ON is that, since it affects all macros that you run, it may trap harmless references to uninitialized variables in existing macros and in macros that you get from other KEDIT users.

SET NUMBER

Format `[Set] NUMber ON|OFF`

KEDIT default: OFF

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description If NUMBER ON is in effect, the line number of each line appears in the prefix area of the line. Line numbers are not displayed if PREFIX OFF is in effect.

The line numbers are continually updated as you add lines to your file or delete them. The line numbers are not part of your file; they are merely displayed on the screen for your convenience. The line number associated with a line is not written to disk when the file is saved.

By default, the prefix area is 5 characters wide, and can display line numbers up to 99999. To display line numbers above this value, you can use SET PREFIXWIDTH to make the prefix area wider.

See also SET PREFIX, SET PREFIXWIDTH

SET OFPW

Format `[Set] OFPW ON|OFF`

KEDIT default: ON

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET OFPW determines whether KEDIT operates in ''one-file-per-window'' mode.

KEDIT is a Windows MDI (''Multiple Document Interface'') application. You can edit multiple files with KEDIT and display them within multiple windows. These windows are known as ''document windows''. The document windows are all displayed within a larger window, known as the ''frame window''.

When you open a new file, most Windows MDI applications create a new document window to display the file. If you want to display the file in multiple windows, you can use the Window New menu item to create an additional document window displaying the same file. When you close a file, all document windows displaying that file are destroyed. When you close a document window, that window is destroyed and, if it is the only document window displaying a file, the file itself is also closed.

With the default of OFPW ON, which is recommended for most KEDIT users, this is also how KEDIT behaves. It is referred to as one-file-per-window mode because once a document window is created and a file displayed in that window, that one file is the only file that will ever be displayed in that window. Moving to a different file (for example, by using the Next File or Previous File toolbar buttons) involves moving to a different document window.

OFPW OFF is available mainly for compatibility with text mode versions of KEDIT. With OFPW OFF, KEDIT for Windows behaves more like text mode KEDIT, where windows are not tied to particular files. Instead, any document window can display any file. When a new file is added to the ring, the new file is displayed in the current window. The old file remains in the ring, but may not be visible in any document window. The Next File and Previous File toolbar buttons do not move you to a different document window, but instead display different files in the same window.

When you close a file with OFPW OFF in effect, the document windows in which it was displayed remain, displaying whatever file preceded the removed file in the ring. When you close a document window with OFPW OFF, the file that was displayed in that window remains in the ring. Exceptions to these general rules come when you close the last file in the ring, in which case all document windows are also closed, and when you close the last document window, in which case all files in the ring are closed.

See also User's Guide Section 3.5, "Editing Multiple Files"

SET OPENFILTER

Format **`[Set] OPENFilter /text1/filter1/ ...`**

KEDIT default: See below

Level: Global

Dialog box: None

Save Settings handling: Not savable

Description SET OPENFILTER controls the file filters displayed at the bottom of the File Open dialog box. These in turn control the files in the dialog box's default list of available files.

SET OPENFILTER takes one or more pairs of strings as its operands, with one pair of strings for each entry in the list of file filters. The first string of each pair is the string that is actually displayed when you use the File Open dialog box. The second string of each pair is the filter, consisting of a DOS fileid specification (possibly including wild-card characters) or of multiple DOS fileid specifications separated by semicolons (";"), used by File Open to build the list of matching files.

The strings used with the SET OPENFILTER command are all delimited with slash ("/") characters, or with some other delimiter character not appearing in the strings.

The default value for SET OPENFILTER (which is really all one long line, though it is split here into multiple lines) is

```
SET OPENFILTER /All files (*.*)/*.*/
Text Files (*.TXT)/*.txt/
C Files (*.C;*.CPP;*.H)/*.c;*.cpp;*.h/
KEDIT Macros (*.KEX;*.KML)/*.kex;*.kml/
```

This specifies four filters. They are displayed as

```
All files (*.*)
Text Files (*.TXT)
C Files (*.C;*.CPP;*.H)
KEDIT Macros (*.KEX;*.KML)
```

with *.*, *.txt, etc. used by File Open to build the list of available files, depending on which file filter is selected.

SET PARSER

Format

[Set] PARSER *parser fileid*

KEDIT default: See the table below

Level: Global

Dialog box: None

Save Settings handling: Not savable

Description

The syntax coloring facility depends on language-specific parameter files, known as KLD (KEDIT Language Definition) files, to determine which text to display as comments, strings, keywords, etc. Use the SET PARSER command to define a syntax coloring parser and load its associated KLD file.

Use the *parser* operand to specify the name of the parser you want to define.

The *fileid* operand specifies a file, with a default extension of .KLD, containing your language definition. The format of these KEDIT Language Definition files is discussed in Chapter 8, "KEDIT Language Definition Files". KEDIT searches for the .KLD file in the same directories it uses when searching for macro files, as controlled by SET MACROPATH.

For example, if you were working with a hypothetical language called LANG and you had described the language in a KEDIT Language Definition file called LANGDEF.KLD, you could define a parser called LANG with the command

```
SET PARSER LANG LANGDEF.KLD
```

After issuing the SET PARSER command, you could then issue the command

```
SET COLORING ON LANG
```

to use this parser to control syntax coloring for the current file.

If files in your language always had an extension of, for example, .LNG, you could use the SET AUTOCOLOR command to tell KEDIT to always use the LANG parser for .LNG files:

```
SET AUTOCOLOR .LNG LANG
```

SET PARSER commands are typically executed from your KEDIT profile when KEDIT is initially loaded. For example:

```
* if first profile execution in a session,
* setup the LANG parser and then
* cause all .LNG files to be colored using the LANG parser
if initial() then do
    'set parser lang langdef.kld'
    'set autocolor .lng lang'
    end
```

Several language definitions are built into KEDIT, and when KEDIT is loaded it automatically issues SET PARSER commands that use these language definitions to set up its default parsers. To distinguish these internal language definition files from actual disk files, KEDIT uses an asterisk as the first character of their names. For example, the command

```
SET PARSER C *C.KLD
```

tells KEDIT to use *C.KLD as the Language Definition File associated with the C parser. The asterisk in the name tells KEDIT to use the special file *C.KLD, which is built into KEDIT, and not to look for the file on disk.

The following parser definitions are automatically put into effect at KEDIT initialization:

Parser	File
BASIC	*BASIC.KLD
C	*C.KLD
COBOL	*COBOL.KLD
CSHARP	*CSHARP.KLD
FORTRAN	*FORTRAN.KLD
HTML	*HTML.KLD
JAVA	*JAVA.KLD
INI	*INI.KLD
KLD	*KLD.KLD
PASCAL	*PASCAL.KLD
REXX	*REXX.KLD
RESOURCE	*RESOURCE.KLD
XBASE	*XBASE.KLD
NULL	*NULL.KLD

Copies of these internal files are included in the SAMPLES subdirectory of the main KEDITW directory. If you modify one of these copies you should save it in a different location (normally the “KEDIT Macros” subdirectory of your Windows Documents

folder, which is sometimes known as the My Documents folder) and load it by issuing a SET PARSER command referring to the modified file.

If you want your own KLD file to be used in place of one of KEDIT's built-in KLD files, you can use a SET PARSER command that specifies the appropriate parser name along with your KLD file. For example

```
SET PARSER C NEWC.KLD
```

would use your NEWC.KLD file in place of the built-in *C.KLD file.

See also SET AUTOCOLOR, SET COLORING, SET ECOLOR, Chapter 8, "KEDIT Language Definition Files"

SET PATH

Format **[Set] PATH ON|OFF|*envvar*|*dirlist***

KEDIT default: *PATH;*INCLUDE;=

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET PATH controls the path search done by KEDIT when you use the KEDIT command or the GET command and specify a file's name and extension but no drive specifier or path and the file that you specify cannot be found in the current directory.

[Set] PATH ON
KEDIT looks for your file in each of the directories specified via the PATH environment variable.

[Set] PATH OFF
KEDIT will look for your file only in the current directory. In this special case, KEDIT will not do its normal path search for your file, and also won't do its usual search of the KEDIT Macros, KEDIT program directory, and USER and SAMPLES subdirectories.

[Set] PATH *envvar*
Tells KEDIT to do a path search for files, with the list of directories to search found in the environment variable *envvar*. Directories in the list are separated from each other by a semicolon, in the same format as is traditionally used with the PATH= environment variable.

SET PATH *dirlist*
Dirlist is a string with a list of semi-colon-delimited directories. It can also contain entries that begin with an asterisk, to indicate an indirect reference to an environ-

ment variable. ‘‘=’’ is also allowed, causing a search of the directory of the current file. For example,

```
SET PATH C:\TEMP;E:\SOURCE;*PATH;=
```

means search the C:\TEMP and E:\SOURCE directories, then the directories listed in the PATH environment variable, and then the directory of the current file.

In order for the value of PATH to affect KEDIT’s search for files added to the ring at the start of a KEDIT session, it needs to be set in an earlier KEDIT session and then saved to the Windows registry via the Options Save Settings dialog box. You can also specify PATH as a KEDIT initialization option, and if you do so its value will override the value in the registry.

When searching for files to be edited, KEDIT proceeds as follows: If a specific drive and/or directory is specified, KEDIT looks only there. Otherwise, KEDIT looks first in the current directory, then does a path search controlled by SET PATH. Finally, it looks in the “KEDIT Macros” subdirectory of your Windows Documents or My Documents folder, and in the directory from which KEDIT was loaded and in the USER and SAMPLES subdirectories of that directory. The search ends successfully as soon as KEDIT finds the file it is looking for; if the file cannot be located, KEDIT assumes you want to edit a new file with the specified name in the current directory.

The default value of *PATH;*INCLUDE;= tells KEDIT that its path search should involve looking in each of the directories listed in the PATH environment variable and the INCLUDE environment variable and then in the directory of the current file.

See also PATH initialization option, SET MACROPATH

SET PCOLOR

Format

[Set] PCOLOR *a foreground* [ON *background*]
[Set] PCOLOR *a* DEFAULT

KEDIT default: See the table below

Level: File

Dialog box: Options SET Command

Save Settings handling: Savable

Description

SET PCOLOR (“printer color’’) controls the colors used by KEDIT to print syntax-colored text when PRINTER WINDOWS and PRINTCOLORING ON are in effect and you are printing to a color printer.

The default SET PCOLOR values, given in the table below, are the same as the default values, controlled by SET ECOLOR, that KEDIT uses to display syntax-colored text

on your screen. But they can be controlled separately so that you have the option of using different color schemes on your printer and on your screen.

[Set] PCOLOR *a foreground* [ON *background*]
Specify the emphasis type involved (in the range A—Z or 1—9), followed by the foreground color to use and, optionally, the background color. If you do not specify a background color, a background of white is used. You can use an asterisk ("*") instead of a letter as the emphasis type to indicate that you want all emphasis types to use the same color. The foreground and background colors that you can choose from are the same as the colors used with the SET COLOR command.

[Set] PCOLOR *a* DEFAULT
If you have made changes to a printer color, you can tell KEDIT for Windows to switch back to using the default color. A table of KEDIT for Windows' default printer colors is given below.

Note

Even when syntax coloring is active, some of the text in your file may not be given an emphasis color. For example, most parsers do not give a special color to ordinary variables. KEDIT uses the color controlled by SET PCOLOR V, which is normally black on white, to print such text.

See also

SET ECOLOR, SET PRINTCOLORING

Examples

```
PCOLOR A RED
```

Emphasis type A (used for comments) is printed in red.

```
PCOLOR * DEFAULT
```

All printer colors are reset to their default values.

Letter	Default Printer Color	Language Element
A	dark green	comments
B	dark cyan	strings
C	dark red	numbers
D	blue	keywords
E	dark red	labels
F	dark red	preprocessor keywords
G	red	header lines
H	black	extra right paren, matchable keyword
I	blue	level 1 paren
J	blue	level 1 matchable keywords
K	dark red	level 1 matchable preprocessor keywords
L	dark green	level 2 paren, matchable keyword

Letter	Default Printer Color	Language Element
M	red	level 3 paren, matchable keyword
N	dark cyan	level 4 paren, matchable keyword
O	dark magenta	level 5 paren, matchable keyword
P	gray	level 6 paren, matchable keyword
Q	dark blue	level 7 paren, matchable keyword
R	magenta	level 8 or higher paren, matchable keyword
S	magenta	incomplete strings
T	blue	HTML markup tags
U	red	HTML character/entity references
V	black	unemphasised text (see note above)
W—Z	black	not currently used
1	red	alternate keyword color 1
2	dark blue	alternate keyword color 2
3	dark red	alternate keyword color 3
4	dark magenta	alternate keyword color 4
5	dark green	alternate keyword color 5
6	dark cyan	alternate keyword color 6
7	red	alternate keyword color 7
8	black	alternate keyword color 8
9	blue	alternate keyword color 9

SET POINT

Format `[Set] Point .name [OFF]`

KEDIT default: No named lines

Level: File

Dialog box: Actions Bookmark

Save Settings handling: Not savable

Description Use the SET POINT command to give a *name* to the focus line. The line can then be referred to in target specifications by that name. The *name* that you specify must be

preceded by a period (“.”). If *name* is already in use for some line of the current file, it is first removed from that line.

Use the OFF operand to remove a line name. If *name* is assigned to some line in the current file, the line name is removed. Otherwise, an error occurs.

A line can have more than one name, but no name can be used for more than one line in the current file. Line names are associated with lines of your file, but are not part of your file and are not saved when your file is written to disk. The line name remains with a line if you change the line or if you use the MOVE command to move the line elsewhere in the current file.

By default, Alt+1, Alt+2, and Alt+3 assign the names Bookmark1, Bookmark2, and Bookmark3, respectively, to the focus line, while Alt+4, Alt+5, and Alt+6 issue LOCATE commands that return you to these lines. The Set Bookmark button on the bottom toolbar also assigns BOOKMARK1 to the focus line, and the Go to Bookmark button returns to the line named Bookmark1.

You can also use the Actions Bookmark dialog box to work with line names.

Examples

```
POINT .ABC
```

Gives the name ABC to the focus line.

```
POINT .ABC OFF
```

Removes the name ABC from whatever line of the current file it is assigned to.

```
.XYZ
```

Makes the line named XYZ become the focus line.

SET PREFIX

Format

[Set] PREfix ON|OFF|Nulls [Left|Right]
[Set] PREfix Synonym *newname oldname*

KEDIT default: OFF LEFT

Level: View

Dialog box: Options SET Command (first form of command only)

Save Settings handling: Savable (first form of command only)

Description

The first form of SET PREFIX controls whether KEDIT’s prefix area is displayed and, if so, whether it is placed at the left or right side of the document window. All prefix commands are entered in the prefix area. The prefix area is normally displayed as five

equal signs next to each line in the file area. With NUMBER ON, the equal signs are not displayed; line numbers of each line in the file area appear instead.

With PREFIX NULLS, if NUMBER OFF is in effect KEDIT displays the prefix area as five blanks rather than as five equal signs. If NUMBER ON is in effect, PREFIX NULLS causes leading zeroes in line numbers to be displayed as blanks.

The second form of the SET PREFIX command allows you to define synonyms for prefix commands. This lets you refer to prefix commands using names of your own choosing. *Newname* is the new name by which you want to refer to the prefix command *oldname*. *Newname* is a string of one to five non-numeric characters.

For example,

```
PREFIX SYNONYM A F
PREFIX SYNONYM B P
```

will make "A" act like the "F" prefix command and "B" act like the "P" prefix command. (You could then think of "A" and "B" as "After" and "Before", instead of "F" and "P" as "Following" and "Preceding".)

You can have up to fifteen prefix synonyms in effect. Prefix synonyms are global, affecting all files in the ring.

See also

User's Guide Chapter 7, "The Prefix Area", LPREFIX, SET NUMBER, SET PREFIXWIDTH

Example

```
PREFIX ON RIGHT
```

Causes the prefix area to be displayed to the right of the file area.

Prefix command summary

Here is a summary of the available prefix commands, which are discussed in more detail in User's Guide Chapter 7, "The Prefix Area":

A	Add a blank line to your file
C	Indicate a line that is to be Copied
D	Delete a line from your file
F	Indicate the line Following which text will be moved or copied. (Used in conjunction with "C" or "M")
I	Insert a blank line into your file—same as "A"
L	Lowercase a line
M	Indicate a line that is to be Moved
P	Indicate the line Preceding which text will be moved or copied. (Used in conjunction with "C" or "M")
S	Show excluded lines
U	Uppercase a line
X	eXclude a line
/	Indicate a line that is to become the new current line

"	Indicate a line that is to be duplicated
<	Indicate a line that is to be shifted left 1 column
>	Indicate a line that is to be shifted right 1 column
SCALE	Specify that the scale line is to be displayed in this line
TABL	Specify that the tab line is to be displayed in this line
.name	Give a line a name
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* lines represented by shadow line
n*U** or **U*n	Uppercase *n* lines
S-*n*	Show last *n* lines represented by shadow line
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
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
>>	Placed in the prefix area of the first and last lines to be shifted right
""	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

SET PREFIXWIDTH

Format **`[Set] PREFIXWIDTH`** ***`n`***

KEDIT default::5

Level: File

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET PREFIXWIDTH controls the width of the prefix area, which can be a value in the range 5 through 9.

This option is useful if you use NUMBER ON in connection with PREFIX ON or PREFIX NULLS to display line numbers within the files you are editing and you want to be able to display line numbers larger than 99999, which is the largest that will fit in the default 5-character width of the prefix area.

See also SET PREFIX, SET NUMBER

SET PRINTCOLORING

Format **`[Set] PRINTCOLORing ON|OFF`**

KEDIT default: ON

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description With the default of PRINTCOLORING ON, syntax-colored text is printed in color when PRINTER WINDOWS is in effect and you print to a color printer.

With PRINTCOLORING OFF, KEDIT prints all text in black and white.

The colors KEDIT uses to print syntax-colored text are controlled by SET PCOLOR.

See also SET PCOLOR, SET PRINTER

SET PRINTER

Format

`[Set] PRINTER WINdows|`*`device`* `CLOSE|NOCLOSE FORM|NOFORM CONVert|NOCONVert`

KEDIT default: WINDOWS CLOSE FORM CONVERT

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description

SET PRINTER determines whether KEDIT uses your Windows printer driver for printer output or bypasses the Windows printer driver and sends output directly to your printer. SET PRINTER also controls some other details of KEDIT's printer handling, such as whether your printer is automatically closed after each print operation.

`[Set] PRINTER WINdows`

This is the default and is the recommended setting for most users. KEDIT uses your Windows printer driver to send output to your printer. If you have multiple Windows printers, you can use the File Print Setup dialog box (which is also accessible from the Setup button of the File Print dialog box) to choose the printer that KEDIT will use. The File Print dialog box's Font and Margins buttons let you control the font that KEDIT uses for printer output and the margins that KEDIT uses on the page.

`[Set] PRINTER `*`device`*

You can bypass the Windows printer and send output directly to LPT1:, LPT2:, LPT3:, COM1:, or COM2. 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 PRINTER WINDOWS is in effect.

KEDIT has no device-specific printer support. When you send output directly to a device, KEDIT accesses your printer through standard system file handles and has no special facilities for recovery from printer errors. For correct operation, your printer must be attached to the specified port, turned on, and properly initialized (possibly by using the system's MODE command before you enter KEDIT). KEDIT does not control the fonts or margins used on your printer; these are determined by defaults built into the printer or by printer escape codes that you imbed in your files.

`CLOSE|NOCLOSE`

When an application like KEDIT sends output through the Windows Print Manager or to a network printer, your output is not actually printed until the application ''closes'' the print file, letting the Print Manager know that all output is complete and ready to go. By default, SET PRINTER's CLOSE operand is in effect and KEDIT automatically closes the printer after each use of the PRINT command. This is almost always desirable behavior; the exception comes when you want to

use multiple PRINT commands for several smaller amounts of data that you would like to print together as a unit. This case can be handled by using SET PRINTER's NOCLOSE option.

FORM | NOFORM

The FORM|NOFORM option controls whether, when KEDIT is printing to a device (that is, when PRINTER LPT1:, etc. is in effect), KEDIT automatically sends a formfeed character, forcing a page eject, whenever the printer is closed. With the default of FORM, KEDIT does send a formfeed character. With NOFORM, formfeed characters are not automatically sent; if you are using a spooler or network printer, your system may automatically eject the page anyway, or you may need to use the Eject or Form Feed button on your printer.

When PRINTER WINDOWS is in effect, KEDIT always sends output to the printer a page at a time and the FORM|NOFORM option has no effect.

CONVERT | NOCONVERT

Files that you edit with KEDIT are normally displayed using either the ANSI or the OEM character set. Similarly, your printer is normally setup to print text in the ANSI character set or in the OEM character set. Incorrect output can be generated if the character set used by your file does not match the character set used by your printer.

With SET PRINTER's CONVERT option, which is the default, KEDIT attempts to compensate for this by automatically converting the data that it sends to your printer from OEM to ANSI if you are using an OEM character set for your file and an ANSI character set for your printer. Similarly, KEDIT converts from ANSI to OEM during printing if you are using an ANSI font for your file and an OEM font for your printer, or if you are using an ANSI font for your file and PRINTER WINDOWS is not in effect (since in most cases the default fonts built into printers use the OEM character set). Use the NOCONVERT option to prevent this conversion from taking place.

Notes

The printer is always closed, and a page eject is always done, when you print your file by using the File Print dialog box or the Print button on the toolbar, regardless of which of SET PRINTER's CLOSE|NOCLOSE or FORM|NOFORM options are in effect.

See also

PRINT, SET PRINTCOLORING

Examples

```
PRINTER WINDOWS CLOSE FORM CONVERT
```

This is the default; output is sent through the Windows Print Manager, with output automatically sent to the printer after each PRINT command.

```
PRINTER LPT1: NOCLOSE NOFORM CONVERT
```

This setting is closest to the default handling in the text mode version of KEDIT. Output is sent directly to LPT1: and may contain device-specific escape codes. The printer is not automatically closed, and formfeeds are not automatically sent after each PRINT command.

SET PRINTPROFILE

Format

`[Set] PRINTPROFile` ***`fileid`***

KEDIT default: PRINTPROFILE

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description

SET PRINTPROFILE lets you change the name of the profile executed when the Windows Explorer invokes KEDIT to print an associated file.

You can use the Windows Explorer to specify that KEDIT will be the application used to print files with certain extensions. Once a file extension has been associated with KEDIT, you can select a file with that extension in the Windows Explorer and choose Print from the button 2 context menu to have the file printed by KEDIT. You can also have KEDIT print the file by dragging the file within the Windows Explorer to your default printer icon.

When the Windows Explorer uses KEDIT to print a file, KEDIT is invoked with a command like

`KEDITW32 /P` ***`fileid`***

When KEDIT sees /P as its first parameter it takes this a signal to run the profile specified via the SET PRINTPROFILE option, instead of your normal profile. This special profile should contain commands to print your file and then exit from KEDIT. An appropriate default profile, PRINTPROFILE, is built into KEDIT, so most KEDIT users will not need to change the value of the PRINTPROFILE option.

SET QUICKFIND

Format

`[Set] QUICKFIND Respect|Ignore Word|NOWord Regexp|NORegexp` ***`string`***

KEDIT default: Preserved from previous editing session

Level: Global

Dialog box: None

Save Settings handling: Automatically saved at the end of a session

Description

SET QUICKFIND is specialized command, most often used from within macros, that sets the value of the search string in the Quick Find toolbar item.

Once the Quick Find string has been set, you can search for it by activating the Quick Find toolbar item and pressing Enter or by clicking on the Find Next toolbar button. Changes to the Quick Find string are also reflected in the default string displayed when you use the Edit Find or Edit Replace dialog boxes. KEDIT automatically updates the Quick Find string whenever you use Edit Find or Edit Replace or issue a LOCATE or CLOCATE command from the command line.

The operands for SET QUICKFIND are RESPECT|IGNORE, which determines whether case will be respected in a search involving the Quick Find string, WORD|NOWORD, which determines whether a search will be limited to whole word boundaries, REGEXP|NOREGEXP, which determines whether the Quick Find string will be treated as a regular expression, and finally the string itself. For example, to set the Quick Find string to ‘‘yesterday’’, and to specify that a search involving this string will ignore case, use word boundaries, and not be treated as a regular expression, you would use the command

```
SET QUICKFIND IGNORE WORD NOREGEXP yesterday
```

The *string* specified with the SET QUICKFIND command cannot be more than 100 characters long.

SET RANGE

Format

`[Set] RANge` *`target1 target2`*

KEDIT default: The entire file

Level: View

Dialog box: Options SET Command

Save Settings handling: Not savable

Description

Use the SET RANGE command to specify the range of lines in your file within which KEDIT commands will operate. *Target1* specifies the first line of the range and *target2* specifies the last line of the range.

Normally, KEDIT commands range over your entire file. For example, if you are editing a 1000-line file, commands can normally affect all lines of your file, from line 1 through line 1000. If you want to spend some time working with only a subset of your file, for example lines 100 through 200, you can use the SET RANGE command to tell KEDIT to limit its operations to that portion of your file:

```
SET RANGE :100 :200
```

KEDIT will then show you only lines 100 through 200 of your file, and any KEDIT commands that you issue will operate only within that range of lines. Line 100 will act very much like the first line of your file, and line 200 will act like the last line of your file. For example, even though there are 1000 lines in your file, the BOTTOM command will go to line 200 of your file. Above line 100, KEDIT displays a top-of-range line, similar to the normal top-of-file line, and lines 1 through 99 of your file are not shown. Below line 200, KEDIT displays an end-of-range line, similar to the normal end-of-file line, and lines 201 through 1000 of your file are not shown.

The FILE and SAVE commands will always write your entire file to disk, regardless of the range in effect. All other KEDIT commands act within the current range, and ignore lines outside of the range.

When you have finished working with a particular range of lines within your file, you can use the SET RANGE command to once again include your entire file in the current range. In our example of a 1000-line file,

```
SET RANGE :1 :1000
```

would allow commands to range over your entire file. A more general way of doing this, which would be independent of the size of your file, would be

```
SET RANGE -* *
```

Examples

```
SET RANGE :10 *
```

Sets the range to extend from line 10 of your file through the end of your file.

SET RANGE -5 5

The line five lines above the focus line becomes the first line of the range, and the line 5 lines below the focus line becomes the last line of the range.

SET RANGE BLOCK

This is handled as a special case. It sets the range to extend from the first to the last lines of the currently defined block.

SET RECENTFILES

Format **[Set] RECENTFiles *n***

KEDIT default: 9

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description At the bottom of the File menu is a list of files that you have recently edited. You can select a file from this list to re-edit it without going into the File Open dialog box. RECENTFILES determines the maximum number of files that can appear on this list. By default, up to 9 recently-used files will appear. RECENTFILES can have values in the range 0 to 25.

SET RECFM

Format **[Set] RECFm Fixed|Varying**

KEDIT default: VARYING

Level: File

Dialog box: Options SET Command

Save Settings handling: Not savable

Description SET RECFM, included for use in specialized situations, controls whether KEDIT writes varying- or fixed-length lines when files are written to disk by the FILE, SAVE, and PUT commands, by the autosave facility, and by File Save and related menu items. When KEDIT writes a file with the default setting of RECFM VARYING, it writes out all characters through the last nonblank character in each line. (Depending on the setting of TRAILING, trailing blanks may also be written.) With RECFM FIXED, KEDIT will write out all lines at the same length (as set by the SET LRECL command).

Lines that are less than the current logical record length are padded with blanks. Lines that are longer than the current logical record length are truncated.

Each line that KEDIT writes with RECFM FIXED normally takes up LRECL+2 bytes on disk, since an end-of-line sequence, controlled by SET EOLOUT and defaulting to a carriage return-linefeed pair, is written after each line. A 10-line file, for example, with LRECL 80, RECFM FIXED, EOLOUT CRLF, and EOFOUT EOL would therefore take 820 bytes when written to disk.

See also User's Guide Chapter 12, "File Processing", SET LRECL

SET REPROFILE

Format **`[Set] REPROFile ON|OFF`**

KEDIT default: OFF

Level: Global

Dialog box: Options SET Command

Save Settings handling: Not savable

Description Use SET REPROFILE to determine whether your profile should be re-executed whenever you start to edit an additional file.

When you first invoke KEDIT, KEDIT assigns values to all SET options for the file you will edit. These are based on KEDIT's built-in default settings, as modified by any changes you have made via Options Save Settings. KEDIT then executes your profile, which may change some of these options, and reads in your file. If, during your session, you edit additional files and REPROFILE OFF is in effect, KEDIT copies over the values of most SET options from the current file to the new file, as discussed in Section 2.4, "Editing Additional Files". Then KEDIT loads in the new file. Your profile is not re-executed.

With REPROFILE ON, when you edit an additional file, the SET options for the new file are not copied from the current file. Instead, they are set to their default values, based on KEDIT's built-in defaults, as modified by your saved settings. Then your profile is re-executed, and finally your file is read in.

Re-executing your profile whenever you begin to edit a new file may slow things down slightly, but is recommended so that your profile can contain commands that adjust KEDIT's settings depending on the extension of the file being edited. With REPROFILE ON, your profile can set things up properly for each new file added to the ring. Using the INITIAL() function, your profile can test if it is being executed for the first time or is being re-executed, and can avoid redefining keys, etc., unnecessarily.

To determine the name of the profile to execute when a new file is loaded with REPROFILE in effect, KEDIT uses the value of the PROFILE option specified on the command line involved and, if the PROFILE option is not specified, uses the value of DEFPROFILE. The default value for DEFPROFILE is WINPROF.KEX. You can use SET DEFPROFILE or the DEFPROFILE initialization option to specify a different default profile.

SET REGSAVE

Format **`[Set] REGSAVE STATE|NOSTATE [HISTory|NOHISTory]`**

KEDIT default: STATE HISTORY

Level: Global

Dialog box: None

Save Settings handling: Not savable

Description SET REGSAVE determines which information KEDIT updates in its section of the Windows registry upon termination. Information saved in the registry at the end of a KEDIT session is used the next time that you run KEDIT to setup the File menu's list of recently-edited files, the initial position of KEDIT's frame window, etc.

The STATE|NOSTATE value controls whether KEDIT updates certain status information within the registry at the end of a session. This includes information about the state (maximized or non-maximized) of KEDIT's frame and document windows, the size and position of the frame window, the screen and printer fonts that you are using, etc. With REGSAVE STATE in effect, the default, this information is updated at the end of the current session. With REGSAVE NOSTATE, it is not.

The HISTORY|NOHISTORY value controls whether, at the end of the current session, KEDIT updates the lists of recently-issued commands, recently-edited files, recently-used Edit Find search strings, etc. that it maintains within the Windows registry. With the default value of HISTORY, the lists are updated; with NOHISTORY they are not.

Notes

STATE and HISTORY information that is not updated in the Windows registry at the end of a KEDIT session is not deleted from the registry, but is instead left unchanged. So if, for example, you put REGSAVE NOSTATE NOHISTORY into effect and then exit KEDIT, information that was saved in the registry at the end of your last KEDIT session will remain in the registry, and will be used the next time you run KEDIT. You can, however, use the REGUTIL command to remove this information from the registry.

KEDIT also uses the Windows registry to save information about SET command options whose values you have changed from their defaults. This information is

not updated automatically at KEDIT termination, but is instead updated via the Options Save Settings dialog box.

If you start KEDIT with the NOREG or NOINI options, REGSAVE NOSTATE NOHISTORY is the default instead of the usual REGSAVE STATE HISTORY. The NOREG (or NOINI) option tells KEDIT not to load any information from the Windows registry but to instead start with default window positions and with empty history lists.

For historical reasons SET INISAVE, which does the same thing as SET REGSAVE, is also available.

See also NOREG initialization option, REGUTIL

SET RESERVED

Format

```
[Set] RESERved line [color] text
[Set] RESERved line OFF
```

KEDIT default: No reserved lines

Level: File

Dialog box: None

Save Settings handling: Not savable

Description

With SET RESERVED, you can reserve lines of the document window for special *text* that you want KEDIT to display in the current file's window. You might, for example, want to display a summary of your function key definitions near the bottom of the document window.

The *line* to be reserved can be specified relative to the top of the document window, the middle of the document window, or the bottom of the document window; see SET CURLINE for a discussion of how to do this.

You can also specify the *color* to be used when displaying the reserved line; see SET COLOR for a discussion of how colors can be specified. If you don't specify a color, KEDIT uses the COLOR IDLINE value to determine the color of the reserved line.

The alternate form of SET RESERVED lets you turn off display of a reserved line.

Examples

```
SET RESERVE -1 WHITE ON BLUE F1=Help F2=Add F3=Quit F8=Dup
```

The specified text (starting with "F1=") is displayed, using white text on a blue background, in the last line of the document window.

```
SET RESERVE -1 OFF
```

This removes the reserved line from the display.

SET RIGHTCTRL

Format **`[Set] RIGHTCTRL ON|OFF`**

KEDIT default: OFF

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET RIGHTCTRL lets you use the right Ctrl key found on most keyboards as an Enter key. (The Enter key on 3270 displays, which many KEDIT users are familiar with, is located in the same position as the right Ctrl key is on most PC keyboards.)

With RIGHTCTRL ON, whenever you press the right Ctrl key, KEDIT acts as if you had pressed the Enter key on the numeric key pad, whose KEDIT keyname is NUMENTER. The default definitions for both ENTER and NUMENTER are the same, so pressing the right Ctrl key and pressing the ENTER key do the same thing by default. If you use the right Ctrl key as an Enter key, you may want to redefine the ''real'' ENTER key to serve some other purpose.

SET SCALE

Format **`[Set] SCALe ON|OFF [`*`line`*`]`**

KEDIT default: OFF M+1

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description When SCALE ON is in effect, KEDIT displays a ''scale line'' in the file area of the document window. The scale line, which visibly displays indications of what column text is in, appears only on your screen and is not a part of your file. The scale line helps you see what column things are in by indicating where every fifth column is located. Also, the left and right zone columns are indicated by ''<'' and ''>'' characters, the left and right margins by ''['' and '']'' characters, the paragraph indent column by a paragraph symbol, the truncation column by a ''T'', and the column pointer by a ''|''. If the tab

line is being displayed on the same line as the scale line, the current tab column settings are also indicated on the scale line. (If the same column is serving several purposes, for example as both the left zone column and the left margin column, only one of the above indicators is displayed.)

The scale line normally appears just below the middle of the document window, but you can specify what *line* of the document window it should appear on. The *line* can be specified relative to the top of the document window, the middle of the document window, or the bottom of the document window; see SET CURLINE for a discussion of how to do this.

See also SET TABLINE

Examples

```
SCALE ON M-1
```

Displays the scale line one line above the middle of the document window.

```
SCALE ON
```

Displays the scale line in the default location (or in the location last set by issuing "SCALE ON *line*").

SET SCOPE

Format `[Set] SCOPE All|Display`

KEDIT default: DISPLAY

Level: View

Dialog box: Options SET Command

Save Settings handling: Not savable

Description Using KEDIT's selective line editing facilities, you can exclude certain portions of your file from your display. With SCOPE DISPLAY, the default, lines that are excluded from display are also excluded from processing by most KEDIT commands; these commands act as if excluded lines were not present in your file. With SCOPE ALL, KEDIT commands operate on all lines of your file, even though some of them may be excluded from display. SCOPE ALL is rarely used except in specialized situations within macros. In most situations the default of SCOPE DISPLAY is appropriate.

While most commands avoid excluded lines when SCOPE DISPLAY is in effect, there are a few exceptions. The FILE and SAVE commands and menu items that write your file to disk always operate on your entire file, writing even excluded lines to disk. The SORT command and the Actions Sort dialog box sort all lines in the target area of your file, even though some of these lines may be excluded lines.

When SCOPE ALL is in effect, the current line is always displayed, regardless of its selection level.

The ALL command with a target operand automatically puts SCOPE DISPLAY into effect whenever it is successfully executed, as does the Edit Selective Editing dialog box when it successfully matches a string that you specify.

See also User's Guide Chapter 8, "Selective Line Editing and Highlighting", ALL, SET DISPLAY, SET SELECT, SET SHADOW

Examples

```
ALL /Lincoln/
DELETE 5
```

The ALL command selects all lines of your file containing ''Lincoln'', makes the first such line become the focus line, and puts SCOPE DISPLAY into effect. Since SCOPE DISPLAY is in effect, the DELETE command operates only on lines that are selected, so it deletes the first five lines containing ''Lincoln''.

```
ALL /Lincoln/
SET SCOPE ALL
DELETE 5
```

After lines containing ''Lincoln'' are selected, the SET SCOPE ALL command tells KEDIT that commands should operate on all lines of your file. The DELETE command therefore deletes the focus line and the four lines below it, for a total of five lines, regardless of whether they contain ''Lincoln''.

SET SCREEN

Format

[Set] SCReen *n* [Horizontal|Vertical]
[Set] SCReen *m* Split

KEDIT default: 1

Level: Global

Dialog box: None

Save Settings handling: Not savable

Description SET SCREEN lets you specify the number of document windows that you want to have displayed within KEDIT's frame window, and it arranges the windows in a tiled fashion so that they completely fill the frame window.

SET SCREEN is provided mainly for compatibility with the text mode version of KEDIT, which only supports tiled windows. SET SCREEN works only when OFPW (''One File Per Window'') OFF is in effect, since OFPW OFF makes KEDIT for Windows use the same rules for placing files in document windows as text mode KEDIT

does. The Window Tile and Window Cascade menu items, and the Window Arrange dialog box, provide useful alternative methods for arranging your document windows.

In text mode KEDIT, document windows are always tiled and the SET SCREEN command provides the only method for changing how many document windows are displayed and how they are arranged. With KEDIT for Windows, you can easily create new document windows (for example, with Window New), you can use the mouse to move and resize windows, So the window arrangement that you specify with SET SCREEN may not remain permanently in effect in the way that it does in text mode KEDIT.

[Set] SCReen *n* [Horizontal|Vertical]
With the first form of the SET SCREEN command, you specify the number of document windows to display within the frame window. If HORIZONTAL is specified (or if nothing is specified, since HORIZONTAL is the default) the frame window will be split horizontally into *n* document windows. If you specify VERTICAL, the frame window is split vertically into *n* document windows.

[Set] SCReen *m* Split
With the second form of the SET SCREEN command, you specify the number of horizontal areas that are to appear on your screen. SPLIT indicates that each of these horizontal areas is to be split down the middle into two windows, giving you a total of 2*m* windows.

See also

User's Guide Section 3.5, "Editing Multiple Files", SET OFPW

Examples

```
SCREEN 2
```

This gives you two document windows, one in the upper half of the frame window and the other in the lower half.

```
SCREEN 2 V
```

This gives you two frame windows, one in the left half of the frame window and the other in the right half.

```
SCREEN 3 S
```

This divides the frame window into three horizontal areas, each split vertically, giving you a total of six document windows.

SET SCROLLBAR

Format **`[Set] SCROLLbar ON|OFF [Vertical|Horizontal|BOTH]`**

KEDIT default: ON BOTH

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET SCROLLBAR controls the scroll bars KEDIT normally displays on your document windows.

The first operand controls whether scroll bars are displayed at all. The second operand lets you display only a VERTICAL scroll bar, only a HORIZONTAL scroll bar, or BOTH vertical and horizontal scroll bars.

See also User's Guide Chapter 4, "Keyboard and Mouse"

SET SELECT

Format **`[Set] SELect [+|-]`*`n`*` [`*`target`*`]`**

KEDIT default: All lines have selection level 0

Level: File

Dialog box: Options SET Command (for focus line only)

Save Settings handling: Not savable

Description Use the SET SELECT command to set the selection level of lines in your file.

Each line of your file has a number, called its selection level, associated with it. Selection levels can range from 0 to 255. You can set the selection level of lines in your file by using the SET SELECT command. (The ALL command, the Edit Selective Editing dialog box, and the X and S prefix commands also manipulate the selection levels of lines in your file.) Using the SET DISPLAY command, you specify the range of selection levels of lines you want selected for display. Lines whose selection levels fall within the range specified by the SET DISPLAY command are selected for display. Lines whose selection levels are outside this range are excluded from your display and, if SCOPE DISPLAY is in effect, are excluded from processing by most KEDIT commands.

SET SELECT affects selection levels of lines in the specified *target* area. If no *target* is specified, SET SELECT sets the selection level of the focus line.

[Set] SELect *n target*
Sets the selection level of lines in the *target* area to *n*.

[Set] SELect +*n target*
Adds *n* to the selection level of lines in the *target* area. (If the result would be greater than 255, the selection level is set to 255.)

[Set] SELect -*n target*
Subtracts *n* from the selection level of lines in the *target* area. (If the result would be less than 0, the selection level is set to 0.)

Note that, like most other KEDIT commands, SET SELECT operates only on lines within the current SCOPE. If SCOPE DISPLAY is in effect, SET SELECT will not operate on excluded lines.

All lines in your file initially have a selection level of 0. When a new line is added to your file, it gets a selection level equal to the *n1* value of the current DISPLAY setting (DISPLAY *n1 n2*). When a line is copied, duplicated, or split, the resulting new line is given the same selection level as the line which is copied, duplicated, or split.

See also
User's Guide Chapter 8, "Selective Line Editing and Highlighting", ALL, SET DISPLAY, SET SCOPE, SET SHADOW

Examples

```
SET SELECT +1
```

KEDIT adds 1 to the selection level of the focus line.

```
SCOPE ALL
SELECT 2 ALL
```

KEDIT gives all lines of your file a selection level of 2.

```
SCOPE DISPLAY
SELECT 2 ALL
```

KEDIT gives all selected lines of your file a selection level of 2.

SET SHADOW

Format **[Set] SHADow ON|OFF**

KEDIT default: ON

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET SHADOW controls how excluded lines are represented on your display. With SHADOW ON, the default, KEDIT displays a shadow line to represent each group of one or more excluded lines. The shadow line lets you see where excluded lines occur in your file, and indicates how many lines are excluded. With SHADOW OFF, KEDIT does not display shadow lines; excluded lines are completely omitted from your display.

Excluded lines most often result from use of the ALL command, the Edit Selective Editing dialog box, or of the X prefix command.

See also User's Guide Chapter 8, "Selective Line Editing and Highlighting", ALL, SET DISPLAY, SET SCOPE, SET SELECT

SET SHARING

Format `[Set] SHARING DENYWRITE|DENYNONE [DENYWRITE|DENYREADWRITE]`

KEDIT default: DENYWRITE DENYREADWRITE

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET SHARING controls the sharing modes used by KEDIT when it reads a file into memory and when it locks a file.

The first operand controls the sharing mode used when KEDIT reads a file, which happens when you begin to edit a file without file locking in effect, when you load a macro or .KML file, and when you use the GET command.

`DENYWRITE`
This is the default sharing mode used when reading files. It means that the attempted read will fail if any other program has write access to the file and that no other program can begin to write to the file while KEDIT is reading it.

`DENYNONE`
KEDIT will attempt to read the file regardless of whether other programs have write access to it. The read will still fail if some other process has exclusive access to the file. Note that with DENYNONE there is a chance that the file may not be read in properly, since it is possible that another program might change the file while KEDIT is in the process of reading it.

The second operand controls the sharing mode used when KEDIT locks a file. KEDIT locks a file when you begin to edit a file and LOCKING ON is in effect or you specify the LOCK option, and when you use the LOCK command to lock a file that is already in the ring.

DENYREADWRITE
This is the default sharing mode used when locking files. It means that KEDIT will have exclusive access to the file while it is locked. If any other programs are using the file when the lock is requested, the request will fail. Once the lock has been obtained, no other programs will be able to access the file.

DENYWRITE
KEDIT will be able to lock the file as long as no other program is currently writing to the file or preventing write access to the file. While the file is locked, no other program will be able to begin writing to the file or open the file in deny write mode. Other programs will be able to read from the file. Note that with DENYWRITE there is a chance that the file may not be read properly by these other programs, since they may try to read from the file while you are saving changes to disk.

There is no operand to control the sharing modes used when KEDIT writes a file that you are editing to disk when file locking is not in effect. In this case, KEDIT always requires exclusive access to the file.

See also User's Guide Chapter 12, "File Processing"

SET STATUSLINE

Format `[Set] STATUSLine ON|OFF`

KEDIT default: ON

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET STATUSLINE lets you control whether or not KEDIT displays a line of status information at the bottom of your frame window.

With the default setting of STATUSLINE ON, KEDIT displays a status line at the bottom of your frame window. The status line is used to display your position in the current file, the number of alterations and undoable changes made to the file, the size of the file, the number of files you are editing, the number of document windows you are using, whether you are in Insert Mode or Overtype Mode, and whether the current file is locked.

Other information that can be displayed on the status line is optional: With CLOCK ON, the status line gives the time of day. With HEXDISPLAY ON, the status line displays the character code, in hexadecimal and in decimal, for the character at the cursor position.

See also SET CLOCK, SET HEXDISPLAY

SET STAY

Format `[Set] STAY ON|OFF`

KEDIT default: ON

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description The STAY option controls the positioning of the focus line after you attempt to locate a string target or string column target or after you use one of the commands listed below.

With STAY OFF (and assuming WRAP is OFF), an unsuccessful LOCATE, CLOCATE, TFIND, or Edit Find operation, makes the end-of-file line become the new focus line. (The top-of-file line is the new focus line after an unsuccessful backward search.)

With STAY ON, the focus line is unchanged after an unsuccessful LOCATE, CLOCATE, TFIND, or Edit Find operation.

With STAY OFF, the last line scanned or affected by the ALTER, ANSITOOEM, CENTER, CHANGE, COMPRESS, COUNT, EXPAND, FIND, FINDUP, LEFTADJUST, LOWERCASE, NFIND, NFINDUP, OEMTOANSI, PRINT, PUT, RIGHTADJUST, SET LINEFLAG, SET SELECT, SHIFT, UPPERCASE, and SORT commands becomes the new focus line after the command has completed. This is also true for the Edit Replace and Actions Sort dialog boxes.

With STAY ON, the location of the focus line is unchanged after these commands finish.

SET STREAM

Format `[Set] STReam ON|OFF`

KEDIT default: ON

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description The setting of STREAM affects the search for string column targets when you use the CLOCATE or CDELETE commands. With STREAM OFF, a search for a string column target is limited to the focus line. With STREAM ON, all lines through the end of the file (or the whole file, if you have set WRAP ON) are searched.

Note that the setting of STREAM only affects the operation of the CLOCATE and CDELETE commands, which are the only two commands whose operands are column targets. The LOCATE command and the Edit Find dialog box are never limited to the focus line and are not affected by SET STREAM.

SET SYNONYM

Format

```
[Set] SYNonym ON|OFF
[Set] SYNonym [LINEND char] newname [n] definition
```

KEDIT default: ON

Level: View

Dialog box: Options SET Command (SYNONYM ON|OFF only)

Save Settings handling: Savable (SYNONYM ON|OFF only)

Description

Use the SET SYNONYM command to control KEDIT's synonym facility.

The first form of the SET SYNONYM command controls whether KEDIT does synonym processing at all. With SYNONYM ON, the default, synonym processing (as described below) is done for commands issued from the command line (and for commands issued from macros via the SYNEX command). With SYNONYM OFF, no synonym processing is done and KEDIT acts as if no synonyms have been defined. Note that synonym processing is normally bypassed for commands issued from macros, unless you use the SYNEX command.

The second, more frequently used, form of SET SYNONYM lets you change the names and definitions of KEDIT's commands. After an optional LINEND character specification (discussed below, and usually not necessary), you give the name which you want a command to be known by. You can follow this with the number of characters that KEDIT will accept as the minimal truncation for this name. Then you tell KEDIT what the command with this name should do.

For example, suppose that you wish KEDIT's LOCATE command were called SEARCH. You would issue the command

```
SYNONYM SEARCH LOCATE
```

Then, instead of

```
LOCATE /abc/
```

you could enter

```
SEARCH /abc/
```

Note that KEDIT does only one level of synonym processing. When KEDIT sees that SEARCH is a synonym for LOCATE, it executes the LOCATE command without looking to see if LOCATE is itself a synonym for some other command.

After specifying the *newname* that you want for a command, you can specify the number *n* of characters that will be accepted as its minimal truncation. In the example above, SEARCH would have to be spelled out in full. With

```
SYNONYM SEARCH 3 LOCATE
```

SEARCH could be given as "SEA", "SEAR", "SEARC", or "SEARCH".

KEDIT's synonym processing can convert what you enter as one command into more than one command. For example, suppose you want to have a command called FIRST that finds the first occurrence of a string in your file. You need to specify a LINEND character to tell KEDIT where each command in your synonym definition ends. (This is different from the character controlled by SET LINEND, and is in effect only during the affected SET SYNONYM command.) The FIRST command will go to the top of the file and then look for the specified string:

```
SYNONYM LINEND + FIRST TOP + LOCATE
```

Here, "+" is set as the LINEND character. Then two commands ("TOP" and "LOCATE") are issued, with the "+" used to indicate the separation between the two commands. You could then enter

```
FIRST /X/
```

to find the first occurrence of "X" in your file. Any operands on the command that you enter are placed by KEDIT at the end of the last command in the synonym definition, so that "/X/" is taken as an operand for the LOCATE command and not of the TOP command.

If your synonym definition ends with a special character, any operands you give are placed immediately after the special character. If it ends with an alphabetic or numeric character, KEDIT adds a blank after the synonym definition and then appends your operands.

The *newname* given in your synonym definition can be a single special character or it can be a string of one to ten alphabetic characters. You can define up to 80 synonyms. The synonyms are global, affecting all files in the ring.

```
Query SYNonym *
```

will display all the SYNONYM definitions currently in effect.

You can cause KEDIT to bypass synonym processing for a command issued from the command line by preceding it with "COMMAND". For example, suppose that you have made COPY a synonym for some other command, but that you now want to execute the "real" KEDIT COPY command, bypassing the synonym.

```
COMMAND COPY :12
```

causes KEDIT to process the command COPY :12, without first looking for synonyms for COPY. On the other hand, synonym processing is bypassed for commands issued from within a macro, so ‘‘COPY :12’’ issued from a macro would execute KEDIT’s COPY command, and not your synonym. ‘‘SYNEX COPY :12’’ would cause KEDIT to look for synonyms of COPY.

See also COMMAND, SYNEX

SET TABLINE

Format **`[Set] TABLine ON|OFF [`*`line`*`]`**

KEDIT default: OFF -2

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description When TABLINE is ON, a ‘‘tab line’’ is displayed in the specified *line* of the document window. This tab line indicates the position of each tab column (controlled via the SET TABS command) by showing a ‘‘T’’ in the appropriate column.

The tab line normally appears near the bottom of the file area, but you can specify what *line* of the document window it should appear on. The *line* can be specified relative to the top of the document window, the middle of the document window, or the bottom of the document window; see SET CURLINE for a discussion of how to do this.

If the tab line is set to appear on the same line as the scale line, then the tab information appears on the scale line, intermixed with the information normally displayed on the scale line.

See also SET SCALE, SET TABS

SET TABS

Format

```
[Set] TABs n1 [n2 n3 ...]
[Set] TABs INCR n
[Set] TABs n1 [n2 ...] INCR n
```

KEDIT default: INCR 8

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description

Use SET TABS to set your tab columns. Whenever you press the Tab key (or, if PREFIX ON is in effect, the F4 key), KEDIT moves the cursor forward to the next tab column. Whenever you backtab (which, unless PREFIX ON is in effect, is normally done by pressing Shift+Tab), the cursor moves backward to the previous tab column.

Initially, tab columns are set up every eight columns, starting at column 1. Issue the SET TABS command if you want to change this.

With the first form of the SET TABS command, you enter a list of up to 32 tab columns. The numbers you give must be in ascending order and can range from 1 to the value of the WIDTH initialization option.

With the second form of the SET TABS command, you specify an increment *n* and KEDIT sets tabs in column 1 and every *n* columns thereafter.

The third form of the SET TABS command lets you give a list of specific tab columns and then specify that tabs will be set every *n* columns thereafter.

Using SET TABS is similar to defining tab stops on a typewriter. It can be very useful if you need to enter text in specific columns. For example, you may be entering tables where entries must be lined up in specific columns, or you may be using a programming language that requires parts of your program to be in specific columns.

Note that KEDIT does not actually enter a tab character (character code 9) into your file when you press the Tab key. KEDIT merely moves the cursor to the next tab column. Your TABS setting also has no effect on how tab characters are processed by KEDIT when it reads files from disk or writes them out to disk; this processing is controlled by SET TABSIN and SET TABSOUT.

See also

COMPRESS, EXPAND, SET TABSIN, SET TABSOUT, SET TABLINE

Example

```
TABS 1 10 16 30 40 INCR 5
```

Columns 1, 10, 16, 30, and 40, and then columns 45, 50, 55, 60, etc., are set up as tab columns.

SET TABSAVE

Format **`[Set] TABSAVE ON|OFF`**

KEDIT default: OFF

Level: File

Dialog box: Options SET Command

Save Settings handling: Not savable

Description

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.

To use TABSAVE, you need to do the following:

Put TABSIN ON into effect (or use TABSIN ON *n*, if you assume tab stops at other than the every-8-columns default)

Put TABSAVE ON into effect.

Put TABSOUT ON into effect (or use TABSOUT ON *n*) if you want tab compression on newly added lines or lines whose contents change during the editing session. If you want all new/changed lines to be saved without tab compression, you should use TABSOUT OFF *n* (where *n* matches the TABSIN ON *n* setting).

Things will work properly only if all three of these (including the TABSOUT ON|OFF *n* setting) are in effect when you load the file involved. Furthermore, you will probably only want to put TABSAVE ON into effect for certain file extensions, because it makes files take twice as much room in memory. You would therefore want something like the following in your WINPROF.KEX file:

```
'reprofile on'
if fext.1() = 'C' | fext.1() = 'H' then do
   'tabsin on'
   'tabsout on'
   'tabsave on'
end
```

KEDIT will then keep track of the original contents of all lines of your file. When you save the file, KEDIT will compare the current contents and the original contents of each line of the file, ignoring changes due to tabs versus space characters. If the contents of a line have not changed, the original version of the line will be written back to disk preserving its original tab characters. If the contents of a line have changed, the updated version of the line will be written back to disk, with its tab compression determined by the TABSOUT setting.

One point to be aware of is that to preserve the tabs, the TABSAVE handling discussed above always needs to be in effect for the files in question – if you manage even once to edit and save the file without the TABSAVE handling, the tabs will of course get changed.

See also SET TABSIN, SET TABSOUT

SET TABSIN

Format **`[Set] TABSIn ON|OFF|TABQUOTE [`*`n`*`]`**

KEDIT default: OFF 8

Level: File

Dialog box: Options SET Command

Save Settings handling: Savable

Description To save space on disk, some text editors compress strings of blanks into tab characters (character code 9) when writing files to disk. (If you set TABSOUT ON, KEDIT will also perform this compression.) When TABSIN ON is in effect, KEDIT automatically handles this situation, expanding to strings of blanks any tabs that it finds when it reads a file in.

When TABSIN ON causes tab expansion, KEDIT normally assumes tab stops every 8 columns (columns 1, 9, 17, etc.). These are standard tab settings used by many programs. The DOS TYPE command, for example, processes tab characters using these tab stops. You can, however, set the tab increment used by TABSIN to some number *n* other than 8 if you need to process files stored with different tab stops. Note that the tab columns used with TABSIN processing are independent of the tab columns used while you are editing your file (which are controlled by SET TABS) or when you write a file out (when SET TABSOUT is relevant).

TABSIN OFF is the default. You would want to leave TABSIN OFF if your file contains tab characters that you want to leave intact and unexpanded. To take effect when your file is read in, TABSIN ON must be one of your saved settings or must be issued from your profile.

With TABSIN ON, KEDIT expands all tab characters, including those found after the first single quote or double quote character on a line. With TABSIN TABQUOTE, KEDIT expands all tab characters in a line up to the first single or double quote, but leaves tabs after the first quote on a line unexpanded.

SET TABSIN also affects how KEDIT expands tab characters in text from other applications pasted into KEDIT via the CLIPBOARD. If TABSIN ON or TABSIN TABQUOTE (which in this situation is treated like TABSIN ON) is in effect, tabs are expanded according to TABSIN's tab stops. With TABSIN OFF, tabs are expanded according to the current SET TABS setting.

See also User's Guide Chapter 12, "File Processing", CLIPBOARD, COMPRESS, EXPAND, SET TABSAVE, SET TABSOUT

SET TABSOUT

Format **`[Set] TABSOut ON|OFF [`*`n`*`]`**

KEDIT default: OFF 8

Level: File

Dialog box: Options SET Command

Save Settings handling: Savable

Description If you set TABSOUT ON, KEDIT will compress strings of blanks into tab characters (character code 9) when it writes your file to disk. Your file may then take up less disk space. If TABSIN ON is in effect when such a file is read back in by KEDIT, the tab characters will be expanded back into blanks.

When TABSOUT ON causes blank compression, KEDIT normally assumes tab stops every eight columns (columns 1, 9, 17, etc.). These are standard tab settings used by many programs. The DOS TYPE command, for example, processes tab characters using these tab stops. You can, however, set the tab increment used by TABSOUT to some number *n* other than 8 if you need to store files with different tab stops. Note that the tab columns used with TABSOUT processing are independent of the tab columns used when your file is read in (SET TABSIN controls this) and while you are editing your file (where SET TABS is relevant).

You may want to leave the default value of TABSOUT OFF in effect because many programs do not properly handle files with tab compression, and the disk space saved may not be worth the extra confusion.

Even with TABSOUT ON, KEDIT does not compress blanks appearing after the first single quote or double quote on a line. Also, even with TABSOUT ON, single blanks are not changed to tabs.

See also User's Guide Chapter 12, "File Processing", COMPRESS, EXPAND, SET TABSAVE, SET TABSIN

SET THIGHLIGHT

Format **`[Set] THIGHlight ON|OFF`**

KEDIT default: ON

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET THIGHLIGHT (''Target HIGHLIGHT'', pronounced ''tee highlight'') controls KEDIT's target highlighting facility.

With THIGHLIGHT ON, when the LOCATE, CLOCATE, or TFIND commands, or the Edit Find dialog box find a string target, KEDIT highlights it on your display.

The target remains highlighted until you issue another command from the command line, until another LOCATE, CLOCATE, or TFIND command is executed, until another Edit Find operation takes place, until you add or delete a line in your file, change the highlighted line, mark a block, or you issue the RESET THIGHLIGHT command (normally assigned, along with RESET BLOCK, to Alt+U).

The color used to highlight the target is determined by SET COLOR THIGHLIGHT.

SET TIMECHECK

Format **`[Set] TIMECHECK ON|OFF`**

KEDIT default: ON

Level: File

Dialog box: Options SET Command

Save Settings handling: Savable

Description

Every disk file has associated with it a timestamp, indicating when the file was last changed. (This is the same as the date and time displayed for a file by the DIR command.)

SET TIMECHECK controls what happens when the timestamp that a file had when you began to edit it differs from the timestamp that the disk copy has when you try to FILE or SAVE it. With TIMECHECK ON, the default, you get a message warning you that the timestamp has changed. This lets you know that some other program or network user may have changed the disk file while you were editing it. You can use the FFILE or SSAVE commands if you want to write the file to disk despite this. With TIMECHECK OFF, the check is not performed.

If you use File Save or a related menu item to save your file to disk with TIMECHECK ON and the timestamps do not match, KEDIT displays a dialog box that asks whether you want to save the file anyway.

After a successful FILE or SAVE, both the disk file's timestamp and KEDIT's internal record of the timestamp are updated.

SET TIMECHECK lets other programs or users change a file you are editing, and then warns about what has happened when you try to write the file to disk. You can use KEDIT's file locking facility, controlled by SET LOCKING, to prevent such changes from occurring.

See also

User's Guide Chapter 12, "File Processing", SET LOCKING

SET TOFEOF

Format

`[Set] TOFEOF ON|OFF`

KEDIT default: ON

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description

With TOFEOF ON, the default, KEDIT displays the top-of-file and end-of-file lines on your display in the usual way. With TOFEOF OFF, KEDIT does not display the normal top-of-file and end-of-file lines, but instead displays blank lines in their place.

You can control the color of the top-of-file and end-of-file lines with the SET COLOR TOFEOF command.

SET TOFEOF is provided for specialized situations and is rarely used.

SET TOOLBAR

Format **`[Set] TOOLBAR ON|OFF Top|Bottom|BOTH`**

KEDIT default: ON TOP

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET TOOLBAR determines whether KEDIT displays toolbars, which are sets of buttons that you can select with the mouse to carry out common operations.

The first operand, ON or OFF, determines whether toolbars are displayed at all. The second operand determines whether, when toolbars are displayed, they should be displayed at the TOP, the BOTTOM, or BOTH the top and bottom of the frame window.

The top and bottom toolbars display different sets of buttons. Useful default toolbar layouts are built into KEDIT, and you can use the SET TOOLSET command (in connection with the SET TOOLBUTTON command) to define your own toolbar contents.

See also SET TOOLBUTTON, SET TOOLSET

SET TOOLBUTTON

Format **`[Set] TOOLButton `*`name visual`*` [COND `*`ccc`*`] /`*`help1`*`/[`*`help2`*`/]`**

KEDIT default: See the table below

Level: Global

Dialog box: None

Save Settings handling: Not savable

Description Use SET TOOLBUTTON to define the buttons and other items that can appear on a toolbar. For each button, you specify the name of that button, how it will look when it is displayed on a toolbar, any conditions under which the button should be disabled, and the help information displayed for the button.

A number of TOOLBUTTON definitions are built into KEDIT and are used on the default toolbars. Use QUERY TOOLBUTTON * to see all of the current toolbutton definitions. You can use the SET TOOLBUTTON command to modify these default TOOLBUTTON definitions or to define new buttons of your own. You can then use the

SET TOOLSET command to specify which of your buttons will appear on the top or bottom toolbar and how they will be arranged on the toolbar.

Here is a sample TOOLBUTTON command:

```
SET TOOLBUTTON ADDLINE ADDLINE.BMP /Add Line/
```

This tells KEDIT to define a button called ADDLINE. When displaying this button on the toolbar, KEDIT uses the image in the bitmap file ADDLINE.BMP. The COND operand, which specifies conditions in which the button should be disabled, is omitted here, so the button is always enabled. When the mouse pointer is positioned over this button, KEDIT will display the help information ''Add Line'' in a popup box and in the status line.

Here are the components of the SET TOOLBUTTON command:

Name

The *name* is used for two things:

First, when you use the SET TOOLSET command to control which buttons appear on the toolbar, you give the names of all of the buttons involved. In the example above, ADDLINE is the name of the button and ADDLINE must be specified in a SET TOOLSET command before the button will be displayed on a toolbar.

Second, whenever you click on a toolbar button, KEDIT runs a macro called TOOL_*name* (that is, ''TOOL_'' followed by the name of the button). Macros for the default toolbar buttons are built into KEDIT, but you are free to redefine them. If you define your own toolbar buttons, you will also need to define the TOOL_*name* macros involved, which must be in-memory macros. In the above example, you would need to define a macro called TOOL_ADDLINE.

Visual

The *visual* representation of a toolbar button can be specified in four ways:

Builtin

You can give the name of one of the predefined toolbar bitmap images built into KEDIT, for example the NEW bitmap (used for the New File toolbar button) or the OPEN bitmap (used for the Open File toolbar button).

Filename.BMP

You can give the name of a bitmap file on disk. The file extension is normally .BMP, and KEDIT uses the same search order to look for these files as it does for macros.

You can create your own bitmaps using a paint program or a resource editing program that can handle .BMP files. The bitmaps can be any size that you like, but all of KEDIT's built-in buttons are 16 pixels wide by 15 pixels high, and it is possible that in future versions of KEDIT we will need to standardize on a fixed bitmap size like this. Two sample bitmap files, ADDLINE.BMP and DELLINE.BMP are supplied with KEDIT and installed by default in the

SAMPLES subdirectory of the main KEDITW directory. The example above uses the sample ADDLINE.BMP file.

/*Text*/

Instead of a bitmap image, KEDIT can display a button containing the *text* you specify. Slash ("/") characters are normally used to delimit the text, but you can use any special character other than backslash ("\") that does not appear within the text.

QUICKFIND

The name QUICKFIND is given special handling; it designates the Quick Find combo box on the default toolbar, used to search for text. All of the other toolbar items are buttons that run the TOOL_*name* macro in response to a mouse click. When you select the Quick Find toolbar item, no macro is executed; Quick Find is instead handled directly by special code within KEDIT. Quick Find can only appear on the top toolbar, and can only appear there once.

COND *ccc*

Use COND, followed by a string of one or more characters, to specify conditions under which the button will be disabled. COND is used by all of KEDIT's default toolbutton definitions, but it is optional and you may want to ignore it if you are just getting started with SET TOOLBUTTON. If COND is omitted, the toolbutton is always enabled and the associated TOOL_*name* macro is always run when the button is selected.

The table near the end of this description lists each of the conditions that you can specify, and the character corresponding to the condition. For example, condition A means the current document window is minimized and condition B means that the current file is empty. So a SET TOOLBUTTON command specifying COND AB would mean that the button should be disabled if the document window is minimized or the file is empty. If you do a Q TOOLBUTTON * to look at KEDIT's default toolbutton definitions, you will see that many of the default buttons are disabled under exactly these conditions.

When a button is disabled, KEDIT displays the reason why (that is, the reason corresponding to the first condition in the list that you specified that turns out to be true) on the status line, instead of the normal help for the button, and beeps at you if you try to select that button. The TOOL_*name* macro associated with the button is not executed.

***/help1/*[*help2/*]**

TOOLBUTTON definitions also include the help text displayed by KEDIT when the mouse pointer is positioned over the button. Help is displayed in two places: in a popup box near the tool button, and on the status line. You can specify either a single help string to be displayed in both places, or you can specify a pair of help strings, with the first string displayed in the popup box and the second string, usually the longer of the two, displayed on the status line. Slash ("/") characters are normally used to delimit the text, but you can use any special character that does not appear within the text.

In our example above, the help text is given as

```
/Add Line/
```

The string ‘‘Add Line’’ is used as the help text in both the popup box and on the status line. Had the example instead used

```
/Add Line/Add a new line below the focus line/
```

the second string would be used as the status line help. The default toolbuttons all use the popup help box to display a short title for the toolbutton, and use the status line for more detailed information.

The file DEFTOOLB.KEX, in the SAMPLES subdirectory of the main KEDITW directory, has commands corresponding to all of the default SET TOOLSET and SET TOOLBUTTON definitions.

Conditions

Here is a table of the conditions, and the character corresponding to them, used with the optional COND operand of SET TOOLBUTTON:

Char	Condition
A	Current document window is minimized
B	Current file is empty
C	No files in the ring
D	Only one file in the ring
E	Only one document window
F	No minimized document windows
G	No non-minimized document windows
H	No selection (non-persistent block) in the current file
I	No block (persistent or non-persistent) in the current file
J	No block in any file
K	No block or command line selection in the current file
L	No block or command line selection in any file
M	No line block in the current file
N	No line block in any file
O	No box block or one-line stream block in the current file
P	No box block or one-line stream block in any file
Q	No stream block in the current file
R	No stream block in any file
S	No line
T	Nothing to Undo

Char	Condition
U	Nothing to Redo
V	There is no text in the clipboard
W	Clipboard Cut and Copy not currently possible
X	Bookmark1 is not defined
Y	Bookmark2 is not defined
Z	Bookmark3 is not defined

If any conditions at all are specified, condition C is also applied and the toolbutton is disabled if there are no files in the ring. You would therefore only need to specify condition C for a button that is disabled only when there are no files in the ring.

See also

SET TOOLBAR, SET TOOLSET

Examples

```
"TOOLB BLKUPPER BLKUPPER COND AI /Uppercase Block/Uppercase block/"
"DEF TOOL_BLKUPPER 'uppercase block'"
```

This is the default definition for the Uppercase Block button on KEDIT's default bottom toolbar and for its associated macro. The two commands are quoted as they would be if included in a macro. The button's name is BLKUPPER; it is displayed as a bitmapped icon called BLKUPPER (this icon is built into KEDIT); the button is disabled if conditions A (the document window is minimized) or I (there is no block in the current file) are true.

```
"TOOLB REV /Reverse/ /Reverse Line/Reverse line"s contents/"
"DEF TOOL_REV 'replace' reverse(curline.3())"
"TOOLSET TOP ADD REV"
```

These three lines, if included in a macro, would define a toolbutton called REV and its associated macro, TOOL_REV, and would add the button to the top toolbar. The button is not displayed as a bitmapped icon, but instead as the text ''Reverse'', with ''Reverse Line'' as the popup toolbar help and ''Reverse line's contents'' as the status line help. No COND operand is specified, so the button is always enabled.

Note that the delimiters for the button text (''Reverse'') are separate from the delimiters for the help text, and that the following would not be valid:

```
"TOOLB REV /Reverse/Reverse Line/Reverse line"s contents/"
```

The TOOL_REV macro replaces the focus line with the reverse of its current contents.

SET TOOLSET

Format

```
[Set] TOOLSet [Top|Bottom|NOFiles] DEFAULT
[Set] TOOLSet [Top|Bottom|NOFiles] USER toolbuttons
[Set] TOOLSet [Top|Bottom|NOFiles] ADD toolbuttons
[Set] TOOLSet [Top|Bottom|NOFiles] DELete toolbuttons
```

KEDIT default: TOP|BOTTOM|NOFILES DEFAULT

Level: TOP and BOTTOM, File level; NOFILES, Global level

Dialog box: None

Save Settings handling: Not savable

Description

Use SET TOOLSET to determine which buttons will appear on one of KEDIT's toolbars, and how the buttons will be arranged.

Here are the components of the SET TOOLSET command:

TOP|BOTTOM|NOFILES
The first operand determines which toolbar's contents you want to set. You can specify the TOP or BOTTOM toolbar for the current file. These are controlled on a per-file basis, so that each file in the ring can potentially have different toolbar contents. You can also specify the special NOFILES toolset, displayed as the top toolbar when no files are in the ring; the bottom toolbar is empty when no files are in the ring. If this operand is omitted, the TOP toolbar is assumed.

DEFAULT
If the second operand is DEFAULT, the specified toolbar displays a default toolset that is built into KEDIT.

USER *toolbuttons*
If the second operand is USER, the list specified via the *toolbuttons* operand determines the toolbar's contents. *Toolbuttons* is a list with the names of the buttons that should appear on the toolbar. Each item in the list must be the name of a button that is either one of the predefined toolbar buttons built into KEDIT, or has been previously defined via the SET TOOLBUTTON command. Periods (".") can also appear in the list to indicate spacing between toolbar buttons, which otherwise appear immediately adjacent to one another.

ADD *toolbuttons*
DELete *toolbuttons*
If you want to make minor adjustments to the current toolbar contents, you can use the ADD or DELETE operands. With ADD, the toolbuttons that you specify are added to the end of the current toolbar contents. With DELETE, the toolbuttons that you specify are removed from the current toolbar contents. If you include a blank-delimited period (".") following any item in your list of buttons to be deleted, KEDIT not only removes the specified toolbutton from the toolbar, but also any spacing that follows that toolbutton on the toolbar.

The file DEFTOOLB.KEX, in the SAMPLES subdirectory of the main KEDITW directory, has commands corresponding to all of the default SET TOOLSET and SET TOOLBUTTON definitions.

See also SET TOOLBAR, SET TOOLBUTTON

Examples

```
TOOLSET TOP USER . OPEN NEW . UNDO REDO . GET PUT
```

This example specifies a user-defined top toolbar that displays some spacing, the OPEN and NEW buttons, more spacing, the UNDO and REDO buttons, more spacing, and then the GET and PUT buttons. OPEN, NEW, UNDO, and REDO are all default tool buttons whose definitions are built into KEDIT. Before issuing this command, you would need to use the SET TOOLBUTTON command to define GET and PUT buttons, since these are not default buttons. You can use QUERY TOOLBUTTON * to see all of the current toolbutton definitions.

```
TOOLSET TOP DEFAULT
TOOLSET TOP ADD . COMPILE DEBUG
```

Here, KEDIT's default toolbar contents are displayed, followed by some spacing, and then COMPILE and DEBUG buttons, both of which must have been previously defined via the SET TOOLBUTTON command. If you wanted to display these buttons only when you were editing C programs, and otherwise display only the default top toolbar, you might have something like the following in your KEDIT profile:

```
'reprofile on'
if initial() then do
   'set toolbutton compile /Compile/ /Compile file/'
   'set toolbutton debug  /Debug/  /Debug file/'
   'define toolmacs.kml'
   end
'set toolset top default'
if fext.1() = 'C' then
   'set toolset top add . compile debug'
```

You would also need to define the macros TOOL_COMPILE and TOOL_DEBUG. You would most likely put them in a .KML file (like TOOLMACS.KML in this example) that is loaded via the DEFINE command during the initial execution of your profile at the start of a KEDIT session.

SET TRAILING

Format **`[Set] TRAILING ON|OFF|SINGLE|EMPTY`**

KEDIT default: OFF

Level: File

Dialog box: Options SET Command

Save Settings handling: Savable

Description SET TRAILING controls how trailing blanks in the files that you edit are handled.

`[Set] TRAILING ON`
If you put TRAILING ON into effect, you can work with files that contain trailing blanks. Trailing blanks are kept when files are read from disk, you can work with them as you edit the file, and they are included as part of your file when it is written back to disk.

`[Set] TRAILING OFF`
This is the default, and it specifies that KEDIT will not process trailing blanks in your files. Trailing blanks are eliminated as your file is read in from disk. The lines of your file are stored internally without any trailing blanks, and any trailing blanks that you add while editing your file are ignored. When your file is written to disk, it contains no trailing blanks.

`[Set] TRAILING SINGLE`
This is a rarely-used operand that works like TRAILING OFF, except that a single trailing blank is added to each line as it is written to disk.

`[Set] TRAILING EMPTY`
This is a rarely-used operand that works like TRAILING OFF, except that a single trailing blank is added to each blank line as it is written to disk. Nonblank lines are written to disk with no trailing blanks.

Note that when RECFM FIXED is in effect, lines in your file can be truncated or padded with blanks when they are written to disk so that they will be exactly LRECL bytes long. So in this case, SET TRAILING does not control the number of trailing blanks written to disk.

See Also User's Guide Chapter 12, "File Processing"

SET TRANSLATEIN, TRANSLATEOUT

Format

[Set] TRANSLATEIn NONE|OEMTOANSI
[Set] TRANSLATEOut NONE|ANSITOOEM

KEDIT default: NONE

Level: File

Dialog box: Options SET Command

Save Settings handling: Not savable

Description

SET TRANSLATEIN controls whether KEDIT converts 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, controls whether KEDIT converts 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. Before using SET TRANSLATEIN or SET TRANSLATEOUT, be sure to read User's Guide Section 3.7, "Character Sets", which discusses the ANSI and OEM character sets and KEDIT's facilities for dealing with them, and which has a discussion of the proper use of SET TRANSLATEIN and SET TRANSLATEOUT.

See also

ANSITOOEM, OEMTOANSI, User's Guide Section 3.7, "Character Sets"

SET TRUNC

Format

[Set] TRunc *n*|*

KEDIT default: * (WIDTH value)

Level: View

Dialog box: Options SET Command

Save Settings handling: Not savable

Description

Use SET TRUNC to control which column serves as KEDIT's ''truncation column''. You can specify a column number *n* or can specify an asterisk (''*'') to indicate that the truncation column should be set equal to the value of the WIDTH initialization option.

The truncation column is the rightmost column of text that is affected by most KEDIT commands. You cannot input, overtype, change, or delete characters to the right of the truncation column. Depending on the VERIFY setting, you may be able to see text beyond the truncation column, but most KEDIT commands act as if text to the right of the truncation column was not there.

SET TRUNC does not affect how text is read in from disk or written out to disk by KEDIT. The value of the WIDTH initialization option normally controls the maximum length of lines read in, and the LRECL setting controls the maximum length of lines that are written out.

Normally, the truncation column is equal to the value of the WIDTH initialization option, so there are normally no columns of your file to the right of the truncation column. Therefore, unless you change the default TRUNC value, it has no effect on KEDIT's operation. The TRUNC value is rarely changed, and is provided for use in specialized situations.

You cannot set your MARGINS or ZONE columns to the right of the truncation column. If you issue a SET TRUNC command that specifies a column to the left of the current MARGINS or ZONE columns, KEDIT automatically resets these columns to be equal to the new truncation column. If you specify a truncation column to the right of the current MARGINS or ZONE columns, they are not reset.

SET UNDOING

Format

`[Set] UNDOING ON|OFF [`*`n`*` [`*`k`*`]]`

KEDIT default: ON 200 512

Level: File

Dialog box: Options SET Command

Save Settings handling: Savable

Description

SET UNDOING controls, whether the undo facility is enabled or disabled for the current file, how many undo levels KEDIT attempts to keep, and the maximum amount of memory that KEDIT will use to hold the undo information.

By default, KEDIT will try to save up to the last 200 levels of changes to your file, with a maximum of 512K of undo information per file.

Whenever KEDIT reaches the limits specified by SET UNDOING, it automatically discards as much undo information as necessary so that editing can continue. No message

is given telling you that this happened, but you can always tell how many levels of changes are available to the UNDO command by looking at the counter displayed on the status line as the third number after ALT=. QUERY UNDO gives more information, telling you how many levels of changes are available for the UNDO command, how many levels are available for the REDO command, and how many kilobytes of memory are occupied by the undo information for the current file.

See also User's Guide Chapter 3, "Using KEDIT for Windows", REDO, UNDO

Examples

```
SET UNDOING ON 400 1024
```

Tells KEDIT to save up to 400 levels of changes to the current file, with a maximum of 1024K for the data involved.

SET VARBLANK

Format **`[Set] VARblank ON|OFF`**

KEDIT default: OFF

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description With VARBLANK ON, a blank in a string target or column string target will match any string of one or more blanks. This is useful if, for example, the FLOW command has justified text and you are looking for a group of words that may have had extra blanks sprinkled among them.

With VARBLANK OFF, there is no special handling of blanks during searches. A blank will only match exactly one blank.

SET VERIFY

Format **`[Set] Verify [Hex] `*`l1 r1`*` [[Hex] `*`l2 r2`*` ...]`**

KEDIT default: 1 * (Column 1 through WIDTH value)

Level: View

Dialog box: Options SET Command

Save Settings handling: Not savable

Description

Use SET VERIFY to control which columns of your file KEDIT displays in the current view of your file.

SET VERIFY has no effect on the contents of your file; it merely affects how KEDIT displays your file while you are editing it.

In a window 80 columns wide, KEDIT normally displays columns 1 through 80 of your file. Columns of text to the right of column 80 are not displayed. With SET VERIFY, you can specify a pair of columns that are the leftmost and rightmost columns that you want KEDIT to display. For example, to have KEDIT display columns 20 through 99 of your file, you could enter

```
SET VERIFY 20 99
```

To view only columns 30 through 40, you could enter

```
SET VERIFY 30 40
```

Since the columns specified do not fill the 80-column width of the window, most of the document window would be empty. If, on the other hand, you specify a column range wider than the window, KEDIT displays only as many columns as will fit. So in an 80 column window,

```
SET VERIFY 100 200
```

would cause KEDIT to display columns 100 through 179.

If the rightmost column to display is specified as an asterisk ("*") or is omitted, KEDIT displays as many columns of your text as will fit. So, in an 80-column window, all three of the following would cause display of columns 1 through 80:

```
SET VERIFY 1 *
SET VERIFY 1
SET VERIFY 1 80
```

A very powerful aspect of SET VERIFY is that it lets you specify more than one pair of columns at a time. For example,

```
SET VERIFY 30 40 10 20 50 *
```

tells KEDIT to display columns 30 through 40 of your text, followed by columns 10 through 20 of your text, and then all text from column 50 on that will fit.

You can precede specification of a pair of columns for SET VERIFY with the word "HEX" to tell KEDIT that you want to display and work with the hexadecimal value of the character codes involved, rather than the characters themselves. The display of each column of your file will then occupy two columns on your display: the first column has the high order hexadecimal digit of the character code for the character and the second column has the low order digit of the code. For example,

```
SET VERIFY HEX 1 20 1 20
```

tells KEDIT to display columns 1 through 20 of your file in hexadecimal format, and then to display the same columns again in the normal character format.

The value of VERSHIFT, which is controlled by the LEFT and the RIGHT commands and by autoscrolling the display, interacts with the SET VERIFY setting to determine which columns are actually displayed by KEDIT. If, for example, you have an 80-column window and you issue the command

```
SET VERIFY 1 40 70 *
```

KEDIT will display columns 1 through 40 and 70 through 109 of your file. If you then tell KEDIT to shift the window to the right by issuing the command RIGHT 10, VERSHIFT will be set to 10 and the columns displayed will be offset by 10 columns from what you specified with SET VERIFY. KEDIT will therefore display columns 11 through 50 and 80 through 119 of your file.

See also LEFT, RIGHT, SET AUTOSCROLL

SET WINMARGIN

Format **`[Set] WINMARgin ON|OFF [`*`n`*`]`**

KEDIT default: ON 6

Level: Global

Dialog box: Options SET Command

Save Settings handling: Savable

Description Use SET WINMARGIN to control the display of a margin area that you can use to mark line blocks.

When WINMARGIN ON is in effect, a margin area—normally 6 pixels wide—is displayed to the left of the first column of each document window. You can use this margin area to mark line blocks by placing the mouse pointer in the margin (the mouse pointer will become an arrow pointing upward and to the right) and dragging with mouse button 1 down. With WINMARGIN OFF, this margin area is not displayed.

The second operand to SET WINMARGIN controls the width, in pixels, of the margin area.

Examples

```
WINMARGIN ON 6
```

Display a margin area 6 pixels wide.

```
WINMARGIN ON
```

Turn on display of the margin area, using whatever pixel width value is currently in effect.

SET WORD

Format

`[Set] WORD NONBlank|ALPHAnum [TRAILing|NOTRAILing]`

KEDIT default: NONBLANK TRAILING

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description

The first operand of SET WORD, NONBLANK or ALPHANUM, controls what KEDIT considers to be a ''word'' when you use Shift+Ctrl+W (SOS DELWORD), Ctrl+Curr (SOS TABWORD), Ctrl+Curl (SOS TABWORDB), use the mouse to mark words, or use the MARK STREAM WORD command. (Most other areas of KEDIT, such as wordwrap facility, paragraph reformatting, and KEXX built-in functions, always consider groups of consecutive nonblank characters to be words and are unaffected by SET WORD. Word targets look for strings bordered by nonalphanumeric characters and are also unaffected by SET WORD.)

With WORD NONBLANK, the default, KEDIT considers any group of consecutive nonblank characters to be a word.

With WORD ALPHANUM, KEDIT considers a group of consecutive alphanumeric characters (that is, letters and numbers) to be a word. (Underscore characters and special characters with character codes greater than 127 are also grouped with the alphanumeric characters here. Underscores are frequently used within variable names in programming languages, and many languages use accented characters with character codes above 127.) KEDIT also treats a sequence of nonblank characters that are not alphanumeric (that is, a sequence of punctuation characters) as a word.

Consider the following example, which might come from a FORTRAN program:

```
CALL PROCESS (PI**2)
```

With WORD NONBLANK, KEDIT would treat this line as having three words: ''CALL'', ''PROCESS'', and ''(PI**2)''. If you placed the cursor on the ''P'' in ''PI'' and deleted a ''word'' with Shift+Ctrl+W, then ''(PI**2)'' would be deleted.

With WORD ALPHANUM, KEDIT would treat this line as having seven words: ''CALL'', ''PROCESS'', ''('', ''PI'', ''**'', ''2'', and '')''. Placing the cursor on the ''P'' in ''PI'' and pressing Shift+Ctrl+W would delete only ''PI''.

The second operand of SET WORD, TRAILING or NOTRAILING, controls whether marking a word (usually done by double-clicking on the word) does or does not cause trailing blanks following the word to also be marked.

SET WORDWRAP

Format `[Set] WORDWrap ON|OFF`

KEDIT default: OFF

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description With WORDWRAP ON, whenever text that you are entering with KEDIT would extend beyond the right margin, a new line is automatically added to the file, with the word you are typing placed starting at the left margin and the cursor positioned so that you can continue with your typing.

With WORDWRAP OFF, KEDIT allows text entry to continue beyond the right margin.

See also User's Guide Chapter 3, "Using KEDIT for Windows", FLOW, SET MARGINS

SET WRAP

Format `[Set] WRap ON|OFF`

KEDIT default: OFF

Level: View

Dialog box: Options SET Command

Save Settings handling: Savable

Description The WRAP option affects how far KEDIT searches for a string target when you issue a LOCATE, TFIND, FIND, NFIND, FINDUP, or NFINDUP command, and when you use the Edit Find dialog box.

If WRAP is OFF, KEDIT searches from the line below the focus line through the bottom of the file for the desired string. (For a backward search, KEDIT searches backward from the line above the focus line to the top of the file.) If the string is not found, you get a ''Target not found'' error message.

If WRAP is ON, no error occurs when the bottom of the file is hit. Instead, KEDIT wraps around to the top of the file and continues to search from there. You get an error message if KEDIT goes full circle and hits the focus line again without finding the

string. (With a backward search, when KEDIT hits the top of the file it will wrap around to the bottom of the file and continue the search.) If the string is located after wrapping around to the top of the file, KEDIT indicates this by displaying the message ‘‘Wrapped...’’.

The WRAP option has a similar effect on string column searches carried out by the CLOCATE command when STREAM ON is in effect.

SET ZONE

Format

`[Set] Zone` *`n1 n2`*

KEDIT default: 1 * (Column 1 through the truncation column)

Level: View

Dialog box: Options SET Command

Save Settings handling: Not savable

Description

The ZONE setting affects which columns KEDIT looks in when it searches for strings during target searches and during the CHANGE and related commands, and when you use the Edit Find, Edit Replace, and Edit Selective Editing dialog boxes. When KEDIT looks for a string, it only looks within the columns specified by the ZONE setting. Normally, ZONE is set so that all columns of text on a line will be searched.

The ZONE setting involves two numbers. The first number you give is the left end of the zone you want to search. The second number is the right end of the zone. The left zone value must be less than or equal to the right zone value. The right zone value must be less than or equal to the truncation column. For example,

```
ZONE 10 20
```

says that searches should only look in columns 10 through 20 of each line. If all or part of the string you are looking for were outside of columns 10 through 20, the string would not be found.

You can specify an asterisk (‘‘*’’) for the right zone setting, and KEDIT will set the right zone column equal to the truncation column.

ZONE also affects the handling of column commands such as CLOCATE and affects the processing of the CMATCH, FILL, LOWERCASE, UPPERCASE, SHIFT, ANSITOOEM, OEMTOANSI, and SORT commands.

Examples

```
ZONE 5 30
```

String searches should only look in columns 5 to 30.

```
ZONE 1 *
```

String searches should look in all columns of your file through the truncation column.

SET =

Format

Set = *text*

KEDIT default: Empty string

Level: View

Dialog box: None

Save Settings handling: Not savable

Description

SET = can be used to set the contents of the equal buffer, which is normally set to repeat the most recently completed command executed from the KEDIT command line. The contents of the equal buffer determine the command that is re-executed when you use the = command or the REPEAT command. SET = is useful mainly from within a KEDIT macro, and provides the only way to change the contents of the equal buffer from within a macro.

Note that toolbar and menu operations, such as use of the Edit Find dialog box, do not affect the contents of the equal buffer.

See also

=, REPEAT

Chapter 5. QUERY and EXTRACT

This chapter discusses the QUERY command, which displays on your screen information about internal KEDIT settings. It discusses the EXTRACT command and the related implied EXTRACT functions, which return information to a macro about internal KEDIT settings. Finally, it gives details on each of the operands accepted by these commands and the information they return.

5.1 QUERY

You can use the QUERY command to find out the current value of any KEDIT SET option. KEDIT displays the result of your QUERY on the message line. The format of the QUERY command is:

```
Query option
```

For example, if you have entered

```
set zone 5 65
```

then the command

```
query zone
```

will display

```
zone 5 65
```

You can use the same minimal truncations for *option* that the SET command accepts.

In addition to the values of KEDIT SET options, you can QUERY the values of some KEDIT initialization options, such as ISA and WIDTH. You can also QUERY a number of values that you cannot directly set at all, such as the SIZE of your file, or the VERSION of KEDIT that you are using. Section 5.3, "QUERY and EXTRACT Operands", gives full details on all values that you can QUERY.

The MODIFY command does the same thing as the QUERY command, except that it displays the result on the command line, preceded by the word ''SET'', so that you can easily change the value of a SET option and re-enter it.

The STATUS command displays, in a dialog box, the values of most SET options.

5.2 EXTRACT and Implied EXTRACTs

EXTRACT command

The EXTRACT command, valid only when issued from a macro, provides one method for the macro to obtain information from KEDIT.

This discussion shows how the EXTRACT command works, giving the general pattern followed with most operands of the command. Section 5.3, "QUERY and EXTRACT Operands", gives specific details on the variables set and information returned for all EXTRACT operands.

Most things that you can QUERY can be EXTRACTed. The EXTRACT command is given a list of one or more QUERY operands, surrounded by delimiter characters (a slash ("/") is used here, but any nonalphanumeric character is acceptable). The main difference between EXTRACT and QUERY is that, with EXTRACT, the results are not displayed, but are instead used to set the values of variables within your macro. For example, if ZONE 5 65 is in effect,

```
query zone
```

would display

```
zone 5 65
```

while

```
extract /zone/
```

which would normally be within quotes in your macro:

```
'extract /zone/'
```

will return the current zone setting: The variable ZONE.1 is set to 5 (the left zone column) and ZONE.2 is set to 65 (the right zone column). ZONE.0 is set to 2, indicating that two pieces of information have been returned to your macro.

```
extract /zone/wrap/
```

sets ZONE.1 to the left zone, ZONE.2 to the right zone, ZONE.0 to 2, WRAP.1 to the value of WRAP ("ON" or "OFF", in uppercase), and WRAP.0 to 1. In general, if

```
query option
```

would display

```
option word1 word2 word3 ... wordn
```

then

```
extract /option/
```

sets *option*.1 to *word1*, *option*.2 to *word2*, etc., and *option*.0 to *n*.

Case of EXTRACT results

The results of a QUERY command are usually displayed in lowercase, while results from EXTRACT are usually in uppercase. In a few cases, where it is important to return text exactly as it exists in your file or as it was entered from the keyboard, the result is returned in mixed case.

When you give an option to the EXTRACT command, you can truncate the option in the same way that you can truncate it with the SET or QUERY command. So, since ''wr'' is an acceptable truncation for WRAP,

```
extract /wr/
```

is acceptable. (Note, however, that WRAP.0 and WRAP.1 would be set in this case, and not WR.0 and WR.1)

Implied EXTRACT functions

Macros can also extract information from KEDIT by using special functions known as *implied EXTRACT* functions. 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. For example,

```
say zone.1()
```

displays the same thing as

```
'extract /zone/'
say zone.1
```

Note that for EXTRACT operands that consist of more than a single word, for example,

```
'extract /color arrow/'
```

the additional words must be passed to the implied EXTRACT as a parameter. For example:

```
color.1('arrow')
```

5.3 QUERY and EXTRACT Operands

The following QUERY and EXTRACT operands, supported in earlier text mode versions of KEDIT, are not used by this version of KEDIT for Windows: BLINK, BORDER, CURSORSHAPE, EAPRESERVE, ISA, KEYBOARD, LOGO, MOUSE, MOUSEBAR, MOUSETEXT, PSCREEN, RETRACE, REXXIO, SCREEN, SHIFTSTATE, SWAP, SYSRC, and TOPVIEW. Any QUERY or MODIFY commands involving these options will yield an error message. EXTRACT commands and implied EXTRACT functions involving these options will return default information; avoiding an error message in this situation means that many existing macros that use these options will continue to work.

ALT

Query ALT displays two numbers: the number of alterations to your file since the last AUTOSAVE or SAVE and the number of alterations since the last SAVE.

EXTract /ALT/ sets these variables:

alt.0	2
alt.1	Number of alterations since last AUTOSAVE or SAVE
alt.2	Number of alterations since last SAVE

ARBCHAR

Query ARBchar displays whether ARBCHAR is ON or OFF, and the first and second ARBCHAR characters.

EXTract /ARBchar/ sets these variables:

arbchar.0	3
arbchar.1	ON|OFF
arbchar.2	First ARBCHAR character
arbchar.3	Second ARBCHAR character

ARROW

Query ARRow displays whether ARROW is ON or OFF.

EXTract /ARRow/ sets these variables:

arrow.0	1
arrow.1	ON|OFF

ATTRIBUTES

QUERY and EXTRACT ATTRIBUTES are supported mainly for compatibility with older versions of KEDIT; the corresponding SET ATTRIBUTES command is also still available but is no longer documented. SET/QUERY COLOR is now preferred.

Query ATTRibutes displays the 30 current attribute values.

EXTract /ATTRibutes/ sets these variables:

attributes.0	30
attributes.1	Filearea attribute value
attributes.2	Curline attribute value
attributes.3	Block attribute value
attributes.4	Cblock attribute value
attributes.5	Cmdline attribute value
attributes.6	Idline attribute value
attributes.7	Msgline attribute value
attributes.8	Arrow attribute value
attributes.9	Prefix attribute value
attributes.10	Pending attribute value
attributes.11	Scale attribute value
attributes.12	Tofeof attribute value
attributes.13	Ctofeof attribute value
attributes.14	Tabline attribute value
attributes.15	Shadow attribute value
attributes.16	Statarea attribute value
attributes.17	Divider attribute value
attributes.18	Scrollbar attribute value
attributes.19	Slider attribute value
attributes.20	Mousebar attribute value
attributes.21	Highlight attribute value
attributes.22	Chighlight attribute value
attributes.23	Thighlight attribute value
attributes.24	Cthighlight attribute value
attributes.25	Dialog attribute value
attributes.26	Alert attribute value
attributes.27	Tooltip attribute value
attributes.28	Currbox attribute value
attributes.29	Boundmark attribute value
attributes.30	Colmark attribute value

AUTOCOLOR Query AUTOCOLOR *ext* displays the name of the parser associated with the specified extension, or NULL if there is no such parser.

EXTract /AUTOCOLOR *ext*/ sets these variables:

autocolor.0	2
autocolor.1	Extension (with a leading period)
autocolor.2	Name of associated parser, or NULL

Query AUTOCOLOR * or simply Query AUTOCOLOR displays, for each extension that has an associated parser, the extension involved and the name of the parser.

EXTract /AUTOCOLOR/ or EXTRACT /AUTOCOLOR */ sets the following variables:

autocolor.0	number of extensions with associated parsers
autocolor.*i*	*i*th extension (with a leading period) and parser

AUTOEXIT Query AUTOEXIT displays whether AUTOEXIT is ON or OFF.

EXTract /AUTOEXIT/ sets these variables:

autoexit.0	1
autoexit.1	ON|OFF

AUTOINDENT Query AUTOIndent displays whether AUTOINDENT is ON or OFF.

EXTract /AUTOIndent/ sets these variables:

autoindent.0	1
autoindent.1	ON|OFF

AUTOSAVE Query AUtosave displays the alteration count at which an AUTOSAVE is triggered, or OFF.

EXTract /AUtosave/ sets these variables:

autosave.0	1
autosave.1	*n*|OFF

AUTOSCROLL Query AUTOSCroll displays the current setting of AUTOSCROLL: HALF, OFF, or *n* columns.

EXTract /AUTOSCroll/ sets these variables:

autoscroll.0	1
autoscroll.1	HALF|OFF|*n*

BACKUP Query BACKup displays the current setting of BACKUP: OFF, TEMP, or KEEP.

EXTract /BACKup/ sets these variables:

backup.0	1
backup.1	OFF|TEMP|KEEP

BEEP Query BEEP displays whether BEEP is ON or OFF.

EXTract /BEEP/ sets these variables:

beep.0	1
beep.1	ON|OFF

BLOCK

Query BLOCK displays the type of block currently marked (LINE, BOX, STREAM, or NONE) and, if a block is defined, the file line and column of the start of the block, the file line and column of the end of the block, the fileid of the file containing the block, and whether the block is PERSISTENT or is a SELECTION.

EXTract /BLOCK/ sets these variables:

block.0	8
block.1	LINE|BOX|STREAM|NONE
block.2	Line number of start of block (this and following values are set to the null string if no block is marked)
block.3	Column number of start of block
block.4	Line number of end of block
block.5	Column number of end of block
block.6	Fileid of file containing marked block
block.7	PERSISTENT|SELECTION
block.8	Contents of the currently marked one-line block, or the null string if there is no marked block or the marked block occupies multiple lines

BOUNDMARK

Query BOUNDMark displays a list of the types of boundary marker that are currently active (ZONE, TRUNC, MARGINS, TABS, VERIFY, and/or WINMARGIN) or displays OFF if no boundary markers are active.

EXTract /BOUNDMark/ sets these variables:

boundmark.0	1
boundmark.1	One or more of ZONE, TRUNC, MARGINS, TABS, VERIFY, and WINMARGIN; or OFF

CASE

Query CASE displays whether text is entered in MIXED case or UPPER case, whether string searches RESPECT or IGNORE case, and whether comparisons made during the CHANGE command RESPECT or IGNORE case.

EXTract /CASE/ sets these variables:

case.0	3
case.1	MIXED|UPPER
case.2	RESPECT|IGNORE
case.3	RESPECT|IGNORE

CLICK

Query CLICK displays which mouse button was last pressed, the mouse pointer's line and column position within the document window when the button was pressed, MARGIN (if WINMARGIN ON is in effect and the mouse pointer was in the window margin when the button was pressed) or NOMARGIN, and the line and column of the file at which the cursor was located when the button was pressed.

EXTract /CLICK/ sets these variables:

click.0	6
click.1	Number of the mouse button pressed
click.2	Line number of mouse pointer within document window when the button was pressed
click.3	Column number of mouse pointer within document window when the button was pressed
click.4	MARGIN (if WINMARGIN ON in effect and mouse pointer was in window margin when button was pressed) or NOMARGIN
click.5	Line number within the file of cursor location when button was pressed (or -1 if the cursor is not on a line of the file)
click.6	Column number within the file of cursor location when button was pressed (or -1 if the cursor is not on some column of the file)

CLIPBOARD

Query CLIPboard displays information about the Windows clipboard.

When there is no text in the clipboard, Query CLIPboard returns NONE. Otherwise, Query CLIPboard returns the following information:

The type of text in the clipboard: LINE if the text came from a line block, BOX if the text came from a box block, STREAM if the text came from a stream block, from a command line selection, or from the CLIPBOARD PUT command, and FOREIGN if the text came from an application other than KEDIT.

If the clipboard text came from a box block, the width of the box block; otherwise, this item is omitted.

The size of the clipboard text, in characters

The size of the clipboard text, in lines

EXTract /CLIPboard/ sets these variables:

clipboard.0	5
clipboard.1	LINE|BOX|STREAM|FOREIGN or NONE
clipboard.2	If CLIPBOARD.1 = BOX, width of the box block; otherwise, the null string
clipboard.3	Size of clipboard text, in characters
clipboard.4	Size of clipboard text, in lines
clipboard.5	Contents of the clipboard, if the length of the text is less than or equal to the WIDTH setting, and otherwise the null string. The text may contain multiple lines of data, with carriage return and linefeed characters marking the end of each line.

CLOCK

Query CLOCK displays whether display of the time of day on the status line is turned ON or OFF.

EXTract /CLOCK/ sets these variables:

clock.0	1
clock.1	ON|OFF

CMDLINE

Query CMDline displays whether CMDLINE is set to ON, OFF, TOP, or BOTTOM.

EXTract /CMDline/ sets these variables:

cmdline.0	3
cmdline.1	ON|OFF|TOP|BOTTOM
cmdline.2	Line number within window of command line, or 0 if CMDLINE OFF
cmdline.3	Contents of command line, in mixed case, or the null string, if CMDLINE OFF

COLMARK

Query COLMark displays a list of the file columns at which column markers are displayed, or OFF if column marking is not active.

EXTract /COLMark/ sets these variables:

colmark.0	1
colmark.1	List of one or more column numbers, or OFF

COLOR

Query COLOR *field* displays the color setting for *field*, where *field* is one of the following: Arrow, Block, BOUNDMark, CBlock, CHIghlight, Cmdline, COLMark, CTHIghlight, CTofeof, CUrline, CURRBox, Filearea, HIghlight, Idline, Msgline, Pending, PRefix, Scale, SHadow, Tabline, THIghlight, TOfeof, TOOLTip.

EXTract /COLOR *field*/ sets these variables:

color.0	1
color.1	Field name followed by the field's color

Query COLOR * or simply Query COLOR displays the color setting for all of the fields that KEDIT uses, in alphabetical order.

If MONITOR WINDOWS is in effect, KEDIT uses 18 field types: ARROW, BLOCK, BOUNDMARK, CMDLINE, COLMARK, CURRBOX, FILEAREA, HIGHLIGHT, IDLINE, MSGLINE, PENDING, PREFIX, SCALE, SHADOW, TABLINE, THIGHLIGHT, TOFEOF, and TOOLTIP. If MONITOR COLOR or MONITOR MONO is in effect, KEDIT uses 5 additional field types: CBLOCK, CHIGHLIGHT, CURLINE, CTHIGHLIGHT, and CTOFEOF.

EXTract /COLOR/ or EXTract /COLOR */ sets the following variables:

color.0	18 (if MONITOR WINDOWS is in effect) or 23 (if MONITOR COLOR or MONITOR MONO are in effect)

color.*i*	*i*th alphabetical field type, followed by its color

COLORING Query COLORING displays whether the syntax coloring facility is ON or OFF for the current file and displays the name of the parser for the current file, or AUTO if KEDIT determines the parser from the current file extension.

EXTract /COLORING/ sets these variables:

coloring.0	3
coloring.1	ON|OFF
coloring.2	AUTO|*parser* (parser specified via SET COLORING)
coloring.3	*parser* (actual parser; same as COLORING.2 unless COLORING.2 is AUTO)

COLUMN Query COLumn displays the column number of the focus column.

EXTract /COLumn/ sets these variables:

column.0	1
column.1	Column number of the focus column

CURLINE Query CURLine displays the desired current line location, as specified by SET CURLINE.

EXTract /CURLine/ sets the variables listed below. Note that CURLINE.1 and CURLINE.2 have information about the location of the current line, while CURLINE.3, CURLINE.4, CURLINE.5, and CURLINE.6 have information about the focus line.

curline.0	6
curline.1	Value of CURLINE setting
curline.2	Line number within window of current line
curline.3	Contents of the focus line, in mixed case
curline.4	"ON" if the focus line has been changed during the editing session (the line's new or change bit is set); otherwise "OFF"
curline.5	"NEW CHANGED" if the focus line has been added during the editing session, "OLD CHANGED" if the focus line has simply been changed during the editing session, and "OLD" otherwise, as determined from the line's new and change bits
curline.6	Selection level of the focus line. (Same value as SELECT.1, but using CURLINE.6 is much more efficient)

Related information: LINE.1 has the line number within the file of the focus line; LINE.2 has the line number within the file of the current line; LENGTH.1 has the length of the focus line; LINEFLAG.1, LINEFLAG.2, and LINEFLAG.3 have the flag

bits for the focus line; NBSCOPE.2 has the line number within the current scope of the focus line.

CURRBOX

Query CURRBox displays two values indicating whether display of a box around the current line is ON or OFF when the cursor is on the command line and when it is in the file area.

EXTract /CURRBox/ sets these variables:

currbox.0	2
currbox.1	ON|OFF (for box drawn when cursor is on command line)
currbox.2	ON|OFF (for box drawn when cursor is in file area)

CURSOR

Query CURSor displays information about the current location of the cursor: the line and column location in the window and also the line and column location within your file (or -1 and -1 if the cursor is not in the file area). The same four numbers are also given for the position the cursor occupied when the last SOS EXECUTE command issued began execution.

EXTract /CURSor/ sets these variables:

cursor.0	8
cursor.1	Line number of cursor in window
cursor.2	Column number of cursor in window
cursor.3	Line number of cursor in file, or -1
cursor.4	Column number of cursor in file, or -1
cursor.5	Previous line number of cursor in window
cursor.6	Previous column number of cursor in window
cursor.7	Previous line number of cursor in file, or -1
cursor.8	Previous column number of cursor in file, or -1

CURSORSIZE

Query CURSORSIze displays information about the size of the cursor, as a percentage of the height or width of a character in the current font. Four numbers are displayed: the width used for a vertical cursor when Overtype Mode and then when Insert Mode are in effect, and the height used for a horizontal cursor when Overtype Mode and then when Insert Mode are in effect.

EXTract /CURSORSIze/ sets these variables:

cursorsize.0	4
cursorsize.1	Width (as percentage of width of characters in current font) of vertical cursor in Overtype Mode
cursorsize.2	Width of vertical cursor in Insert Mode
cursorsize.3	Height of horizontal cursor in Overtype Mode
cursorsize.4	Height of horizontal cursor in Insert Mode

CURSORTYPE

Query CURSORType displays the type of cursor being used: VERTICAL, HORIZONTAL, or dependent on the INTERFACE setting.

EXTract /CURSORType/ sets these variables:

cursortype.0	1
cursortype.1	VERTICAL|HORIZONTAL|INTERFACE

DEBUGGING Query DEBUGGing displays whether or not the debugging window is ON or OFF, the height of the debugging window, and the trace setting for macros run with the DEBUG command.

EXTract /DEBUGGing/ sets these variables:

debugging.0	3
debugging.1	ON|OFF
debugging.2	Height of the debugging window in lines
debugging.3	Initial trace setting in effect for macros run with the DEBUG command

DEFEXT Query DEFEXT displays whether DEFEXT is ON or OFF.

EXTract /DEFEXT/ sets these variables:

defext.0	1
defext.1	ON|OFF

DEFPROFILE Query DEFPROFile displays the fileid of the default profile.

EXTract /DEFPROFile/ sets these variables:

defprofile.0	1
defprofile.1	Fileid of default profile

DEFSORT Query DEFSORT displays information on how DIR.DIR files are sorted: either OFF or some combination of DATE, EXTENSION, NAME, PATH, and SIZE.

EXTract /DEFSORT/ sets these variables:

defsort.0	1
defsort.1	OFF or one or more of DATE, EXTENSION, NAME, PATH, and SIZE

DIRECTORY Query DIRectory [*d:*] displays the name of the current directory of the specified drive. If no drive is specified, the current directory of the current drive is displayed.

EXTract /DIRectory/ sets these variables:

directory.0	2
directory.1	Current directory of current drive, in uppercase.
directory.2	Current directory of current drive, in mixed case

EXTract /DIRectory *d*:/ sets these variables:

directory.0	2
directory.1	Current directory of drive *d:*, in uppercase
directory.2	Current directory of drive *d:*, in mixed case

DIRFILEID

Query DIRFileid displays information about a file in a directory listing. The focus line is assumed to contain a file description. If the current file was created by the DIR command or has an extension of .DIR, the description is assumed to be in the usual DIR.DIR format. Otherwise, the focus line is assumed to have a complete fileid, starting in column 1. Information returned is the full fileid involved, and then each of the four components of the fileid: the drive specifier (with a trailing colon; a null string is returned if the fileid is a UNC name), the directory specification (with no trailing backslash added), the file name, and the file extension.

Note that invalid fileids yield unpredictable results, except that if the focus line is the top-of-file or end-of-file line or has a blank in column 1, null strings will always be returned for all values.

EXTract /DIRFileid/ sets these variables:

dirfileid.0	5
dirfileid.1	Full fileid
dirfileid.2	Drive specifier, or null string if fileid is a UNC name
dirfileid.3	Directory specification
dirfileid.4	File name
dirfileid.5	File extension

DIRFORMAT

Query DIRFORMat displays the number of columns used to display a file's name, extension, and year of last modification in a DIR.DIR file.

EXTract /DIRFORMat/ sets these variables:

dirformat.0	3
dirformat.1	The number of columns used to display the file's name
dirformat.2	The number of columns used to display the file's extension (or 0 if the name and extension are displayed in a single combined field)
dirformat.3	The number of columns used to display the year that the file was last modified

DISPLAY

Query DISPlay displays the minimum and maximum displayable selection levels.

EXTract /DISPlay/ sets these variables:

display.0	2
display.1	Minimum displayable selection level
display.2	Maximum displayable selection level

DOCSIZING Query DOCSIZing indicates whether new and cascaded document windows are sized using STANDARD or EXTENDED rules, and displays the column width used for document windows sized according to extended rules.

EXTract /DOCSIZing/ sets these variables:

docsizing.0	2
docsizing.1	STANDARD|EXTENDED
docsizing.2	Column width for document windows sized according to extended rules

DRAG Query DRAG displays the type of block marking or other operation invoked by dragging the mouse (LINE, BOX, STREAM, CMDLINE, DRAGDROP, or NONE), whether a block to be marked is PERSISTENT or a SELECTION, the ANCHOR or WORD option in effect, and RESET if dragging will reset any existing block.

EXTract /DRAG/ sets these variables:

drag.0	1 if DRAG NONE or DRAG DRAGDROP in effect, otherwise 4
drag.1	BOX|LINE|STREAM|CMDLINE|DRAGDROP|NONE
drag.2	PERSISTENT|SELECTION (this and the following items not set if DRAG NONE or DRAG DRAGDROP in effect)
drag.3	ANCHOR|WORD
drag.4	RESET or null string

ECOLOR Query ECOLOR *c* displays the ECOLOR setting for the specified item, which must be in the range A—Z or 1—9.

EXTract /ECOLOR *c*/ sets these variables:

ecolor.0	1
ecolor.1	The letter or number that you specified, followed by the corresponding color.

Query ECOLOR * or simply Query ECOLOR displays the ECOLOR setting for each of the characters, A—Z and 1—9, used with ECOLOR.

EXTract /ECOLOR/ or EXTract /ECOLOR */ sets the following variables:

ecolor.0	35
ecolor.*i*	*i*th character (A—Z or 1—9), followed by the corresponding color

EFILEID Query EFILEId displays the fileid that the current file had when it was added to the ring.

EXTract /EFILEId/ sets these variables:

efileid.0	2
efileid.1	Original fileid of current file, in uppercase
efileid.2	Original fileid of current file, in mixed case if FCASE ASIS in effect or lowercase if FCASE LOWER in effect

EOF

Query EOF displays whether or not the focus line location is ON or OFF the end-of-file (or end-of-range) line.

EXTract /EOF/ sets these variables:

eof.0	1
eof.1	ON|OFF

EOFIN

Query EOFIN displays the current setting of EOFIN: ALLOW or PREVENT.

EXTract /EOFIN/ sets these variables:

eofin.0	1
eofin.1	ALLOW|PREVENT

EOFOUT

Query EOFOUT displays the current setting of EOFOUT: EOL, EOLEOF, EOF, or NONE.

EXTract /EOFOUT/ sets these variables:

eofout.0	1
eofout.1	EOL|EOLEOF|EOF|NONE

EOL

Query EOL displays whether or not the focus column is located ON or OFF the end-of-line column (which is one column to the right of the right zone column, by analogy with the end-of-file line).

EXTract /EOL/ sets these variables:

eol.0	1
eol.1	ON|OFF

EOLIN

Query EOLIN displays the current setting for EOLIN: CR, CRORLF, LF, or NONE.

EXTract /EOLIN/ sets these variables:

eolin.0	1
eolin.1	CR|CRORLF|LF|NONE

EOLOUT Query EOLOUT displays the current setting for EOLOUT: CR, CRLF, LF, or NONE.

EXTract /EOLOUT/ sets these variables:

eolout.0	1
eolout.1	CR|CRLF|LF|NONE

FCASE Query FCASE displays whether KEDIT is keeping fileids in all LOWER case, or whether fileids are kept ASIS (“as is”).

EXTract /FCASE/ sets these variables:

fcase.0	1
fcase.1	ASIS|LOWER

FEXT Query FExt displays the extension of the current fileid.

EXTract /FExt/ sets these variables:

fext.0	2
fext.1	Extension of current fileid in uppercase
fext.2	Extension of current fileid, in mixed case if FCASE ASIS is in effect or lowercase if FCASE LOWER is in effect

FIELD Query FIELD displays the contents of the cursor field, the character at the cursor location, the location of the cursor within the data of the cursor field, and the type of field involved (COMMAND, TEXT, or PREFIX).

EXTract /FIELD/ sets these variables:

field.0	4
field.1	Contents of cursor field
field.2	Character at cursor location
field.3	Location of cursor within data of cursor field
field.4	COMMAND|TEXT|PREFIX

FIELDWORD Query FIELDWORD displays the word nearest the cursor location in the cursor field. Two interpretations of a word are displayed. For the first, KEDIT considers any group of consecutive alphanumeric characters, or any group of nonblank characters that are all non-alphanumeric, to be a word. For the second, KEDIT considers any group of nonblank characters to be a word.

EXTract /FIELDWORD/ sets these variables:

fieldword.0	2
fieldword.1	Field word consisting of any group of consecutive alphanumeric characters or a group of nonblank characters that are not alphanumeric
fieldword.2	Field word consisting of a group of nonblank characters

FILEID

Query FILEId displays the fileid of the current file: drive specifier, path, name, extension.

EXTract /FILEId/ sets these variables:

fileid.0	2
fileid.1	Fileid of current file, in uppercase
fileid.2	Fileid of current file, in mixed case if FCASE ASIS is in effect or lowercase if FCASE LOWER is in effect

FILESEARCH

When you use QUERY FILESEARCH *fileid*, KEDIT searches for *fileid* in the same way that it would if you issued a KEDIT command using that specified *fileid*. QUERY FILESEARCH returns information about the result of that search, but does not actually edit the specified file, as the KEDIT command would do.

QUERY FILESEARCH *fileid* returns ERROR if an error is encountered in searching for the file, RING if the specified file is already in the ring, DISK if the file is on disk and would be loaded into the ring by a KEDIT command, or NEW if the file does not already exist and would be edited as a new file.

When QUERY FILESEARCH *fileid* returns DISK, RING, or NEW, it also returns the fully-qualified name of the fileid involved, which can be affected by the path search KEDIT carries out for the file, and by settings like FCASE and DEFEXT.

For example, if SAMPLE.FIL is found in the current directory, the current directory is C:\TEST, and SAMPLE.FIL is not currently in the ring,

```
QUERY FILESEARCH SAMPLE.FIL
```

would return

```
DISK C:\TEST\SAMPLE.FIL
```

EXTract /FILESEARCH *fileid*/ sets these variables:

filesearch.0	1 if FILESEARCH.1 is ERROR, else 2
filesearch.1	DISK|RING|NEW|ERROR
filesearch.2	If FILESEARCH.1 is not ERROR, set to fully qualified fileid in the case (lower or mixed, as determined by the FCASE setting) that KEDIT would store the fileid internally

FILESTATUS

Query FILESTATUS displays three items:

The sharing mode under which the current file is locked: DENYWRITE or DENYREADWRITE, with NONE for a file that is not locked.

The type of access KEDIT has to the file: READONLY or READWRITE. KEDIT considers itself to have read-only access to a file if the file's directory entry was marked as read-only when KEDIT began editing the file. Otherwise, KEDIT considers itself to have read-write access to the file.

The third item is the end-of-line sequence KEDIT found at the end of the first line of the file when it read the file in. This can be CR, LF, or CRLF, indicating a carriage return, linefeed, or carriage return-linefeed pair. It can also be NONE, indicating that the file is a new file that did not exist on disk, that the file contained no end-of-line sequences, or that the file was read in with EOLIN NONE in effect.

EXTract /FILESTATUS/ sets these variables:

filestatus.0	3
filestatus.1	DENYWRITE|DENYREADWRITE|NONE
filestatus.2	READONLY|READWRITE
filestatus.3	CR|LF|CRLF|NONE

FLSCREEN

Query FLscreen displays the line numbers of the first and last lines of the file that are visible in the current window (or which would be visible if the display were refreshed).

EXTract /FLscreen/ sets these variables:

flscreen.0	2
flscreen.1	File line number of first file line in window
flscreen.2	File line number of last file line in window

FMODE

Query FMode displays the drive specifier of the current fileid as a drive letter followed by a colon, or displays NONE if the current fileid is a UNC name.

EXTract /FMode/ sets these variables:

fmode.0	2
fmode.1	Drive specifier of current fileid, in uppercase, or the null string if the current fileid is a UNC name.
fmode.2	Drive specifier of current fileid, in uppercase if FCASE ASIS is in effect or lowercase if FCASE LOWER is in effect, or the null string if the current fileid is a UNC name

FNAME

Query FName displays the file name of the current file.

EXTract /FName/ sets these variables:

fname.0	2
fname.1	File name of current file, in uppercase
fname.2	File name of current file, in mixed case if FCASE ASIS is in effect or lowercase if FCASE LOWER is in effect

FOCUSWORD

Query FOCUSWORD displays the word on the focus line nearest the cursor or nearest the current column if the cursor is not on the focus line. Two interpretations of a word are displayed. For the first, KEDIT considers any group of consecutive alphanumeric characters, or any group of nonblank characters that are all non-alphanumeric, to be a word. For the second, KEDIT considers any group of nonblank characters to be a word.

When the cursor is in the file area, QUERY FOCUSWORD returns the same information as QUERY FIELDWORD.

EXTract /FOCUSWORD/ sets these variables:

focusword.0	2
focusword.1	Focus word consisting of any group of consecutive alphanumeric characters or a group of nonblank characters that are not alphanumeric
focusword.2	Focus word consisting of a group of nonblank characters

FORMAT

Query FORMAT displays the values controlling paragraph justification (JUSTIFY or NOJUSTIFY), how paragraph boundaries are determined (BLANK or EXTENDED), and whether sentences are followed by SINGLE or DOUBLE blanks when reformatted.

EXTract /FORMAT/ sets these variables:

format.0	3
format.1	JUSTIFY|NOJUSTIFY
format.2	BLANK|EXTENDED
format.3	DOUBLE|SINGLE

FPATH

Query FPath displays the directory path of the current fileid. No trailing backslash is added to the directory path.

EXTract /FPath/ sets these variables:

fpath.0	2
fpath.1	Directory path of current fileid, in uppercase
fpath.2	Directory path of current fileid, in mixed case if FCASE ASIS is in effect or lowercase if FCASE LOWER is in effect

FTYPE

Query FType displays the extension of the current fileid.

EXTract /FType/ sets these variables:

ftype.0	2
ftype.1	Extension of current fileid, in uppercase
ftype.2	Extension of current fileid, in mixed case if FCASE ASIS is in effect or lowercase if FCASE LOWER is in effect

HELPDIR Query HELPDIR displays the Help file directory name if it has been specified via SET HELPDIR, or else displays *COMMAND.

EXTract /HELPDIR/ sets these variables:

helpdir.0	2
helpdir.1	SET HELPDIR directory name, in mixed case, or *COMMAND

HEX Query HEX displays whether HEX is ON or OFF.

EXTract /HEX/ sets these variables:

hex.0	1
hex.1	ON|OFF

HEXDISPLAY Query HEXDISPlay displays whether HEXDISPLAY is ON or OFF.

EXTract /HEXDISPlay/ sets these variables:

hexdisplay.0	1
hexdisplay.1	ON|OFF

HIGHLIGHT Query HIGHlight displays the current setting of HIGHLIGHT: OFF, ALTERED, SELECT *n m*, or TAGGED.

EXTract /HIGHlight/ sets these variables:

highlight.0	3 if HIGHLIGHT SELECT is in effect; otherwise 1
highlight.1	OFF|ALTERED|SELECT|TAGGED
highlight.2	If HIGHLIGHT SELECT is in effect, the lowest selection level highlighted
highlight.3	If HIGHLIGHT SELECT is in effect, the highest selection level highlighted

IDLINE Query IDline displays whether IDLINE is ON or OFF.

EXTract /IDline/ sets these variables:

idline.0	1
idline.1	ON|OFF

IMPMACRO Query IMPMACro displays whether IMPMACRO is ON or OFF.

EXTract /IMPMACro/ sets these variables:

impmacro.0	1
impmacro.1	ON|OFF

INISAVE Query INISAVE displays whether state information will be saved in the Windows registryat the end of the editing session (STATE|NOSTATE) and whether history information will be saved (HISTORY|NOHISTORY).

EXTract /INISAVE/ sets these variables:

inisave.0	2
inisave.1	STATE|NOSTATE
inisave.2	HISTORY|NOHISTORY

QUERY/EXTRACT REGSAVE, which returns the same information as QUERY/EXTRACT INISAVE, is now the preferred form.

INITIALDIR Query INITIALDIR displays how the current directory is determined at the start of an editing session when no initial fileid is specified (PRESERVE or RECALL) and when an initial fileid is specified (PRESERVE, RECALL, or FIRSTFILE).

EXTract /INITIALDIR/ sets these variables:

initialdir.0	2
initialdir.1	PRESERVE|RECALL
initialdir.2	PRESERVE|RECALL|FIRSTFILE

INITIALDOCSIZE Query INITIALDOCsize displays whether the initial document window created at the start of a session is MAXIMIZED, NORMAL (non-maximized) or RECALL (maximized or non-maximized depending on the state of the last document window in the preceding editing session).

EXTract /INITIALDOCsize/ sets these variables:

initialdocsize.0	1
initialdocsize.1	MAXIMIZED|NORMAL|RECALL

INITIALFRAMESIZE Query INITIALFRAMEsize displays whether KEDIT's frame window is initially MAXIMIZED, NORMAL (non-maximized) or RECALL (maximized or non-maximized depending on its state at the end of the preceding editing session).

EXTract /INITIALFRAMEsize/ sets these variables:

initialframesize.0	1
initialframesize.1	MAXIMIZED|NORMAL|RECALL

INITIALINSERT Query INITIALINSert displays whether Insert Mode is ON or OFF at the start of a KEDIT session.

EXTract /INITIALINSert/ sets these variables:

initialinsert.0	1
initialinsert.1	ON|OFF

INITIALWIDTH Query INITIALWidth displays the WIDTH value that KEDIT puts into effect by default at the start of an editing session.

EXTract /INITIALWidth/ sets these variables:

initialwidth.0	1
initialwidth.1	INITIALWIDTH value

INPUTMODE Query INPUTMode displays the current setting of INPUTMODE: OFF, FULL, or LINE.

EXTract /INPUTMode/ sets these variables:

inputmode.0	1
inputmode.1	OFF|FULL|LINE

INSERTMODE Query INSERTmode displays whether INSERTMODE is ON or OFF.

EXTract /INSERTmode/ sets these variables:

insertmode.0	1
insertmode.1	ON|OFF

INSTANCE Query INSTANCE displays the current setting of INSTANCE: SINGLE if only a single instance of KEDIT will run at a time, or else MULTIPLE.

EXTract /INSTANCE/ sets these variables:

instance.0	1
instance.1	SINGLE|MULTIPLE

INTERFACE Query INTERFACE displays the type of interface conventions in effect, either CUA or CLASSIC.

EXTract /INTERFACE/ sets these variables:

interface.0	1
interface.1	CUA|CLASSIC

INTERNATIONAL Query INTERNATional displays the values controlling whether international case conventions are in use (CASE or NOCASE) and whether international sorting conventions are in use (SORT or NOSORT).

EXTract /INTERNATional/ sets these variables:

international.0	2
international.1	CASE|NOCASE
international.2	SORT|NOSORT

KEYSTYLE

Query KEYSTYLE displays whether the STANDARD or ADJUSTED behavior of the of the Enter, Home, Delete, Backspace, and Alt keys will be used when INTERFACE CUA is in effect.

EXTract /KEYSTYLE/ sets these variables:

keystyle.0	5
keystyle.1	STANDARD|ADJUSTED (Enter key)
keystyle.2	STANDARD|ADJUSTED (Home key)
keystyle.3	STANDARD|ADJUSTED (Delete key)
keystyle.4	STANDARD|ADJUSTED (Backspace key)
keystyle.5	STANDARD|ADJUSTED (Alt key)

The default definitions for these keys test the relevant KEYSTYLE setting and use it to decide how they will operate. In fact, none of KEDIT's ''hard-coded'' behavior depends on the value of KEYSTYLE; KEYSTYLE is only used within these default key definitions.

LASTKEY

Query LASTKEY displays information about the last key read from the keyboard by KEDIT's keyboard handler. (This is normally the key that KEDIT is currently processing. For example, if you press the F4 key and the macro assigned to that key issues the QUERY LASTKEY command, it is F4 that is reported on.) QUERY LASTKEY does not report on keys read with the READV CMDLINE, DIALOG, or ALERT commands, but does report on keys read via READV KEY.

KEDIT gives you the name of the last key read (as described in Chapter 7, "Built-in Macro Handling"), the character associated with the key (or a null string for function keys, etc.), the scan code (a decimal number from 0 to 255) of the key, and the Shift Status at the time KEDIT read the key. See the description of the READV KEY command for more information on the information returned.

Query LASTKEY *n*, where *n* can range from 1 to 8, gives similar information for the *n*th most recently read key. That is, Query LASTKEY 1 gives information on the most recently read key (this is the same as Query LASTKEY), Query LASTKEY 2 gives information on the second most recently read key, etc.

If no keys (or not enough keys) have been read yet by KEDIT, null strings are returned for the key name and character code 0 is returned for the scan code and Shift Status.

EXTract /LASTKEY [*n*]/ sets these variables:

lastkey.0	5 under Windows XP/2000/Vista; 4 under Windows 98/Me
lastkey.1	Key name (in uppercase, with possible "C-", "S-", "A-", "S-C-", or"A-C-" prefix; "-" is always used in the prefix rather than "+" for compatibility with earlier versions of KEDIT)
lastkey.2	Character (or null string)
lastkey.3	Scan code

lastkey.4	Shift Status (see the table given with the READV command on page 111)
lastkey.5	Extended Shift Status (see the table given with the READV command on page ; not set on Windows 98/Me)

LASTMSG

Query LASTmsg displays the text of the last message or error message generated for display in the current window. If MSGMODE OFF is in effect, the message may not actually have been displayed. KEDIT truncates the text of messages that are longer than 160 characters.

EXTract /LASTmsg/ sets these variables:

lastmsg.0	1
lastmsg.1	Text of last message, in mixed case

LASTOP

Query LASTOP *command* displays the operand used when *command* was last issued from the command line. The *command* can be ALter, Change, CLocate, COUnt, Find, Locate, SCHange, or TFind.

EXTract /LASTOP *command*/ sets these variables:

lastop.0	1
lastop.1	name of the command and, in mixed case, its last operand

Query LASTOP * or simply QUERY LASTOP displays a list of the names and last operands for all eight commands that have remembered operands.

EXTract /LASTOP */ sets these variables:

lastop.0	8
lastop.1	ALTER and, in mixed case, its last operand
lastop.2	CHANGE and, in mixed case, its last operand
lastop.3	CLOCATE and, in mixed case, its last operand
lastop.4	COUNT and, in mixed case, its last operand
lastop.5	FIND and, in mixed case, its last operand
lastop.6	LOCATE and, in mixed case, its last operand
lastop.7	SCHANGE and, in mixed case, its operand
lastop.8	TFIND and, in mixed case, its last operand

LASTRC

Query LASTRC displays the return code generated by the last command issued from the command line (that is, the return code in effect when an SOS EXECUTE command last completed execution).

EXTract /LASTRC/ sets these variables:

lastrc.0	1
lastrc.1	Return code from last command line command. (The macro variable *RC*, set after each command to the return code from that command, is more frequently useful.)

LENGTH Query LENgth displays the length of the focus line, with trailing blanks ignored.

EXTract /LENgth/ sets these variables:

length.0	1
length.1	Length of focus line

LINE Query LIne displays the line number within the current file of the focus line.

EXTract /LIne/ sets these variables:

line.0	2
line.1	Line number within current file of focus line. (See NBSCOPE.2 for line number within current scope)
line.2	Line number within current file of the current line

LINEFLAG Query LINEFLAG displays the flags associated with the focus line: NEW|NONEW, CHANGE|NOCHANGE, and TAG|NOTAG.

EXTract /LINEFLAG/ sets these variables:

lineflag.0	3
lineflag.1	NEW|NONEW
lineflag.2	CHANGE|NOCHANGE
lineflag.3	TAG|NOTAG

LINEND Query LINENd displays whether LINEND is ON or OFF, and displays the linend character.

EXTract /LINENd/ sets these variables:

linend.0	2
linend.1	ON|OFF
linend.2	Linend character

LOCKING Query LOCKING displays whether LOCKING is ON or OFF.

EXTract /LOCKING/ sets these variables:

locking.0	1
locking.1	ON|OFF

LRECL Query LRecl displays the LRECL (logical record length) value.

EXTract /LRecl/ sets these variables:

lrecl.0	1
lrecl.1	LRECL value

LSCREEN Query LScreen displays the height in lines and width in columns of the current window.

EXTract /LScreen/ sets these variables:

lscreen.0	4
lscreen.1	Height in lines of current document window
lscreen.2	Width in columns of current document window
lscreen.3	Serial number of current document window (same as UNIQUEID.3)
lscreen.4	Serial number of current document window (same as UNIQUEID.3)

LSCREEN.3 and LSCREEN.4 are supplied for compatibility with text mode KEDIT. In text mode KEDIT these give the line and column of the upper left corner of the window. This specific information is not relevant under Windows, but some macros use it only to uniquely identify a particular window. Returning the window's serial number in LSCREEN.3 and LSCREEN.4 lets these macros work without change in KEDIT for Windows.

MACRO Query MACRO *macroname* displays the definition of the specified in-memory macro.

EXTract /MACRO/ is not currently supported.

MACROPATH Query MACROPath displays ON, OFF, the name of the environment variable used for macro searches, or the directory list used for macro searches.

EXTract /MACROPath/ sets these variables:

macropath.0	1
macropath.1	ON|OFF|*envvar*|*dirlist*

MACROSEARCH Query MACROSEARCH *fileid* [*defaultextension*] causes KEDIT to look for a file in the same directories it looks in when searching for a macro. It then displays the fully qualified name of the resulting file, or the null string if an error is encountered or the file can't be found. If *fileid* has no extension then *defaultextension*, if specified, is assumed.

EXTract /MACROSEARCH *fileid* [*defaultextension*]/ sets these variables:

macrosearch.0	1
macrosearch.1	Fully qualified file name, or the null string

MARGINS Query MARgins displays the current margin settings: left margin, right margin, and paragraph indent.

EXTract /MARgins/ sets these variables:

margins.0	3
margins.1	Left margin column

margins.2	Right margin column
margins.3	Paragraph indent value

MARKSTYLE Query MARKSTYLE displays whether line blocks, box blocks, and stream blocks marked with the mouse under INTERFACE CUA are marked as persistent blocks or as selections.

EXTract /MARKSTYLE/ sets these variables:

markstyle.0	3
markstyle.1	SELECTION|PERSISTENT (Line blocks)
markstyle.2	SELECTION|PERSISTENT (Box blocks)
markstyle.3	SELECTION|PERSISTENT (Stream blocks)

KEDIT's default mouse macros test the relevant MARKSTYLE setting and use it to decide how they will operate. In fact, none of KEDIT's ''hard-coded'' behavior depends on the value of MARKSTYLE; MARKSTYLE is only used within these default mouse macros.

MEMORY Query MEMory displays information about KEDIT's memory usage.

Seven values are displayed; most of them are present only for compatibility with earlier versions of KEDIT.

EXTract /MEMory/ sets these variables, with all values in kilobytes:

memory.0	7
memory.1	0
memory.2	0
memory.3	Always 10 (for compatibility with earlier versions of KEDIT)
memory.4	Approximate amount of memory, in kilobytes, being used by KEDIT to hold the contents of your files and related internal data structures.
memory.5	0
memory.6	0
memory.7	Approximate amount of memory, in kilobytes, holding undo information

MONITOR Query MONitor displays whether MONITOR is set to WINDOWS, COLOR, or MONO.

EXTract /MONitor/ sets these variables:

monitor.0	1
monitor.1	WINDOWS|COLOR|MONO

MOUSEBEEP Query MOUSEBEEP displays whether MOUSEBEEP is ON or OFF.

EXTract /MOUSEBEEP/ sets these variables:

mousebeep.0	1
mousebeep.1	ON|OFF

MSGLINE

Query MSGLine displays whether MSGLINE is ON or OFF (it is currently always ON), your specification for the first line used for messages, the number of lines that can be used for messages, and OVERLAY if the first message line can overlay a file line.

EXTract /MSGLine/ sets these variables:

msgline.0	4
msgline.1	ON
msgline.2	Message line location
msgline.3	Number of message lines
msgline.4	OVERLAY|null string

MSGMODE

Query MSGMode displays whether MSGMODE is ON or OFF.

EXTract /MSGMode/ sets these variables:

msgmode.0	1
msgmode.1	ON|OFF

NBFILE

Query NBFile displays both the number of files in the ring and the maximum number of files allowed in the ring.

EXTract /NBFile/ sets these variables:

nbfile.0	2
nbfile.1	Number of files in the ring
nbfile.2	The maximum number of files allowed in the ring (currently 500)

NBSCOPE

Query NBScope displays the number of lines within the current scope (which is the number of lines in the current range if SCOPE ALL is in effect, and the number of lines selected for display if SCOPE DISPLAY is in effect), and the line number within the current scope of the focus line.

EXTract /NBScope/ sets these variables:

nbscope.0	2
nbscope.1	Number of lines in current scope
nbscope.2	Line number of focus line within current scope

NBWINDOW Query NBWindow displays the number of document windows that currently exist.

EXTract /NBWindow/ sets these variables:

nbwindow.0	1
nbwindow.1	Number of document windows

NEWLINES Query NEWLines displays where new lines are input into your file: SAMELINE, BELOW, or BELOWCURR.

EXTract /NEWLines/ sets these variables:

newlines.0	1
newlines.1	SAMELINE|BELOW|BELOWCURR

NOVALUE Query NOVALUE displays ON or OFF, depending on whether use of uninitialized variables in KEXX macros causes an error.

EXTract /NOVALUE/ sets these variables:

novalue.0	1
novalue.1	ON|OFF

NUMBER Query NUMber displays whether display of line numbers in the prefix area is ON or OFF.

EXTract /NUMber/ sets these variables:

number.0	1
number.1	ON|OFF

OFPW Query OFPW displays ON or OFF, depending on whether ''one-file-per-window'' mode, in which each file added to the ring gets its own document window, is in effect.

EXTract /OFPW/ sets these variables:

ofpw.0	1
ofpw.1	ON|OFF

OPENFILTER Query OPENFilter displays the current open filter string, which can be used in the File Open dialog box to determine the types of files listed.

EXTract /OPENFilter/ sets these variables:

openfilter.0	1
openfilter.1	Open filter string

OPMODE Query OPMODE displays information on the mode in which KEDIT is running. The result can be: FULLSCREEN (OS/2 KEDIT in fullscreen session), TEXTWINDOW (OS/2 KEDIT in Presentation Manager text window), DETACHED (OS/2 KEDIT in detached session), REAL (DOS KEDIT), WINDOWS (DOS KEDIT running under

Windows 3.0 or above), GUI (Graphical-User-Interface version of KEDIT, in particular KEDIT for Windows), or UNKNOWN. For the GUI version of KEDIT, additional information is displayed: the windowing system involved (for KEDIT for Windows, this is "Windows"), the version of the windowing system (for example, under Windows Vista, 6.00), and possibly some additional identifying text.

EXTract /OPMODE/ sets these variables:

opmode.0	4 (for KEDIT for Windows) or 1 (for text mode versions of KEDIT)
opmode.1	GUI (for KEDIT for Windows); or else FULLSCREEN|TEXTWINDOW|DETACHED|REAL|WINDOWS|UNKNOWN
opmode.2	WINDOWS (for KEDIT for Windows); this and the following values are not set in text mode versions of KEDIT
opmode.3	Windows version (for example, 6.00)
opmode.4	Additional identifying text or null string

A macro can determine whether it is running under a graphical version of KEDIT by testing whether OPMODE.1 equals GUI. Once it has been established that OPMODE.1 is GUI, OPMODE.2, OPMODE.3, and OPMODE.4 can be checked to get further information about the windowing system involved. Note that OPMODE.2, OPMODE.3, and OPMODE.4 are not set in text mode versions of KEDIT, so macros intended to run under both text mode and GUI versions of KEDIT should reference them only after establishing that OPMODE.1 equals GUI. Note also that if OPMODE.1 equals WINDOWS, it does *not* mean that you are running KEDIT for Windows; it means that the DOS version of KEDIT is running under Windows.

If you simply want to test from within a macro whether you are running under KEDIT for Windows, we recommend that you test whether VERSION.1 is equal to ''KEDIT/WINDOWS''.

OPSYS

Query OPSYS displays the name of the operating system KEDIT is running under and the operating system version number. For example, running under Windows Vista, KEDIT would display "Windows Vista 6.00".

EXTract /OPSYS/ sets these variables:

opsys.0	3
opsys.1	Name of operating system
opsys.2	Version number of operating system
opsys.3	Additional identifying text or null string

If you want to test in a macro whether you are running under KEDIT for Windows, we recommend that you test whether VERSION.1 is equal to ''KEDIT/WINDOWS''.

PARSER

Query PARSER *parser* displays the fileid of the KEDIT Language Definition file from which the specified parser was loaded, as specified on the SET PARSER command that defined the parser.

EXTract /PARSER *parser*/ sets these variables:

parser.0	2
parser.1	Parser name
parser.2	Fileid of .KLD file

Query PARSER * or simply Query PARSER displays, for each defined parser, the name of the parser and the fileid of the associated KLD file.

EXTract /PARSER/ or EXTRACT /PARSER */ sets the following variables:

parser.0	Number of parsers currently defined
parser.*i*	Name of *i*th parser, and fileid of associated KLD file

PATH

Query PATH displays ON, OFF, the name of the environment variable used for file searches, or the directory list used for file searches.

EXTract /PATH/ sets these variables:

path.0	1
path.1	ON|OFF|*envvar*|*dirlist*

PCOLOR

Query PCOLOR *c* displays the PCOLOR setting for the specified item, which must be in the range A—Z or 1—9.

EXTract /PCOLOR *c*/ sets these variables:

pcolor.0	1
pcolor.1	The letter or number that you specified, followed by the corresponding color.

Query PCOLOR * or simply Query PCOLOR displays the PCOLOR setting for each of the characters, A—Z and 1—9, used with PCOLOR.

EXTract /PCOLOR/ or EXTract /PCOLOR */ sets the following variables:

pcolor.0	35
pcolor.*i*	*i*th character (A—Z or 1—9), followed by the corresponding color

POINT

Query Point displays names currently assigned to the focus line.

EXTract /Point/ sets these variables:

point.0	0 if focus line is not a named line; otherwise, 1
point.1	Line number and names of the focus line, if focus line is named

Query Point * displays the line number and assigned names of all named lines in the current file.

EXTract /Point */ sets these variables:

point.0	Number of named lines
point.*i*	Line number and names of the *i*th named line

PREFIX

Query PREfix displays whether the prefix area is ON or OFF or displaying NULLS, and whether the prefix area displays on the LEFT or RIGHT of the window.

EXTract /PREfix/ sets these variables:

prefix.0	2
prefix.1	ON|OFF|NULLS
prefix.2	LEFT|RIGHT

Query PREfix Synonym displays, for each defined prefix synonym, the newname and oldname of the prefix command.

EXTract /PREfix Synonym/ sets these variables:

prefix.0	Number of prefix command synonyms
prefix.*i*	Newname and oldname for *i*th prefix synonym

PREFIXWIDTH

Query PREFIXWIDTH displays the width of the prefix area.

EXTract /PREFIXWIDTH/ sets these variables:

prefixwidth.0	1
prefixwidth.1	Width of prefix area

PRINTCOLORING

Query PRINTCOLORing displays whether PRINTCOLORING is ON or OFF.

EXTract /PRINTCOLORing/ sets these variables:

printcoloring.0	1
printcoloring.1	ON|OFF

PRINTER

Query PRINTER displays whether printer output is being sent to WINDOWS, LPT1:, LPT2:, LPT3:, COM1:, or COM2:. It also displays whether the CLOSE|NOCLOSE option, the FORM|NOFORM option, and the CONVERT|NOCONVERT options are in effect.

EXTract /PRINTER/ sets these variables:

printer.0	4
printer.1	WINDOWS|LPT1:|LPT2:|LPT3:|COM1:|COM2:
printer.2	CLOSE|NOCLOSE
printer.3	FORM|NOFORM
printer.4	CONVERT|NOCONVERT

PRINTPROFILE Query PRINTPROFile displays the fileid of the default printer profile macro.

EXTract /PRINTPROFile/ sets these variables:

printprofile.0	1
printprofile.1	Fileid of default printer profile macro

PRINTSIZE Query PRINTSIZE displays the number of lines per page, and the number of columns per page, used for print operations, based on the currently selected printer, font, and margins. If KEDIT cannot determine the result, -1 will be returned for the lines and/or columns. -1 will always be returned if PRINTER WINDOWS is not in effect, and can also be returned if, for example, no printers are installed.

EXTract /PRINTSIZE/ sets these variables:

printsize.0	2
printsize.1	Number of lines per printed page, or -1
printsize.2	Number of columns per printed page, or -1

QUICKFIND Query QUICKFIND displays information on the search string displayed in the Quick Find toolbar item: whether, if that string is searched for, case will be respected or ignored, whether the search will be limited to whole words only, and whether the string is a regular expression, followed by the string itself.

EXTract /QUICKFIND/ sets these variables:

quickfind.0	4
quickfind.1	IGNORE|RESPECT
quickfind.2	WORD|NOWORD
quickfind.3	REGEXP|NOREGEXP
quickfind.4	Quick Find search string

RANGE Query RANge displays the line numbers within the current file of the first and last lines in the current range.

EXTract /RANge/ sets these variables:

range.0	2
range.1	Line number of first line in range
range.2	Line number of last line in range. (Equal to one less than RANGE.1 if no lines in current range)

RECENTFILES Query RECENTFiles the maximum number of recently-edited files that KEDIT will display on the File menu.

EXTract /RECENTFiles/ sets these variables:

recentfiles.0	1
recentfiles.1	RECENTFILES value

RECFM Query RECFm displays whether the current record format is FIXED or VARYING.

EXTract /RECFm/ sets these variables:

recfm.0	1
recfm.1	FIXED|VARYING

REPROFILE Query REPROFile displays whether REPROFILE is ON or OFF.

EXTract /REPROFile/ sets these variables:

reprofile.0	1
reprofile.1	ON|OFF

REGSAVE Query REGSAVE displays whether state information will be saved in the Windows registryat the end of the editing session (STATE|NOSTATE) and whether history information will be saved (HISTORY|NOHISTORY).

EXTract /REGSAVE/ sets these variables:

regsave.0	2
regsave.1	STATE|NOSTATE
regsave.2	HISTORY|NOHISTORY

RESERVED Query RESERved displays a list of lines within the current window occupied by reserved lines.

EXTract /RESERved/ sets these variables:

reserved.0	0 if no reserved lines; otherwise 1
reserved.1	List of reserved line numbers, if any

RIGHTCTRL Query RIGHTCTRL displays whether RIGHTCTRL is ON or OFF.

EXTract /RIGHTCTRL/ sets these variables:

rightctrl.0	1
rightctrl.1	ON|OFF

RING Query RING displays the number of files in the ring and gives information on each file's fileid, current line, current column, size, and alteration counts.

EXTract /RING/ sets these variables:

ring.0	Number of files in the ring
ring.*i*	Information on the *i*th file in the ring: fileid (in uppercase and possibly containing blanks), current line, current column, size, and alteration count

Query RING FILEID displays the number of files in the ring and the fileid of each of the files in the ring.

EXTract /RING FILEID/ sets these variables:

ring.0	Number of files in the ring
ring.*i*	The fileid of the *i*th file in the ring, in mixed case and possibly containing blanks.

QUERY/EXTRACT RING and QUERY/EXTRACT RING FILEID consider the current file to be the first file in the ring.

SCALE

Query SCALe displays whether SCALE is ON or OFF, and the scale line's specified location in the window.

EXTract /SCALe/ sets these variables:

scale.0	2 if SCALE OFF, 3 if SCALE ON
scale.1	ON|OFF
scale.2	Scale line location specification
scale.3	Line number within window occupied by scale line (set only if SCALE ON)

SCOPE

Query SCOPE displays the current scope setting, either DISPLAY or ALL.

EXTract /SCOPE/ sets these variables:

scope.0	1
scope.1	DISPLAY|ALL

SCROLLBAR

Query SCROLLbar displays whether SCROLLBAR is ON or OFF and whether, if SCROLLBAR ON is in effect, the HORIZONTAL scrollbar only, the VERTICAL scrollbar only, or BOTH kinds are displayed.

EXTract /SCROLLbar/ sets these variables:

scrollbar.0	2
scrollbar.1	ON|OFF
scrollbar.2	HORIZONTAL|VERTICAL|BOTH

SELECT

Query SELect displays the selection level of the focus line, and the maximum selection level of all lines in the current file.

EXTract /SELect/ sets these variables:

select.0	2
select.1	Selection level of focus line. (Same value as CURLINE.6, but using CURLINE.6 is much more efficient)
select.2	Maximum selection level of all lines in current range in current file

SHADOW Query SHADow displays whether SHADOW is ON or OFF.

EXTract /SHADow/ sets these variables:

shadow.0	1
shadow.1	ON|OFF

SHARING Query SHARING displays the file sharing modes used by KEDIT when it reads a file into memory (DENYWRITE or DENYNONE), and when it locks a file (DENYWRITE or DENYREADWRITE).

EXTract /SHARING/ sets these variables:

sharing.0	2
sharing.1	DENYWRITE|DENYNONE
sharing.2	DENYWRITE|DENYREADWRITE

SIZE Query SIze displays the number of lines in the current file.

EXTract /SIze/ sets these variables:

size.0	1
size.1	Number of lines in current file. (See NBSCOPE.1 for number of lines in current scope)

STARTUP Query STARTUP displays information about how KEDIT was invoked:

The fully qualified name of the KEDIT module that is executing

The command line arguments passed to KEDIT or, if there were none, ‘‘(none)’’

The value of the KEDITW environment variable, or ‘‘(none)’’

The fully qualified name of the profile executed at the start of the current editing session, or ‘‘(none)’’

EXTract /STARTUP/ sets these variables:

startup.0	4
startup.1	The fully qualified name of the KEDIT module that is executing
startup.2	The command line arguments passed to KEDIT, or the null string
startup.3	The value of the KEDITW environment variable, or the null string
startup.4	The fully qualified name of the initial profile, or the null string

STATUSLINE Query STATUSLine displays whether the status line display is ON or OFF.

EXTract /STATUSLine/ sets these variables:

statusline.0	2
statusline.1	ON|OFF
statusline.2	BOTTOM

STAY Query STAY displays whether STAY is ON or OFF.

EXTract /STAY/ sets these variables:

stay.0	1
stay.1	ON|OFF

STREAM Query STReam displays whether STREAM is ON or OFF.

EXTract /STReam/ sets these variables:

stream.0	1
stream.1	ON|OFF

SYNONYM Query SYNonym displays whether SYNONYM is ON or OFF.

EXTract /SYNonym/ sets these variables:

synonym.0	1
synonym.1	ON|OFF

Query SYNonym * displays, for all defined synonyms, the newname, the minimal truncation, and the synonym definition.

EXTract /SYNonym */ sets these variables:

synonym.0	Number of synonyms defined
synonym.*i*	Newname, truncation, and definition of *i*th synonym

TABLINE Query TABLine displays whether TABLINE is ON or OFF, and its specified location in the window.EXTract /TABLine/ sets these variables:

tabline.0	2 if TABLINE OFF, 3 if TABLINE ON
tabline.1	ON|OFF
tabline.2	Tabline location specification
tabline.3	Line number within window occupied by tab line (set only if TABLINE ON)

TABS

Query TABs displays the current tab setting (a list of specific tab columns, INCR *n*, or both)

EXTract /TABs/ sets these variables:

tabs.0	2
tabs.1	Current tab setting (a list of specific tab columns, INCR *n*, or both)
tabs.2	Current tab columns (up to a maximum response length of approximately 1000 characters)

TABSAVE

Query TABSAVE displays whether the TABSAVE facility is ON or OFF.

EXTract /TABSAVE/ sets these variables:

tabsave.0	1
tabsave.1	ON|OFF

TABSIN

Query TABSIn displays whether TABSIN is ON, OFF, or set to TABQUOTE, and the tab increment used for TABSIN processing.

EXTract /TABSIn/ sets these variables:

tabsin.0	2
tabsin.1	ON|OFF|TABQUOTE
tabsin.2	Tab increment

TABSOUT

Query TABSOut displays whether TABSOUT is ON or OFF, and the tab increment used for TABSOUT processing.

EXTract /TABSOut/ sets these variables:

tabsout.0	2
tabsout.1	ON|OFF
tabsout.2	Tab increment

TARGET

Query TARGet displays information about the target of the last CLOCATE, LOCATE, or TFIND command issued for any file in the ring: the line and column number within the file of the start of the matching target, and the line and column number of the end of the matching target. If the target was not a string target, the column pointer value is returned for both column numbers. In the current version of KEDIT, the line number of the start and end of the matching target will always be the same.

EXTract /TARGet/ sets these variables:

target.0	5
target.1	Line number of start of target
target.2	Column number of start of target

target.3	Line number of end of target
target.4	Column number of end of target
target.5	Set equal to the text of the last string target. Valid only if your macro issues EXTRACT /TARGET/ immediately (that is, no intervening KEDIT commands) after successful use of the LOCATE, CLOCATE, or TFIND commands to search for a string target

THIGHLIGHT Query THIGHlight displays whether THIGHLIGHT is ON or OFF.

EXTract /THIGHlight/ sets these variables:

thighlight.0	1
thighlight.1	ON|OFF

TIME Query TIME displays the current date and time.

EXTract /TIME/ sets these variables:

time.0	5
time.1	Date in country-dependent format
time.2	Time in hh:mm format
time.3	Date in mm-dd-yy format
time.4	Time in hh:mm:ss.hh format
time.5	Date in yyyy-mm-dd format

TIMECHECK Query TIMECHECK displays whether TIMECHECK is ON or OFF.

EXTract /TIMECHECK/ sets these variables:

timecheck.0	1
timecheck.1	ON|OFF

TOF Query TOF displays whether the focus line location is ON or OFF the top-of-file (or top-of-range) line.

EXTract /TOF/ sets these variables:

tof.0	1
tof.1	ON|OFF

TOFEOF Query TOFEOF displays whether display of the top-of-file and end-of-file lines is ON or OFF.

EXTract /TOFEOF/ sets these variables:

tofeof.0	1
tofeof.1	ON|OFF

TOL

Query TOL displays whether or not the focus column is located ON or OFF the top-of-line column (which is one column to the left of the left zone column, by analogy with the top-of-file line).

EXTract /TOL/ sets these variables:

tol.0	1
tol.1	ON|OFF

TOOLBAR

Query TOOLBAR displays toolbar display is ON or OFF, and whether toolbars are set to appear at the TOP of the frame window, the BOTTOM, or BOTH.

EXTract /TOOLBAR/ sets these variables:

toolbar.0	2
toolbar.1	ON|OFF
toolbar.2	TOP|BOTTOM|BOTH

TOOLBUTTON

Query TOOLButton *name* displays information about the specified toolbutton: the button name, how it appears on the toolbar (bitmap name or delimited text string), any conditions under which the button is disabled, and the tooltip and status line help for the button.

EXTract /TOOLButton *name*/ sets these variables:

toolbutton.0	4
toolbutton.1	Button name
toolbutton.2	Built-in or on-disk bitmap file, or delimited string used for button text
toolbutton.3	COND *ccc* (conditions under which button is disabled, if any), or null string
toolbutton.4	Delimited text of tooltip and status line help for button, if any, or null string

Query TOOLButton * or simply Query TOOLButton displays a line of information for each of the currently-defined toolbuttons: the button name, bitmap name or delimited text string, conditions under which button is disabled, delimited text of tooltip and status line help.

EXTract /TOOLButton */ or simply EXTract /TOOLButton/ sets these variables:

toolbutton.0	Number of toolbuttons
toolbutton.*i*	Information about the *i*th toolbutton

TOOLSET Query TOOLSet or Query TOOLSet TOP gives information about the top toolbar; Query TOOLSet BOTTOM gives information about the bottom toolbar. TOP or BOTTOM is displayed, followed by DEFAULT or USER, followed by the contents of the toolset.

EXTract /TOOLSet [TOP|BOTTOM]/ (where TOP is the default) sets these variables:

toolset.0	3
toolset.1	TOP|BOTTOM
toolset.2	DEFAULT|USER
toolset.3	Contents of default or user-defined toolset

TRAILING Query TRAILING displays the current setting of TRAILING: ON, OFF, SINGLE, or EMPTY.

EXTract /TRAILING/ sets these variables:

trailing.0	1
trailing.1	ON|OFF|SINGLE|EMPTY

TRANSLATEIN Query TRANSLATEIn displays the current setting of TRANSLATEIN, which is either NONE or OEMTOANSI.

EXTract /TRANSLATEIn/ sets these variables:

translatein.0	1
translatein.1	NONE|OEMTOANSI

TRANSLATEOUT Query TRANSLATEOut displays the current setting of TRANSLATEOUT, which is either NONE or ANSITOOEM.

EXTract /TRANSLATEOut/ sets these variables:

translateout.0	1
translateout.1	NONE|ANSITOOEM

TRUNC Query TRunc displays the column number of the truncation column.

EXTract /TRunc/ sets these variables:

trunc.0	1
trunc.1	Truncation column

UNDO

Query UNDO displays, for the current file, how many levels of changes can be undone, how many levels of changes can be redone, and the amount of storage (in kilobytes) being used to hold the file's undo information.

EXTract /UNDO/ sets these variables:

undo.0	3
undo.1	Levels of undoable changes for current file
undo.2	Levels of redoable changes for current file
undo.3	Amount of memory, in kilobytes, holding undo information for current file

UNDOING

Query UNDOING displays whether UNDOING is ON or OFF, the maximum number of undo levels that will be kept when it is ON, and the maximum amount of memory (in Kbytes) that KEDIT will set aside to hold undo information.

EXTract /UNDOING/ sets these variables:

undoing.0	3
undoing.1	ON|OFF
undoing.2	The maximum number of undo levels KEDIT will attempt to keep in memory
undoing.3	The maximum amount of memory (in Kbytes) that KEDIT will use to hold undo information.

UNIQUEID

Query UNIQueid displays the unique serial numbers corresponding to the current file, view, and window.

EXTract /UNIQueid/ sets these variables:

uniqueid.0	3
uniqueid.1	Serial number of current file
uniqueid.2	Serial number of current view
uniqueid.3	Serial number of current window

KEDIT assigns a unique serial number to each new file that it creates, to each new view of a file that it creates, and to each new window that it creates. You can use this information to tell if a file, view, or window is the ''same'' one that you worked with at some earlier point, despite changes to the fileid involved, window position, etc.

VARBLANK

Query VARblank displays whether VARBLANK is ON or OFF.

EXTract /VARblank/ sets these variables:

varblank.0	1
varblank.1	ON|OFF

VERIFY

Query Verify displays the current VERIFY setting, as a list of one or more column pairs, each possibly preceded by an H if they are displayed in hexadecimal.

EXTract /Verify/ sets these variables:

verify.0	1
verify.1	VERIFY column pairs

VERSHIFT

Query VERShift displays the number of columns the VERIFY setting has been offset by the LEFT or RIGHT commands, or by autoscrolling.

EXTract /VERShift/ sets these variables:

vershift.0	1
vershift.1	VERSHIFT offset

VERSION

Query VERSION displays KEDIT's version string. The first word of the version string identifies the product (''KEDIT/Windows''). Next is the version number in the form of a major version number, a decimal point, and a two digit minor version number (for example, 1.50). Next is a two character revision level (for example, W1) and possible additional identifying text. Finally, there is the date of this version of KEDIT, in the form ''*Mmm dd yyyy*''.

EXTract /VERSION/ sets these variables:

version.0	4
version.1	''KEDIT/WINDOWS'' (this is returned for KEDIT for Windows; the DOS and OS/2 text mode versions of KEDIT return ''KEDIT'').
version.2	Version number
version.3	Revision level and possible additional identifying text
version.4	Date of this KEDIT version, in the form ''*MMM dd yyyy*''

If you want to test in a macro whether you are running under KEDIT for Windows, we recommend that you test for VERSION.1 equal to ''KEDIT/WINDOWS''.

WIDTH

Query Width displays the value of the WIDTH setting, which is set via the WIDTH initialization option. It is the length of the longest line that KEDIT can process in this editing session.

EXTract /Width/ sets these variables:

width.0	1
width.1	Value of the WIDTH setting

WINDIR

Query WINDIR displays the fully-qualified name of your Windows directory, and of your Windows system directory.

EXTract /WINDIR/ sets these variables:

windir.0	2
windir.1	Fully-qualified name of Windows directory
windir.2	Fully-qualified name of Windows system directory

WINDOWNAME

Query WINDOWNAME tells you the name of the current document window. This is usually simply the name of the file that you are editing, for example "C:\MyDir\MyFile.txt". But if you have multiple views of the same file in different windows (these are normally created with the Window/New Window menu item), the title bars of the windows involved include a colon (":") followed by a window number. And the output from Query WINDOWNAME includes this window number, giving for example "C:\MyDir\MyFile.txt:2".

EXTract /WINDOWNAME/ sets these variables:

windowname.0	1
windowname.1	Name of the current document window, in mixed case

WINMARGIN

Query WINMARgin displays whether the ability to drag the mouse in the window margin area to mark line blocks is turned ON or OFF, and displays the width in pixels of the window margin area.

EXTract /WINMARgin/ sets these variables:

winmargin.0	2
winmargin.1	ON|OFF
winmargin.2	Width of window margin area, in pixels

WORD

Query WORD displays whether “words” are strings of NONBLANK characters or strings of ALPHANUMeric characters,, and TRAILING or NOTRAILING depending on whether word selection does or does not include trailing blanks.

EXTract /WORD/ sets these variables:

word.0	2
word.1	NONBLANK|ALPHANUM
word.2	TRAILING|NOTRAILING

WORDWRAP

Query WORDWrap displays whether WORDWRAP is ON or OFF.

EXTract /WORDWrap/ sets these variables:

wordwrap.0	1
wordwrap.1	ON|OFF

WRAP

Query WRap displays whether WRAP is ON or OFF.

EXTract /WRap/ sets these variables:

wrap.0	1
wrap.1	ON|OFF

ZONE Query Zone displays the current left and right zone columns.

EXTract /Zone/ sets these variables:

zone.0	2
zone.1	Left zone column
zone.2	Right zone column

= Query = displays the contents of the equal buffer for the current file.

EXTract /=/ sets these variables:

equalsign.0	1
equalsign.1	Contents of equal buffer, in mixed case

Chapter 6. Macro Reference

This chapter presents a somewhat formal description of the KEXX macro language. A tutorial introduction to KEXX is given in User's Guide Chapter 10, "Using Macros". Other aspects of how to use KEDIT's macro facilities, such as the DEFINE command, .KEX and .KML files, and the KEXX debugging facility, are also discussed there.

If you want more information about KEXX, you may want to look at books about the REXX language, since KEXX includes a large enough subset of REXX for these to be helpful. The definitive reference for the REXX language is *The REXX Language: A Practical Approach to Programming* by Michael Cowlishaw (Second Edition, Prentice-Hall, 1990), which is recommended for all "power users" of KEDIT's macro facilities. Several IBM REXX manuals include somewhat similar reference information.

See also User's Guide Chapter 10, "Using Macros", User's Guide Chapter 11, "Sample Macros", DEFINE, IMMEDIATE, MACRO, MACROS, SET IMPMACRO, SET MACROPATH

6.1 Program Structure

Each line of a KEXX macro can be up to 250 characters long, and you can have up to 4000 lines in a KEXX macro.

Except within literal strings, case is not significant in the text of KEXX macros. The names of variables, keywords, and functions can be given in uppercase, in lowercase, or in mixed case.

KEXX *clauses* are the equivalent of statements in other languages. You can have one or more clauses on a line. If you have more than one clause on a line, you must use a semicolon (";") to separate each pair of clauses. The last or only clause on a line can optionally be followed by a semicolon. Clauses must fit on a single line; you cannot have a clause that extends over two or more lines.

Comments There are two ways to indicate comments within a KEXX macro:

When a line of a KEXX macro has an asterisk ("*") as its first nonblank character, the entire line is taken as a comment.

```
* this is a comment line
```

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
```

6.2 Tokens

The clauses in a KEXX macro are built up from *tokens*, which fall into these groups:

Symbols

Symbols are used as the names of KEXX variables, as the names of functions that KEXX macros can call, and as keywords for instructions like IF, DO, and EXIT. Symbols are made up of groups of alphabetic characters (“a—z“, ”A—Z”), numeric characters (“0—9”) and special characters (“!”, “?”, “_”, and “.”). Examples:

```
EXIT     WRAP.1     VarName    _x  .A1
```

Numbers

Numbers are a special type of symbol. They are composed of one or more digits and can optionally include a decimal point. Numbers are optionally preceded by a plus sign or a minus sign. You can use floating point notation by appending to a number “E” or “e”, an optional plus or minus sign, and one or more additional digits to represent the power of ten involved.

Examples:

```
24     -54     +081     1234.5678901  142.36e12    15E-5
```

This documentation will sometimes refer to *whole numbers*. These are numbers that can be expressed within the current number of NUMERIC DIGITS (which is normally nine digits) without a decimal point and without using exponential notation.

Literal strings

Literal strings are strings that are delimited by single or double quotes. For example,

```
'Error in data entry'   "Please enter your name"    ''
```

If you have a literal string delimited by single quotes that contains single quotes the embedded single quotes must be doubled, as in this example:

```
'Mary''s book'
```

And similarly, with any literal string delimited by double quotes that contains double quotes, the embedded double quotes also must be doubled:

```
"He said, ""Hello"" to Bob."
```

A special type of literal string is a *hexadecimal string*. When used in a KEXX macro, a hexadecimal string is equivalent to a string consisting of the characters whose character codes are given in the string. For instance, the hexadecimal string '0C'x specifies a one-character string containing the character whose character code in hexadecimal is 0C.

Hexadecimal strings are made up of pairs of hexadecimal digits (“0—9”, “a—f”, “A—F”), with each pair representing one character. Multiple pairs can be concatenated together or can be separated by blanks. The entire string must be delimited by single or double quotes and followed by the character “x” or “X”.

Binary strings are also available. They consist of strings of binary digits (“0“ or ”1”), with each group of four binary digits representing one hexadecimal digit. Binary strings are delimited by single or double quotes and are followed by the character “b” or “B”.

```
'0C'X    '01ecf8'x   "01 ec f8"x   '0010 0100'b    '11010111'B
```

Operators

Operators, such as “+” and “*”, are used in KEXX expressions and are discussed below in Section 6.5, “Operators and Expressions”.

Miscellaneous

KEXX also uses parentheses (for grouping within expressions and in function calls), commas (to separate arguments to functions), equal signs (in assignments), and colons (to indicate labels).

6.3 Symbols and Variables

As discussed above, symbols are made up of groups of alphabetic characters (“a—z”, “A—Z”), numeric characters (“0—9”) and special characters (“!”, “?”, “_”, and “.”).

Constant symbols

Symbols that begin with a period or with a number are known as *constant symbols*. The value of a constant symbol is always the name of the symbol in uppercase, and cannot be changed. Examples of constant symbols:

```
0   19   .ABC   54321   9g   4.2
```

Variable symbols

All other symbols (that is, symbols whose first character is non-numeric and is not a period) are known as *variable symbols*, and can be used to refer to variables. The value of all variables is initially equal to the name of the variable, in uppercase, but variables can be assigned new values when used on the left side of assignments, as discussed in the next section.

There are three types of variable symbols: *simple symbols*, *stem symbols*, and *compound symbols*, and they can be used to refer to the three types of variables: *simple variables*, *stem variables*, and *compound variables*.

Simple variables

Simple symbols are variable symbols that contain no periods. They are used to name simple variables, which correspond most closely to ordinary variables in other languages. Examples:

```
A    WRAP    CDe7
```

Stem variables

Stem symbols are variable symbols that contain a single period, which is the last character of the symbol. Examples:

```
A.   WRAP.   CDe7.
```

Stem symbols, which correspond most closely to the names of arrays in other languages, can be used in expressions just as simple symbols and compound symbols can,

but when used on the left side of an assignment, special handling applies, as discussed in the next section.

Compound variables

Compound symbols are formed by taking a stem symbol and concatenating a symbol that contains no periods. Examples:

```
A.12          A.I
```

In these examples, A. is known as the *stem component* of the compound symbol and 12 and I are known as the *tail component*. When you use a compound symbol, KEXX derives the name of the variable referred to (known as the *derived name* of the variable) by taking the stem component of the symbol as is, but replacing the symbol in the tail by its value. So the compound symbol *A.12* refers to the variable *A.12*, since the stem A. is taken as is, and 12 is a constant symbol with the value 12. Assuming that *I* has the value 9, *A.I* refers to the variable *A.9*, since A. is taken as is, and the variable *I* is replaced by its value, 9.

Compound variables correspond most closely to array elements in other languages. You can think of *A.12* as referring to the twelfth element of the array A, and you can think of *A.I*, where *I* is a variable whose value can vary, as letting you loop through the elements of the array A.

You can use multiple symbols in the tail of a compound symbol, separating them by periods, to make compound variables act like arrays with two or more dimensions. Examples:

```
A.11.12     B.I.J   C.I.J.1
```

KEXX derives the name of the compound variable referred to in these cases by substituting, for each symbol in the tail, the value of the symbol. So if *I* is 3 and *J* is 5, the derived name of *A.11.12* is *A.11.12*, the derived name of *B.I.J* is *B.3.5*, and the derived name of *C.I.J.1* is *C.3.5.1*.

Compound variables are actually more general than arrays in most other languages, because the values of the symbols in the tail need not be numeric. For example, if *NAME* has the value ‘‘Fred’’, *A.NAME* refers to *A.Fred*.

The values of symbols in the tail can contain any characters, including blanks and special characters. However, KEXX does not properly handle symbols in the tail whose values are null or contain the null character (character code 0).

6.4 Assignments

You can use *assignments* to change the values of variables. Assignments have the form

```
variable = expression
```

Examples of assignments:

```
X.1 = 17
A = SUBSTR(B, 1, 2)
CH2 = C || D || E
```

Stem assignment is a special case. When you assign a value to a stem, that value is also assigned to all possible compound variables whose name begins with that stem. (If you think of compound variables as corresponding to the array elements found in other languages, you can think of stem assignment as initializing an entire array to a given value.) For example,

```
A. = 'XYZ'
```

assigns "XYZ" to *A.0*, *A.1*, *A.2*, etc. KEXX uses special internal logic to handle stem assignments, and they use no more time or memory than assignments to other variables.

All variables always have values. If you use a variable that your macro has not yet assigned a value to, the value of the variable is the name of the variable in uppercase.

The values of all variables (simple, compound, and stem) are always character strings. Even numeric values are stored by KEXX as character strings.

6.5 Operators and Expressions

A KEXX *expression* can consist of a single variable, literal string, or function call. You can also use *operators* (along with parentheses for grouping) to combine these together into more complex expressions. This section discusses the different types of operators that you can use in KEXX: arithmetic operators, comparison operators, concatenation operators, and logical operators.

Arithmetic operators

Arithmetic operations performed by KEXX are:

Addition ("+")

Subtraction ("-")

Multiplication ("*")

Division ("/")

Integer division ("%")

Remainder ("//")

Exponentiation (“**”)

Unary plus (“+”)

Unary minus (“-”)

Some examples:

```
5 + 3            '8'
5 - 3            '2'
5 * 3            '15'
5 / 3            '1.66666667'
5 % 3            '1'
5 % -3           '-1'
5 // 3           '2'
5 // -3          '2'
5 ** 3           '125'
+(-5)            '-5'
-(-5)            '5'
```

Some rules that KEXX follows when processing arithmetic operations:

KEXX stores all values, including those that appear to be “numeric”, as character strings. As a result, all values can take part in “character” operations like concatenation and substring. When KEXX needs to do an arithmetic operation, it converts the operands from character to numeric form, giving you an error if the operands cannot be converted. Then KEXX does the operation and converts the result back to character form. For example, all of the following assignments have the effect of setting *X* to the string “103”.

```
X = '99' + '4'
X = 99 + 4
X = 99 + '4'
```

The NUMERIC DIGITS instruction controls how many significant digits are retained in arithmetic results. The default for NUMERIC DIGITS is 9, and results involving more digits than this are rounded to 9 digits. You can use NUMERIC DIGITS to specify up to 1000 significant digits.

```
numeric digits 9
say 123456 + .123456 /* result is 123456.123 */
say 123456**3        /* result is 18816403E+15 */
numeric digits 50
say 123456 + .123456 /* result is 123456.123456 */
say 123456**3        /* result is 1881640295202816 */
```

Superfluous leading zeros are removed from all results. Trailing zeros after a decimal point are removed from the results of division and exponentiation but are retained after other arithmetic operations. However, arithmetic results that are equal to zero always return '0', with no leading zeros, trailing zeros, or decimal point.

```
say 4.00 * 2      /* result is 8.00 */
say 4.00 / 2      /* result is 2 */
say 4.00 * 0      /* result is 0 */
```

Exponential notation is used for results that would otherwise require a large number of digits before or after the decimal point. Specifically, if the number of digits before the decimal point would be greater than the current NUMERIC DIGITS value (normally 9) or the number of digits after the decimal point would be more than twice that value (that is, more than 18 digits), KEXX expresses the result in exponential notation.

```
/* assuming default of NUMERIC DIGITS 9 */
say 1234567*100     /* result is 123456700 */
say 1234567*10000   /* result is 1.23456700E+10 */
say .1234567/1E2    /* result is 0.001234567 */
say .1234567/1E20   /* result is 1.234567E-21 */
```


Comparison operators

The comparison operators compare two values to see if the first is equal to the second, greater than the second, etc., yielding 1 if so and 0 if not.

KEXX has two sets of comparison operators. The *normal comparison* operators ignore leading and trailing blanks in the strings to be compared and, if the strings to be compared contain valid KEXX numbers, do numeric comparisons. *Strict comparison* compares two strings character by character, without any special handling of blanks and numbers. Here are some examples using the normal equality operator ("=") and the strict equality operator ("==", consisting of two equal signs):

```
'abc' = 'abc'                 '1'
'abc' == 'abc'                '1'

'ABC' = 'abc'                 '0'
'ABC' == 'abc'                '0'

'   abc  ' = 'abc'            '1'
'   abc  ' == 'abc'           '0'

'abc     ' = '   abc'         '1'
'abc     ' == '   abc'        '0'

'99' = '99'                   '1'
'99' == '99'                  '1'

'099' = '99'                  '1'
'099' == '99'                 '0'
```

Normal comparison operators

The normal comparison operators are:

Equal ("=")

Greater than (">")

Less than (“<”)

Not equal (“\=”, “<>”, “><”)

Greater than or equal to (“>=“, ”\<”)

Less than or equal to (“<=“, ”\>”)

Strict comparison operators

The strict comparison operators are:

Strictly equal (“==”)

Strictly greater than (“>>”)

Strictly less than (“<<”)

Not strictly equal (“\==”)

Strictly greater than or equal to (“>>=”, “”\<<”)

Strictly less than or equal to (“<<=”, “\>>”)

Some examples:

```
'abc' < 'cde'                  '1'
'abc' << 'cde'                 '1'

'012' >= '9'                   '1'
'012' >>= '9'                  '0'

' abc ' \< 'abc'               '1'
' abc ' \<< 'abc'              '0'
```

Concatenation operators

The concatenation operators join two strings together to form a longer string. The concatenation operators are:

Blank
When KEXX encounters, within an expression, two values separated by one or more blanks, KEXX concatenates the two values, placing a single blank between them.

Abuttal
When KEXX encounters, within an expression, two values immediately adjacent to each other, KEXX concatenates the two values, with no blank between them.

||
Two values that you want to concatenate cannot always be unambiguously placed immediately adjacent to each other. For example, if the variable name *X* is placed immediately adjacent to the variable name *Y*, KEXX does not concatenate the values of *X* and *Y*, but instead takes this as a reference to the variable *XY*. For these

cases, KEXX provides the concatenation operator ||, which also concatenates two values, placing no blanks between them. (The concatenation operator consists of two occurrences of character code 124, which appears on most U.S. keyboards as a split vertical bar, located on the backslash key.)

In the following examples, assume that *X* has the value "Hello" and *Y* has the value "there.", while *XY* has the value "Goodbye":

```
X Y              'Hello there.'
X  Y             'Hello there.'
X'='Y            'Hello=there.'
XY               'Goodbye'
X||Y             'Hellothere.'
X || Y           'Hellothere.'
```


Logical operators

Only the values 1 and 0, representing logical true and false values, are valid as operands of the logical operators. The results of the logical operations are themselves either 1 or 0.

Negation ("\")
Negation, a unary operator, yields 0 if its operand has the value 1, and yields 1 if its operand has the value 0.

And ("&")
And takes two operands, yielding 1 if both operands have the value 1, and yielding 0 if either operand has the value 0.

Inclusive or ("|")
Inclusive or takes two operands, yielding 1 if either operand has the value 1, and yielding 0 if both operands have the value 0. (The inclusive or operator has character code 124, and appears on most U.S. keyboards as a split vertical bar, located on the backslash key.)

Exclusive or ("&&")
Exclusive or takes two operands, yielding 1 if exactly one of the operands has the value 1, and yielding 0 if both operands have the value 0 or both operands have the value 1.

Some examples:

```
\'1'                          '0'
'1' & '0'                     '0'
'1' | '0'                     '1'
'1' && '0'                    '1'
'1' && '1'                    '0'
(4 < 6) & (7 > 0)             '1'
\('012' == '12')              '1'
```

Operator precedence

Operators of higher precedence are evaluated before operators of lower precedence. When operators of equal precedence are encountered, KEXX evaluates expressions

left-to-right. You can use parentheses to group subexpressions together if you need to change the normal order of evaluation. Operator precedence, from highest to lowest:

Unary plus, unary minus, negation

Exponentiation

Multiplication, division, integer division, remainder

Addition, subtraction

Concatenation

Comparison

And

Or, exclusive or

6.6 Commands

Any clause that is not recognized as an assignment and is not one of the keyword instructions discussed in the next section is taken as an expression that is to be evaluated and passed to KEDIT as a command. It is only by issuing commands to KEDIT that a macro can cause KEDIT to take any action (for example, move the cursor, locate a string, or delete a line).

When the content of a KEDIT command string that you want to issue from a macro is fixed in advance, it should appear in the macro as a literal string enclosed in quotes. This prevents KEXX from interpreting parts of the command as KEXX keywords, variables, or operators. Some examples:

```
'save \total.dat'
':17'
'i 27'
```

Note the problems that could occur if you left out the quotes in these examples. The first two examples would be invalid KEXX expressions. The third example would cause unexpected results in a macro that happened to use *I* as a variable.

A command string passed to KEDIT from a KEXX macro need not be determined in advance. The result of evaluating any KEXX expression can be passed to KEDIT as a command. For example:

```
'i' date() time()
'fileid' fn'.'fext
':'n
```

After a command issued from a macro completes, a return code is placed in the variable *RC* indicating success or failure of the command. Chapter 9, “Error Messages and Return Codes”, has a discussion of the return codes set by KEDIT commands.

6.7 Keyword Instructions

If a clause is not an assignment, KEXX checks to see if it begins with a KEXX keyword and is a KEXX *keyword instruction*. (A clause that is neither an assignment nor a keyword instruction is taken as an expression whose value is passed to KEDIT as a command.)

Below are the keyword instructions that you can use.

ARG [template]

The ARG instruction takes the arguments of a KEXX macro, internal routine, or external routine and parses their values, in uppercase, according to the specified template. ARG is a shorthand equivalent of PARSE UPPER ARG. See Section 6.9, "The Parse Instruction", for a discussion of the PARSE instruction.

When executed from a KEXX macro while no internal routine is active, ARG parses the argument string passed to the macro when it was invoked. When executed from an internal routine within a KEXX macro, or from an external routine, ARG parses the arguments to that routine.

```
* ARG example
 call test 2,4,6
 exit
test:
 * this internal routine expects three arguments
 arg a,b,c
 say a*b*c
 return
```

This example would output the value 48.

CALL name [expr [,expr ...]]
CALL ON condition [NAME trapname]
CALL OFF condition

The first form of the CALL instruction invokes a KEXX function as a subroutine. The values of any expressions that you specify are passed as arguments to the function.

See Section 6.8, "Functions", for more about KEXX functions.

If KEXX finds a label within the currently executing macro equal to the name of the function you want to call, KEXX sets the variable *SIGL* equal to the line number of the CALL instruction and then passes control to the internal routine beginning at that label. (This step is bypassed if the *name* is specified in quotes.)

If *name* is not an internal routine, KEXX looks for a built-in function by that name, then (although in practice this would rarely be useful) for a KEDIT Boolean function or Implied EXTRACT function, and finally for an external routine, giving an error if the function cannot be found.

The variable *RESULT* is set equal to the value returned by the function; if no value is returned, the variable *RESULT* is dropped.

```
* CALL Example
 * call internal routine to input 5 lines
 call multi 'This is one',5
 * call internal routine to input 10 lines
 call multi 'This is another',10
 exit
multi:
 parse arg s,n
 do i = 1 to n
    'input' s
 end
 return
```

Two other forms of the CALL instruction are

```
CALL ON condition [NAME trapname]
```

and

```
CALL OFF condition
```

They are discussed in Section 6.10, "Conditions".

DO [repetitor] [conditional]
clause(s)
END

The DO instruction executes a set of clauses as a unit. There are several ways to specify how many times the clauses are to execute:

DO

A simple DO group is executed once. This construct allows you to group together several clauses that are to be executed as a unit, normally within IF—THEN—ELSE constructs. For example:

```
if x > 100 then do
   'down 20'
   'delete'
   end
```

DO *expr*

You can give an expression specifying the number of times the loop is to be executed. For example:

```
* move the cursor three characters to right
do 3; 'cursor right'; end
```

DO *var* **=** *initexpr* **[TO** *toexpr***] [BY** *byexpr***] [FOR** *forexpr***]**

This construct is closest to the "DO loop" in other languages. A control variable (*var*), which must be a simple variable and not a stem or compound variable, is given an initial value (*initexpr*). While the value of *var* is less than or equal to the

value of *toexpr*, the loop is executed and then the *var* is incremented by the value of *byexpr* (or by 1, if this is not specified). (If *byexpr* is negative, the loop repeats while *var* is greater than or equal to *toexpr*.) An optional *forexpr* expression specifies the maximum number of times that the loop will be executed. For example:

```
* uppercase every second line of the file
do i = 2 to size.1() by 2
   'locate :'i
   'uppercase'
   end
```

On a DO loop involving a control variable, the terminating END can optionally be followed by the name of the control variable (END *var*). This forces an error if the variable name does not correspond to the control variable for the loop being ended, helping you track down mismatched DO/END pairs.

DO FOREVER

You can specify that a loop repeat indefinitely, until it is terminated by a LEAVE, EXIT, or RETURN instruction within the loop.

```
* loop until user enters a number
do forever
   say 'Enter a number'
   'readv cmdline'
   if datatype(readv.1) = 'NUM' then leave
   end
```

DO WHILE *expr*

You can use DO WHILE to specify that a loop repeat for as long as some specified expression is true (that is, evaluates to 1). The expression will be evaluated at the start of each iteration of the loop, and the loop will terminate if the expression is false.

```
* loop while file has < 1000 lines
do while size.1() < 1000
   'get data.fil'
   end
```

DO UNTIL *expr*

You can use DO UNTIL to specify that a loop repeat until an expression becomes true. The expression will be evaluated at the end of each iteration of the loop, and the loop will terminate if the expression is true.

```
* loop until user enters a number
do until datatype(readv.1) = 'NUM'
   say 'Enter a number'
   'readv cmdline'
   end
```

You can also combine repetitive DO loops with conditional loops:

```
* read 10 files, stopping after any error
do i = 1 to 10 until rc \= 0
   'get file.'i
   end
```

DROP var1 [var2 ...]

The DROP instruction resets the specified variables to their default values (that is, the name of the variables in uppercase), freeing up any space used to hold the variables and their current values. If you DROP a stem, the values of all compound variables whose names begin with the specified stem are also dropped.

The list of variables can also include variable names in parentheses. When a variable name in parentheses is encountered, the variable itself is not dropped, but its value is taken as a list of variables that should themselves be dropped. For example:

```
list = 'a b c.19'
drop v1 v2 (list) xyz
```

would drop the variables V1, V2, A, B, C.19, and XYZ.

EXIT [expr]

The EXIT instruction terminates execution of a KEXX macro. If *expr* is given, it specifies a numeric return code in the range -32767 to 32767 to be passed back from the macro. (A return code of 0 is passed back if *expr* is not specified or has an invalid value.)

```
'locate /abc/'
if rc \= 0 then
   exit rc
else
   'input def'
```

IF expr THEN clause [ELSE clause]

The IF instruction looks at the value of the specified expression, which must evaluate to 0 or 1. If the expression is true (evaluates to 1), KEXX executes the THEN clause. If the expression is false (evaluates to 0), KEXX evaluates the optional ELSE clause. The THEN and ELSE clauses can be assignments, instructions, or commands; multiple clauses can be executed by embedding them in a DO—END pair. You can begin a new line before or after THEN and before or after ELSE. If you want to have both a THEN clause and an ELSE clause on the same line, though, a semicolon (“;”) must follow the THEN clause.

Some examples:

```
if size.1() = 0 then 'qquit'; else say 'File is not empty'
```

```
if size.1() = 0 then
   'qquit'
else
   say 'File is not empty'
```

```
if x < 100 then do
   'down 20'
   'delete'
   end
else
   'top'
```

INTERPRET expr

The INTERPRET instruction takes the value of the expression *expr* and executes it, as if it were one of the lines of your macro. This lets you construct and execute KEXX instructions ‘‘on the fly’’ while your macro is running.

The text that you execute can consist of a single clause or multiple clauses separated by semicolons. It can contain DO instructions if it contains the complete DO construct, including the terminating END instruction.

```
* INTERPRET Example
 a = 'say 5'
 b = '+ 6'
 * the next line would output 11
 interpret a b
 v = 'test'
 * the next line would set the variable test to 17
 interpret v '= 17'
```

Note that the KEDIT IMMEDIATE command works much like the INTERPRET instruction. The difference is that INTERPRET executes instructions as part of the current macro, while IMMEDIATE executes instructions as a separate one-line macro. When your macro uses INTERPRET, the instructions that are executed can access and change variables in your macro and, if they contain syntax errors, can terminate execution of your macro. If your macro instead uses the IMMEDIATE command, the instructions that are executed cannot access or change variables in your macro, and if they contain syntax errors, a bad return code will be passed back to your macro, but the macro will continue to execute.

ITERATE [var]

The ITERATE instruction can be used only within a repetitive DO loop (as opposed to a simple DO group). If *var* is not specified, ITERATE ends execution of the current iteration of the innermost DO loop. If *var* is specified, ITERATE ends execution of the current iteration of the innermost DO loop using *var* as its control variable. Control returns to the top of the loop for execution of the next iteration, if any.

```
* ITERATE Example
 do i = 1 to 10
   if i = 3 then iterate
   * next line skipped when i is 3
   say i 'is not equal to 3'
  end
```

LEAVE [var]

The LEAVE instruction can be used only within a repetitive DO loop (as opposed to a simple DO group). If *var* is not specified, LEAVE ends execution of the innermost DO loop. If *var* is specified, LEAVE ends execution of the innermost DO loop using *var* as its control variable. Control passes to the clause after the END instruction at the end of the loop.

```
* LEAVE Example
 do i = 1 to 10
   'down' i
   * loop terminates on nonzero return code
   if rc \= 0 then leave
   'change /abc/def/'
  end
 'add 5'
```

NOP

```
NOP ("no operation") is a dummy instruction that does nothing.
```

NUMERIC DIGITS [expression]
NUMERIC FUZZ [expression]

With NUMERIC DIGITS *expression*, the value of *expression*, which must be a positive whole number, determines the precision used in arithmetic operations. By default, 9 significant digits will be used for results of arithmetic operations, but you can specify that KEXX should use up to 1000 significant digits. NUMERIC DIGITS values of less than 9 can be specified but are generally not useful. If *expression* is omitted, the default of NUMERIC DIGITS 9 is put into effect.

Less often used than NUMERIC DIGITS is NUMERIC FUZZ. With NUMERIC FUZZ *expression*, the value of *expression*, which must be a non-negative whole number less than the current NUMERIC DIGITS setting, determines how many low-order digits are ignored during numeric comparisons. By default, NUMERIC FUZZ is 0 and no digits are ignored. But if, for example, NUMERIC DIGITS 9 and NUMERIC FUZZ 1 are in effect and two nine-digit numbers are compared, KEXX actually rounds these numbers to 8 digits and compares the two eight-digit numbers. This allows for comparisons that yield useful results when comparing numbers that are only approximately equal due to rounding that occurs during arithmetic operations. If NUMERIC FUZZ is used and no *expression* is specified, the default of NUMERIC FUZZ 0 is put into effect.

The NUMERIC settings are preserved and restored across calls to internal routines.

OPTIONS expression

OPTIONS is a REXX instruction that allows you to pass special options to the language processor. KEXX does not use any such special options, so the OPTIONS instruction is allowed within KEXX programs, but is ignored.

PARSE [UPPER] origin [template]

The PARSE instruction takes character strings and assigns portions of their values to a set of KEXX variables according to a template that you provide.

UPPER is optional and specifies that the strings are to be converted to uppercase before being parsed according to the template.

Origin specifies where the PARSE instruction is to obtain the data to be parsed.

Template is a set of pattern specifications that controls the parsing process, intermixed with lists of variables to which the parse data is to be assigned.

The PARSE instruction is discussed in detail in Section 6.9, “The Parse Instruction”.

PROCEDURE [EXPOSE var1 var2 ...]

The PROCEDURE instruction is valid only at the start of a KEXX internal routine; it must be the first instruction encountered after the label that begins the internal routine. The PROCEDURE instruction tells KEXX to make all variables used in the routine local to the routine. Variables in the routine do not inherit values from the calling routine and variables set in the routine do not affect the values of variables of the same name in the calling routine. In contrast, when PROCEDURE is not the first instruction of an internal routine, all variables used in the routine are shared with the calling routine.

```
* PROCEDURE Example
 do i = 1 to 10
   say sum(i)
   end
 exit
sum: procedure
 * i in this procedure doesn't affect i in main routine
 total = 0
 do i = 1 to arg(1)
    total = total + i
    end
 return total
```

PROCEDURE can optionally be followed by the keyword EXPOSE and a list of variables that should not be local to the procedure, but should instead be shared with the caller of the procedure. That is, references to variables in the list are treated as references to variables of the same name in the calling procedure. For example:

```
 x = 1
 y = 2
 call test
 say x y
 exit
test: procedure expose x
 x = 55
 y = 66
 return
```

would display "55 2", because changes made within TEST to the value of X are exposed to the caller, while changes made to Y are not.

The list of variables to be exposed can include simple variables, compound variables, and stem variables. Items in the list can also be variable names in parentheses. In this case, the variable in parentheses is first exposed, and then its value is taken to be a list of additional variables to be exposed. For example:

```
 height = 12
 width  = 13
 depth  = 14
 globals = 'height width'
 call test
 exit
test: procedure expose (globals)
 say height width depth
 return
```

would display "12 13 DEPTH", since *height* and *width* are exposed, but *depth* is not.

PULL [template]

The PULL instruction waits for a line of input to be entered on the KEDIT command line, uppercases it, and then parses it according to the specified template. PULL is a shorthand equivalent of PARSE UPPER PULL. See Section 6.9, "The Parse Instruction", for a discussion of the PARSE instruction.

RETURN [expr]

The RETURN instruction ends execution of an internal or external KEXX routine and returns control to the caller. If the routine was invoked as a function, *expr* is required and its value is used as the result of the function. If the routine was invoked through the CALL instruction as a subroutine, *expr* is optional and, if it is specified, the special variable *RESULT* is set equal to its value. If no internal routine is active and RETURN is executed from the main body of a macro, RETURN is equivalent to EXIT and serves to terminate execution of the macro.

SAY [expr]

The SAY instruction displays the value of the specified expression on the KEDIT message line. If no expression is given, a blank line is displayed. When the KEXX

debugger is active, SAY instructions issued from the debugging command line send output to the debugging window rather than the KEDIT message line.

```
say 'There are' size.1() 'lines in this file.'
```

SELECT
WHEN expr THEN clause
...
[OTHERWISE instructionlist]
END

The SELECT instruction lets you execute one sequence of instructions out of a set of possible alternatives. SELECT offers a cleaner way of specifying what would otherwise be a sequence of IF—THEN—ELSE instructions.

An example of a SELECT instruction:

```
select
  when option = 'A' then call optiona
  when option = 'B' then call optionb
  when option = 'C' then do
     say 'Option C selected'
     call optionc
     end
  otherwise
     say 'Unknown option specified'
     return
end
```

SELECT constructs begin with a SELECT instruction and end with an END instruction. In between, you use one of more instructions of the form

`WHEN expression THEN instruction`

Each WHEN expression is tested in sequence. As soon as one is found to be true (that is, has the value 1), the corresponding THEN instruction is executed. To execute more than one instruction when an expression is found to be true, group them together within a DO—END pair. After the THEN instruction corresponding to the true expression has been executed, no additional WHEN expressions are evaluated, and execution continues with the instruction following the END that terminates the SELECT construct.

`OTHERWISE instructionlist`

The set of WHEN—THEN instructions can be followed by the keyword OTHERWISE and a sequence of zero or more instructions to be executed if none of the WHEN expressions are true. If you know that at least one of the WHEN expressions will always be true you can omit the OTHERWISE construct, but KEXX generates an error if none of the WHEN expressions are true and OTHERWISE is not present.

SIGNAL label
SIGNAL ON condition [NAME trapname]
SIGNAL OFF condition

Use the SIGNAL instruction to immediately transfer control to some other location within a KEXX program. To do this, use

```
SIGNAL label
```

where *label* is the label to which you want to transfer control. Execution of any active DO, IF, SELECT, and INTERPRET instructions in the current routine is terminated, and execution continues at the specified label.

Also available are

```
SIGNAL ON condition [NAME trapname]
```

and

```
SIGNAL OFF condition
```

They are discussed in Section 6.10, “Conditions”.

TRACE setting

The TRACE instruction controls the type of trace output displayed while you are debugging a KEXX macro and whether tracing is done interactively or non-interactively. The TRACE instruction has no effect when the KEXX debugging window, controlled by the SET DEBUGGING command, is not active.

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. When interactive debugging is active, TRACE instructions issued from your macro are ignored so that interactive debugging will not be unexpectedly interrupted.

The tracing level is preserved and restored across calls to internal routines, so changes to the tracing level in a subroutine do not affect the tracing level in the calling routine.

Here are the tracing levels that you can use. Note that only the first character of the TRACE setting is significant:

`TRACE Off`
: No trace output is produced. (TRACE Off also turns off interactive tracing if it is in effect.)

`TRACE Error`
: Any command passed to KEDIT that yields a nonzero return code is traced, along with its 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.

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 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.

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.

6.8 Functions

To call a function from within a KEXX expression, use the name of the function, followed by a left parenthesis, followed by any arguments to the function (the arguments are themselves KEXX expressions, and are separated from each other by commas), followed by a right parenthesis. There can be no intervening blanks between the name of the function and the left parenthesis that follows it. All functions invoked

in this manner return a value, which KEXX uses to evaluate the expression involving the function. Some examples:

```
lastchr = substr(s, length(s))
if size.1() = 0 then say 'File is empty'
if after() then 'sos endchar'
```

Instead of using the value of a function within an expression, you can use the CALL instruction to invoke a function as a subroutine. This is usually done when the function is called for its side effects, and the function either does not return a value or the returned value is of secondary importance. If a function invoked via the CALL instruction does return a value, the value is assigned to the special variable *RESULT*. Note that while parentheses are required around the parameter list when invoking a function from within an expression, parentheses are not allowed around the parameter list passed with the CALL instruction.

```
call time r
call process 'Some data', 'More data'
```

When searching for a function, KEDIT looks first for an internal routine with the specified name, then for a built-in function, an implied EXTRACT function, a Boolean function, and finally an external routine. (The search for an internal routine is bypassed if the name of the function is in quotes.)

Internal routines

You can write your own functions, place them inside a KEXX macro as internal routines, and call them from elsewhere in the same KEXX macro.

An internal routine must begin with a label—the name of the routine followed by a colon. Control is returned from the internal routine to the calling routine with a RETURN instruction. The internal routine can return a value to its caller (and is required to do so if it was invoked as a function) by specifying a result expression with the RETURN instruction.

Here is an example of an internal routine:

```
 say sum3(2,4,6)
 say sum3(6,8,10)
 exit
sum3:
 * add up three numbers and return the sum
 arg n1, n2, n3
 return n1+n2+n3
```

This sample macro would output 12 and 24 on the KEDIT message line.

An internal routine can normally access all of the caller's variables, and all variables set in the internal routine are normally accessible to the caller after the internal routine returns. With PROCEDURE as the first instruction of an internal routine, variables used in the internal routine will instead be local to the routine and the routine will not have access to the variables of its caller. You can use PROCEDURE EXPOSE to share specific variables between the caller and the internal routine, leaving all other variables as local to the internal routine.

The current NUMERIC and TRACE settings, and the value of the elapsed-time clock used with the TIME() function, are saved whenever an internal routine is invoked and are restored on return from that routine, so that changes made to these settings by an internal routine do not affect the environment of its caller.

Built-in functions

Some functions are "built into" KEXX in the sense that they perform generic operations on their arguments, independent of the status of your KEDIT session. KEXX's built-in functions are documented in the next section.

Implied EXTRACT functions

KEXX macros can use KEDIT's EXTRACT command to obtain information about the status of KEDIT, with the results placed into KEXX variables. Implied EXTRACT functions, documented in Section 5.2, "EXTRACT and Implied EXTRACTs", generally provide a more direct and efficient way to access this information within a KEXX expression.

Boolean functions

KEXX macros can obtain additional information about the status of KEDIT by using KEDIT's Boolean functions, which return 1 or 0 depending on whether certain conditions within KEDIT are true or false. Boolean functions are documented in Section 6.8.3, "Boolean Functions".

External routines

A KEXX macro can call another KEXX macro as an external subroutine. The external subroutine can use PARSE ARG to access the arguments passed to it, and can use the RETURN instruction to pass a result back to the caller.

When searching for an external routine, the order of searching is the same as it is with the MACRO command: in-memory macros loaded via the DEFINE command are searched for first, and then macros in disk files (with a default extension of .KEX) are searched for, with the search order controlled by the current SET MACROPATH setting.

External routine names with drive or path components, or including special characters, must be specified as quoted names.

For example,

```
CALL 'F:\MACROS\SAMPLE' X, Y, Z
```

or

```
SAY '&&&'(17)
```

6.8.1 Built-in Functions

KEXX supports all of the built-in functions described in Cowlishaw's *The REXX Language: A Practical Approach to Programming* (Second Edition, Prentice-Hall, 1990), with the exception of the STREAM() function. It also supports the following functions, which are not included in Cowlishaw's book: BEEP(), DATECONV(), DELIMIT(), DOSENV(), DOSDIR(), ANSIUPPER(), ANSILOWER(), ANSIDATATYPE(), ANSITOOEM(), OEMTOANSI(), UPPER(), LOWER(), COUNTSTR(), CHANGESTR(). LONGNAME(), and SHORTNAME().

Some general rules for the arguments to these built-in functions:

Where a function accepts an *option* argument, only the first character of *option* is significant and it may be specified in uppercase or in lowercase.

Where a function accepts a *pad* argument, the argument must be a single character.

Where an optional argument is omitted, you should specify nothing, not even a null string, in its place. For example, to omit the third argument to the SUBSTR() function, you could use

```
t = substr(s,3,,'.')
```

The two adjacent commas in the above example are required to indicate that it is the third argument that is omitted and that the fourth is supplied. However, if the final argument or arguments are omitted, trailing commas are not necessary and are usually not supplied. So here is a valid way of omitting the third and fourth arguments for SUBSTR():

```
t = substr(s,3,,)
```

But here is the form in which this function call would normally appear:

```
t = substr(s,3)
```

ABBREV(string1,string2[,length])

Returns 1 if *string2* is equal to the first characters in *string1*; otherwise, 0 is returned. If *length* is specified, *string2* must be at least *length* characters long or 0 will be returned.

```
abbrev('clocate','cl')              1
abbrev('clocate','clo',2)           1
abbrev('clocate','c',2)             0
```

ABS(number)

Returns the absolute value of *number*, formatted according to the current NUMERIC settings.

```
abs(-3.2)                       3.2
abs(0)                          0
abs('   +32760')               32760
```

ANSIDATATYPE(string[,option])

ANSIDATATYPE() works in exactly the same way as the more-commonly-used DATATYPE() function, except that the “A”,“L”, “M”, and “U” options assume that *string* uses the ANSI character set and they treat all alphabetic characters, including accented letters with character codes above 127, as alphabetic. DATATYPE(), on the other hand, treats only the 26 characters of the English alphabet as alphabetic.

If *option* is not specified, ANSIDATATYPE() returns ‘‘NUM’’ if *string* is a valid KEXX number or ‘‘CHAR’’ if it is not. If *option* is specified, it is handled as follows:

A (‘‘Alphanumeric’’) Returns 1 if *string* consists entirely of ANSI alphabetic and/or numeric (‘‘0—9’’) characters, else returns 0;

B (‘‘Binary’’) Returns 1 if *string* consists entirely of the characters ‘‘0’’ and ‘‘1’’, else returns 0;

L (‘‘Lowercase’’) Returns 1 if *string* consists entirely of lowercase ANSI alphabetic characters , else returns 0;

M (‘‘Mixed case’’) Returns 1 if *string* consists entirely of ANSI alphabetic characters, else returns 0;

N (‘‘Numeric’’) Returns 1 if *string* is a valid KEXX number, else returns 0;

S (‘‘Symbol’’) Returns 1 if *string* consists entirely of characters that are valid in KEXX symbols, else returns 0;

U (‘‘Uppercase’’) Returns 1 if *string* consists entirely of uppercase ANSI alphabetic characters, else returns 0;

W (‘‘Whole number’’) Returns 1 if *string* is a whole number. That is, 1 is returned if *string* is a valid KEXX number that can be expressed as an integer, with no decimal point or exponential notation, under the current NUMERIC DIGITS setting. Otherwise returns 0;

X (‘‘heXadecimal’’) Returns 1 if *string* is the null string or consists entirely of valid hexadecimal digits (‘‘a—f’’, ‘‘A—F’’, ‘‘0—9’’), with optional blanks between pairs of hexadecimal digits. Otherwise returns 0.

ANSILOWER(string)

Returns the value of *string* with any uppercase letters translated to lowercase. *String* is assumed to be in the ANSI character set and characters ‘‘A—Z’’, as well as accented uppercase characters with ANSI character codes above 127, are affected.

```
ansilower('AbCd123')          'abcd123'
```

ANSITOOEM(string)

Returns the result of converting *string*, which is assumed to be in the ANSI character set, to the OEM character set.

ANSIUPPER(string)

Returns the value of *string* with any lowercase letters translated to uppercase. *String* is assumed to be in the ANSI character set and characters ‘‘a—z’’, as well as accented lowercase characters with ANSI character codes above 128, are affected.

```
ansiupper('AbCd123')          'ABCD123'
```

ARG([n[,option]])

ARG() returns information about the arguments passed to a macro or, when used within an internal routine, about arguments to the internal routine.

ARG() with no parameters returns the number of arguments passed to the macro or internal routine. Macros can have either 0 or 1 arguments. Internal routines can have 0 to 10 arguments.

ARG(*n*) returns the value of the *n*th argument, or returns a null string if the *n*th argument was omitted.

ARG(*n*,*option*) accepts the option ‘‘E’’ (‘‘Exists’’), returning 1 if the *n*th argument to the macro or internal routine is present and 0 if it was omitted, and accepts the option ‘‘O’’ (‘‘Omitted’’), returning 1 if the *n*th argument was omitted and 0 if it is present.

In the first group of examples, assume that the macro TEST is invoked via the following KEDIT command:

```
macro test I think, therefore, I am.

arg()                   1
arg(1)                  'I think, therefore, I am.'
arg(2)                  ''
arg(1,'e')              1
arg(1,'o')              0
```

In the next group of examples, assume that ARG() is used within an internal routine INTEST invoked by

```
call intest 'I think', 'therefore', 'I am.'

arg()                   3
arg(1)                  'I think'
arg(2)                  'therefore'
arg(3)                  'I am.'
arg(1,'e')              1
arg(1,'o')              0
```

BEEP()

Causes the PC's speaker to beep and returns a null string as its value.

```
call beep
```

BITAND(string1[,string2[,pad]])

Returns the result of the logical ANDing of the bits in *string1* with those in *string2*, which defaults to the null string. The shorter string is padded with the *pad* character, if specified, prior to the ANDing. If *pad* is not specified, the remaining characters of the result are the remaining characters in the longer string.

```
bitand('31ff'x, 'ff30'x)            '10' /* (`3130'x) */
bitand('ABC','abc')                 'ABC'
bitand('0011'x, '0101'x)            '0001'x
bitand('ABC','abcef')               'ABCef'
bitand('ABCEF','abc')               'ABCEF'
```

BITOR(string1[,string2[,pad]])

Returns the result of the logical ORing of the bits in *string1* with those in *string2*, which defaults to the null string. The shorter string is padded with the *pad* character, if specified, prior to the ORing. If *pad* is not specified, the remaining characters of the result are the remaining characters in the longer string.

```
bitor(135,222)                      337
bitor('313335'x, '323232'x)         '333337'x
bitor('ABC',,' ')                   'abc'
bitor('0011'x, '0101'x)             '0111'x
```

BITXOR(string1[,string2[,pad]])

Returns the result of the logical eXclusive ORing of the bits in *string1* with those in *string2*, which defaults to the null string. The shorter string is padded with the *pad* character, if specified, prior to the operation. If *pad* is not specified, the remaining characters of the result are the remaining characters in the longer string.

```
bitxor('abc', '   ')                'ABC'
bitxor(3579,,'01'x)                 2468
bitxor('0011'x, '0101'x)            '0110'x
```

B2X(binary-string)

Converts a string in binary notation (that is, consisting of zeroes and ones) into the equivalent hexadecimal string. Blanks may be included in *binary-string*, at four character boundaries.

```
b2x('1101')                    'D'
b2x('00001101')                '0D'
b2x('1011010011111')           '169F'
```

CENTER(string,length[,pad])
CENTRE(string,length[,pad])

Returns a string of length *length* with *string* centered within it, padded if necessary with the *pad* character, which defaults to a blank.

```
center('title',12)                    '   title    '
center('title',12,'*')                '***title****'
center('title   ',12)                 '  title     '
center('title   ',12,'*')             '**title  **'
```

CHANGESTR(needle, haystack, newneedle)

Returns the result of changing all occurrences of the string *needle* in the string *haystack* into occurrences of the string *newneedle*.

```
changestr('a', 'abABab', 'x')              'xbABxb'
changestr('aa', 'aaabbaa', 'x')            'xabbx'
changestr('3', '123123123', '')            '121212'
```

CHARIN(fileid[,[start][,length]])

Reads data from a file. The read position for the file is first moved to the specified *start* position in the file, and up to *length* characters are read in and returned as CHARIN's result. Fewer than *length* characters may be read if the file does not exist, if end-of-file is reached, or if an I/O error occurs. If *length* is zero, then the read position is moved to the specified *start* position, but no data is read.

Start defaults to the current read position for the file and *length* defaults to 1. If *fileid* is not already open, it is first opened and the read position is set to 1 (the beginning of the file). On completion of the read operation, the read position for the file is then moved just beyond the last character read.

The maximum allowable value for *start* and for *length* is 999999999.

See page 374 for some general notes on KEXX's I/O functions.

```
charin('sample.fil',,14)              'Data from file'
charin('sample.fil',100,19)           '100 bytes into file'
```

CHAROUT(fileid[,[string][,start]])

Writes data to a file. The write position for the file is first moved to the specified *start* position in the file, and the contents of *string* are written to the file. CHAROUT returns

as its value the number of characters from *string* that were not written out; this value will be 0 unless an I/O error or disk full condition is encountered.

Start defaults to the current write position for the file. If *fileid* is not already open, it is first opened and the write position is initially set to the end of the file, so that the file will be appended to. After the write operation completes, the write position for the file is moved just beyond the last character written. If *start* is specified but *string* is omitted, the write position for the file is updated but no data is written.

The maximum allowable value for *start* is 999999999.

If both *string* and *start* are omitted, the file is closed.

See page 374 for some general notes on KEXX's I/O functions.

```
charout('name.ext','Some data')                    0 /* if no error */
charout('name.ext','Data for byte 100',100)        0 /* if no error */
```


CHARS(fileid)

Returns the number of characters remaining to be read in the specified file, starting from the current read position.

The maximum value returned is 999999999, regardless of the actual size of the file involved.

```
chars('name.ext')              1234 /* perhaps */
chars('nonexist.ent')          0    /* perhaps */
```

COMPARE(string1,string2[,pad])

Returns 0 if *string1* equals *string2*. Otherwise, the position of the first mismatched character is returned. The shorter string is padded with *pad*, which defaults to a blank.

```
compare('this is it', 'this is it')                0
compare('this is it', 'this is not it')            9
```

CONDITION([option])

Returns information about the currently trapped condition. For information about condition handling, see Section 6.10, "Conditions". If no condition is currently being handled, the CONDITION() function returns the null string.

Possible values for *option*, which defaults to "I", are:

C ("Condition name") Returns the name of the current condition, which can be ERROR, FAILURE, HALT, SYNTAX, or NOVALUE.

D (“Description”) Returns a description of the current condition. For HALT and SYNTAX conditions, this is the null string. For the ERROR or FAILURE condition, this is the command string that led to the error or failure. For the NOVALUE condition, this is the derived name of the variable involved.

I (“Instruction”) Returns CALL or SIGNAL, depending on the method used to invoke the signal handler.

S (“State”) Returns the state of the current condition: ON, OFF, or DELAY.

COPIES(string,n)

Returns *n* copies of *string* concatenated together.

```
copies('-',10)                  '——————'
copies('abc',3)                 'abcabcabc'
copies('',5)                    ''
```

COUNTSTR(needle, haystack)

Returns the number of occurrences of the string *needle* in the string *haystack*.

```
COUNTSTR('a', 'abABab')              2
COUNTSTR('aa', 'aaabbaa')            2
COUNTSTR('3', '123123123')           3
```

C2D(string)

Returns the decimal value of the internal representation of *string*. *String* is most often one character long, in which case this amounts to returning the decimal value of the character code for the specified character. An error occurs if the result cannot be expressed as a whole number according to the current NUMERIC DIGITS setting.

```
c2d('a')                97
c2d(1)                  49
c2d('ff'x)              255
c2d('yz')               31098
```

C2X(string)

Returns a string consisting of the two-character hexadecimal values of the character codes for each of the characters in *string*.

```
c2x('a')                61
c2x(1)                  31
c2x('ff'x)              'FF'
c2x('yz')               '797A'
```

DATATYPE(string[,option]**)**

If *option* is not specified, returns ‘‘NUM’’ if *string* is a valid KEXX number or ‘‘CHAR’’ if it is not. If *option* is specified, it is handled as follows:

A (‘‘Alphanumeric’’) Returns 1 if *string* consists entirely of alphabetic (‘‘A—Z’’, ‘‘a—z’’) and/or numeric (‘‘0—9’’) characters, else returns 0;

B (‘‘Binary’’) Returns 1 if *string* consists entirely of the characters ‘‘0’’ and ‘‘1’’, else returns 0;

L (‘‘Lowercase’’) Returns 1 if *string* consists entirely of lowercase alphabetic characters (‘‘a—z’’), else returns 0;

M (‘‘Mixed case’’) Returns 1 if *string* consists entirely of alphabetic characters (‘‘A—Z’’, ‘‘a—z’’), else returns 0;

N (‘‘Numeric’’) Returns 1 if *string* is a valid KEXX number, else returns 0;

S (‘‘Symbol’’) Returns 1 if *string* consists entirely of characters that are valid in KEXX symbols, else returns 0;

U (‘‘Uppercase’’) Returns 1 if *string* consists entirely of uppercase alphabetic characters (‘‘A—Z’’), else returns 0;

W (‘‘Whole number’’) Returns 1 if *string* is a whole number. That is, 1 is returned if *string* is a valid KEXX number that can be expressed as an integer, with no decimal point or exponential notation, under the current NUMERIC DIGITS setting. Otherwise returns 0;

X (‘‘heXadecimal’’) Returns 1 if *string* is the null string or consists entirely of valid hexadecimal digits (‘‘a—f’’, ‘‘A—F’’, ‘‘0—9’’), with optional blanks between pairs of hexadecimal digits. Otherwise returns 0.

For each of these options except ‘‘X’’ (which returns 1), 0 is returned if *string* is the null string.

```
datatype(16)                    'NUM'
datatype(16YY)                  'CHAR'
datatype(16,'N')                1
datatype(16.0,'N')              1
datatype('Kexx','A')            1
datatype('Kexx','L')            0
datatype('Kexx','U')            0
datatype('Kexx','M')            1
datatype('KEXX 5.5','A')        0
datatype('KEXX55','A')          1
datatype('KEXX','L')            0
datatype('KEXX','U')            1
datatype('KEXX','M')            1
datatype('','M')                0
```

```
datatype('KEXX','S')                1
datatype('1010','B')                1
datatype('10 10 ','B')              0
datatype('4e4f','X')                1
datatype('4e 4f','X')               1
datatype('4.3','W')                 0
```

DATE([option])

Returns information about the current date. Possible values for *option*, which defaults to "N", are:

B ("Base") Returns a unique serial number for each day (based on the number of days since 1 January 0001);

D ("Days") Returns the day of the year;

E ("European") Returns the date in the form *dd*/*mm*/*yy*;

J ("Julian") Returns the date in the form *yyddd*;

M ("Month") Returns the name of the current month;

N ("Normal") Returns the date in the form *dd mmm yyyy*, with no leading zero on the day of the month;

O ("Ordered") Returns the date in the form *yy*/*mm*/*dd*;

S ("Standard") Returns the date in the form *yyyymmdd*;

U ("USA") Returns the date in the form *mm*/*dd*/*yy*;

W ("Weekday") Returns the English name of the current day of the week.

Here are the values returned by the DATE() function on August 23, 1994:

```
date()                  '23 Aug 1994'
date('b')               728162
date('d')               235
date('e')               '23/08/94'
date('j')               94235

date('m')               'August'
date('n')               '23 Aug 1994'
date('o')               '94/08/23'
date('s')               19940823
date('u')               '08/23/94'
date('w')               'Tuesday'
```

DATECONV(date,input,[output]**)**

DATECONV converts *date* from one format to another and returns the result. The *input* argument specifies the current format of *date*, which can be "B", "D", "E", "J", "N", "O", "S", or "U". The *output* format can be "B", "D", "E", "J", "M", "N", "O", "S", "U", or "W" and defaults to "N". The formats involved are based on those used with the DATE() function. A null string is returned if *date* is not a valid date according to the specified *input* format.

```
dateconv('25 Dec 1994','n','w')          'Sunday'
dateconv('19940823','s','u')             '08/23/94'
dateconv('19950201','s')                 '1 Feb 1995'
```

When DATECONV() is used to convert a date that is specified using a 2-digit year, it assumes that the date falls within a 100 year sliding window starting at

```
(current year - 50)
```

and ending at

```
(current year + 49)
```

For example, in 2007, DATECONV would convert 2-digit years to 4-digit years in the range 1957–2056. So, in 2007, DATECONV would yield these results:

```
dateconv('12/17/02','u','n')             '17 Dec 2002'
dateconv('11/22/57','u','n')             '22 Nov 1957'
```

But in 2008, the sliding window would move by a year to cover 1958–2057. So, in 2008, DATECONV would yield these results:

```
dateconv('12/17/02','u','n')             '17 Dec 2002'
dateconv('11/22/57','u','n')             '22 Nov 2057'
```

DELIMIT(string1[,string2]**)**

Searches an internal list of valid KEDIT delimiter characters for a character that does not appear in *string1* and returns a string consisting of the delimiter character, *string1*, and the delimiter character. If *string2* is also specified, looks for a delimiter character that appears in neither string and returns a string consisting of the delimiter, *string1*, the delimiter, *string2*, and the delimiter. In the very rare situation where no valid delimiter is available, a null string is returned.

```
delimit('Hello')              '/Hello/'
delimit('///')                '!///!'
delimit('a/b','c/d')          '!a/b!c/d!'
```

Several KEDIT commands, like LOCATE and DIALOG, require as an operand a string delimited by special characters that do not appear in the string. Commands like the CHANGE command involve two strings and a delimiter that appears in neither string.

In a macro where the strings involved are variable and no ''safe'' delimiter can be determined in advance, you can use the DELIMIT() function to supply one.

```
* Build CHANGE command from two strings
say "Enter a string"
parse pull s1
say "Enter another"
parse pull s2
'CHANGE' delimit(s1,s2) 'ALL *'
```

DELSTR(string,n[,length])

Returns the value of *string* after deleting the characters starting at position *n* for a length of *length*. If *length* is not specified, the remaining characters of *string* are deleted.

```
delstr('I think, therefore, I am', 8)                    'I think'
delstr('I think, therefore, I am', 8, 12)                'I think I am'
```

DELWORD(string,n[,length])

Returns the value of *string* after deleting the words starting at word *n* for *length* words. If *length* is not specified, the remaining words are deleted. Trailing blanks after the last word deleted are also deleted. Any string of consecutive nonblank characters is considered to be a word.

```
delword('I think, therefore, I am', 3)               'I think, '
delword('I think, therefore, I am', 3, 1)            'I think, I am'
```

DIGITS()

Returns the current NUMERIC DIGITS setting.

```
digits()                9  /* default value */
```

DOSDIR([fileid][,[output][,search]])

Returns directory information for *fileid*.

The first argument is the *fileid* involved. If *fileid* includes a drive and/or path specification, DOSDIR() looks in the specified drive and directory; otherwise DOSDIR() looks in the current directory of the current drive. If *fileid* contains wildcards in the name or extension, information about the first matching file is returned and you can make subsequent calls to DOSDIR() in which *fileid* is omitted to process additional matching files.

If *fileid* is not found, a null string is returned. Otherwise the second argument, *output*, specifies the type of information to be returned and the order in which to return it. You can specify one or more of the following: 'N' to return the file's name and extension, 'S' to return its size, 'D' for its date, 'T' for its time, and 'A' for the

file's attributes; by default *output* is 'NSDTA' and all of this information is returned.

Information is returned in the following format: the name and extension of the file in the form *name.ext*, the size of the file in bytes, the file's date in the form *mm/dd/yyyy*, the file's time in the form *hh:mm:ss*, and the files attribute characters (one or more of "R" for a Read-only file, "H" for a Hidden file, "S" for a System file, "D" for a Directory, and "A" if the Archive bit is set; or a minus sign if none of these attributes applies).

The maximum value returned for the size of a file is 999999999, regardless of the actual size of the file involved.

The third argument, *search*, controls whether DOSDIR() processes certain special files. If *search* is omitted or is null, DOSDIR() omits directories, system files, and hidden files from its search and looks only for "normal" files. You can specify *search* as one or more of "D" (for directories), "S" (for system files), or "H" (for hidden files) to include these types of files in the search. This argument is ignored when the *fileid* argument is omitted and DOSDIR is searching for the next match for a previously-specified fileid pattern.

```
dosdir('test.c')              'TEST.C 5907 07/14/94 15:47:54 A'
dosdir('test.c','nd')         'TEST.C 07/14/94'

* total the sizes of all .TXT files in current dir
total = 0
size = dosdir('*.txt','s')
do while size \= ''
    total = total + size
    size = dosdir(,'s')
    end
say total
```

DOSENV(envvar)

Returns the value of the specified system environment variable *envvar*, or a null string if *envvar* is not present in the environment. A typical usage:

```
dosenv('path')          'C:\DOS;D:\UTIL;C:\KEDIT'
```

D2C(whole-number)

Returns the character string that corresponds to the internal binary representation of *whole-number*. Most often, *whole-number* is in the range 0 to 255 and this amounts to returning the character that has *whole-number* as its character code.

```
d2c(97)                 'a'
d2c(255)                'ff'x
d2c(1234)               '04D2'x
```

D2X(whole-number)

Converts *whole-number* to hexadecimal and returns the result.

```
d2x(0)                  0
d2x(80)                 50
d2x(255)                'FF'
d2x(1234)               '4D2'
```

ERRORTEXT(n)

Returns the text of KEXX error message *n*. If *n* is not the number of a KEXX error message, ERRORTEXT() returns the null string. KEXX error messages are currently in the range 93—134.

```
errortext(100)              'Control stack full'
errortext(500)              ''
```

FORMAT(number[,[before][,[after]]])

Formats *number* to have a specified number of digits *before* and *after* the decimal point and returns the result. The number is first rounded according to the current NUMERIC DIGITS setting.

If *before* is omitted, *number* is formatted with as many digits as necessary before the decimal point. If *before* is smaller than necessary, an error results; if *before* is larger than necessary, the result is padded on the left with blanks.

If *after* is omitted, *number* is formatted with as many digits as necessary after the decimal point. If *after* is smaller than necessary, *number* is rounded to the number of decimal places that you specify; if *after* is larger than necessary, the result is padded on the right with zeros. If *after* is zero, the result will be rounded to an integer and there will be no decimal point.

```
format(123.456)                     '123.456'
format(123.456,5,0)                 '  123'
format(-123.456,5,2)                ' -123.46'
format(-123.456,,4)                 '-123.4560'
```

FUZZ()

Returns the current NUMERIC FUZZ setting.

```
fuzz()                  0  /* default value */
```

INSERT(new,target[,[n][,[length][,pad]]])

Returns the result of inserting string *new* into string *target* after position *n*. The default for *n* is 0 (that is, the beginning of the string *target*). If necessary, prior to the insertion,

new will be truncated to a length of *length* or will be padded to that length with the *pad* character, which defaults to a blank.

```
insert('title', '************',6)              '******title******'
```

LASTPOS(string1,string2[,end])

Returns the starting position of the last occurrence of *string1* in *string2*. LASTPOS() only considers occurrences of *string1* that end at or before the *end* position of *string2*, where *end* defaults to the length of *string2*. Returns 0 if there are no such occurrences.

```
lastpos('I', 'I think, therefore, I am')                21
lastpos('I', 'I think, therefore, I am',10)             1
```

LEFT(string,length[,pad])

Returns the leftmost *length* characters of *string*. If *string* has less than *length* characters, the result will be padded on the right with the *pad* character, which defaults to a blank.

```
left('123',2)                 '12'
left('123',5,'0')             '12300'
```

LENGTH(string)

Returns the length of *string*.

```
length('314159')              6
length('a . b')               5
length('')                    0
```

LINEIN(fileid[,[line][,count]])

Returns the contents of a line of data read in from a file.

Input lines in the file are assumed to be terminated by a linefeed or by a carriage return-linefeed pair; this end-of-line sequence is not part of the value returned by LINEIN. After completion of the read operation, the read position for the file is moved to just beyond the end-of-line sequence. LINEIN returns the null string if no data can be read because the file does not exist, the read position for the file is at the end of the file or at an end-of-file character (character code 26), or because an I/O error is encountered.

Line and *count* are usually omitted. By default, LINEIN starts reading at the file's current read position, but you can specify *line* as 1 to indicate that LINEIN should begin reading at the start of the file; no other values are permitted for *line*. *Count* specifies the number of lines to be read in; it can be either 1, in which case 1 line is read in, or 0, in which case no data is read but the read position is reset to the start of the file if *line* is 1.

If *fileid* is not already open, it is first opened and the read position is set to the beginning of the file.

See page 374 for some general notes on KEXX's I/O functions.

```
linein('twoline.fil')          'First line of 2 line file'
linein('twoline.fil')          'Second line of 2 line file'
linein('twoline.fil')          ''
```

LINEOUT(fileid[,[string]])

Writes a line of data to a file. Returns 0 if successful and returns 1 if an I/O error or disk full condition occurs.

LINEOUT starts writing at the current write position for the file. If *fileid* is not already open, it is first opened and the write position is initially set to the end of the file, so that the file will be appended to. After the write operation completes, the write position for the file is moved just beyond the last character written.

If *string* is omitted, the file is closed.

See page 374 for some general notes on KEXX's I/O functions.

```
lineout('name.ext','Line of data')          0 /* if no error */
```

LINES(fileid)

Returns 1 if there are any remaining lines to be read in *fileid* and 0 if there are no remaining lines. That is, LINES returns 0 if *fileid* does not exist, or if the read position for *fileid* is positioned at the end of the file or at an end-of-file character (character code 26).

```
lines('name.ext')              0 /* if at eof, 1 if not */
```

LONGNAME(fileid)

Takes the *fileid* of an existing file and returns the long filename equivalent. That is, all components of *fileid* that are 8.3 MS-DOS short format aliases are converted to their long filename equivalents, and all components of *fileid* are converted to the correct case. If *fileid* does not exist, the null string is returned.

```
LONGNAME('C:\PROGRA~1\KEDITW\README.TXT')
 'C:\Program Files\KEDITW\ReadMe.txt'     /* perhaps */
```

LOWER(string**)**

Returns the value of *string* with any uppercase letters translated to lowercase. The characters "A—Z" are affected; accented characters with codes above 127 are not affected.

```
lower('ABCDEF')                 'abcdef'
lower('1F3De5')                 '1f3de5'
```

MAX(n1[,n2 ... n10]**)**

Returns the maximum number in the given list of up to ten numbers.

```
max(-1,3.2,4,5,0,-200,100)                          100
max(-1,-2,-4,-8.53,-16,-32.6,-64,-128)              -1
```

MIN(n1[,n2 ... n10]**)**

Returns the minimum number in the given list of up to ten numbers.

```
min(-1,3.2,4,5,0,-200,100)                          -200
min(-1,-2,-4,-8.53,-16,-32.6,-64,-128)              -128
```

OEMTOANSI(string**)**

Returns the result of converting *string*, which is assumed to be in the OEM character set, to the ANSI character set.

OVERLAY(new, target[,[n][,[length][,pad]]]**)**

Returns the result of overlaying the string *target* with the string *new* starting at position *n*, which defaults to 1. If necessary, prior to the operation, *new* is padded to a length of *length* with the *pad* character, which defaults to a blank.

```
overlay('dark', 'edge of night')            'dark of night'
overlay('dark', 'edge of night',9,5)        'edge of dark '
```

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*.

```
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
```

RANDOM(max)
RANDOM([min][,[max][,seed]])

Returns a pseudo-random number.

If only a single argument is given it specifies the maximum possible result, and the result will be in the range 0 through that number. Otherwise you can specify *min* and *max* (which default to 0 and 999), and the result will be in the range *min* through *max*.

If *seed* is specified, it will be used as the seed for a repeatable sequence of random numbers; otherwise an initial seed based on internal system counters is generated the first time that RANDOM() is called after the start of each command-level macro.

All arguments must be non-negative whole numbers, *max* minus *min* cannot exceed 100,000, and the result will always be a non-negative whole number.

```
random()                        942  /* range 0..999 */
random(16384)                   8453 /* range 0..16384 */
random(400,500,123)             467  /* repeatable, 400..500 */
```

REVERSE(string)

Returns the result of reversing the characters in *string*.

```
reverse('123456789')            '987654321'
```

RIGHT(string,length[,pad])

Returns the rightmost *length* characters of *string*. If *string* has less than *length* characters, the result will be padded on the left with the *pad* character, which defaults to a blank.

```
right('123',2)                  '23'
right('123',4,'0')              '0123'
right('123',10,'.')             '.......123'
```

SHORTNAME(fileid)

Takes the *fileid* of an existing file and returns the short, MS-DOS style, form of *fileid*. That is, all components of *fileid* are uppercased and are converted to their corresponding 8.3 MS-DOS short format alias. If *fileid* does not exist, or Windows is unable to process the conversion, the null string is returned.

```
SHORTNAME('C:\Program Files\KEDITW\ReadMe.txt')
 'C:\PROGRA~1\KEDITW\README.TXT'     /* perhaps */
```

SIGN(number)

Returns -1, 0, or 1 indicating respectively that *number* (after rounding according to the current NUMERIC DIGITS setting) is negative, zero, or positive.

```
sign(-3451)                    -1
sign(3451.22E14)                1
sign(0)                         0
```

SOURCELINE([n])

SOURCELINE(), with no arguments, returns the number of lines in the currently-executing KEXX macro.

SOURCELINE(*n*), where *n* can be no greater than the number of lines in the current macro, returns the text of the *n*th line of the macro.

SPACE(string[,[n][,pad]])

Replaces any strings of one or more inter-word blanks in *string* with *n* copies of the *pad* character and returns the result. The default value of *n* is 1 and the default value of *pad* is a blank. Leading and trailing blanks in *string* are removed.

```
space(' edge   of night ')                  'edge of night'
space(' edge   of night ',0)                'edgeofnight'
space(' edge   of night ',2,'-')            'edge–of–night'
```

STRIP(string[,[option][,char]])

Depending on whether *option* is "L" ("Leading"), "T" ("Trailing"), or "B" ("Both", the default), all leading, trailing, or both leading and trailing occurrences of *char* are deleted and the result is returned. The default for *char* is a blank.

```
strip('   title   ')                        'title'
strip('   title   ','L')                    'title  '
strip('   title   ','T')                    '   title'
strip('--title--',,'-')                     'title'
```

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,'-')              '—'
```

SUBWORD(string,n[,length])

Returns the substring of *string* starting at the *n*th blank-delimited word for a length of *length* blank-delimited words. The default for *length* is the remaining number of words in *string*. For example

```
subword('red blue green', 1, 2)             'red blue'
subword('red blue green', 3)                'green'
subword('red blue green', 4, 3)             ''
subword(' red   blue  green', 1, 2)         'red  blue'
```

SYMBOL(name)

Returns "BAD" if *name* is not a valid KEXX symbol, returns "VAR" if *name* is a variable to which a value has been assigned, and otherwise returns "LIT".

After

```
x = 17
a. = 'Hello'
drop a.17
drop y
```

the SYMBOL() function would yield:

```
symbol('==')            'BAD'
symbol('y')             'LIT' /* y has no assigned value */
symbol('a.x')           'LIT' /* a.17 has no assigned value */
symbol('x')             'VAR' /* x has an assigned value */
symbol(x)               'LIT' /* 17 is a literal */
```

TIME([option])

Returns a form of the current time depending on *option*. The value of *option*, which defaults to "N", may be one of the following:

C ("Civil") Returns the time in the format *hh*:*ss*am or *hh*:*ss*pm, where the hour does not have a leading zero;

H ("Hours") Returns the number of hours since midnight, from 0 to 23;

L ("Long") Returns the time in the format *hh*:*mm*:*ss*.*uuuuuu*, where *uuuuuu* is in microseconds, but is accurate only to the resolution of the system clock;

M ("Minutes") Returns the number of minutes since midnight, from 0 to 1439;

N ("Normal") Returns the time in the format *hh*:*mm*:*ss*;

S (“Seconds”) Returns the number of seconds since midnight, from 0 to 86399.

At one o’clock in the afternoon, the TIME() function would return these values:

```
time()                  13:00:00
time('c')               1:00pm
time('h')               13
time('l')               13:00:00.000000
time('m')               780
time('n')               13:00:00
time('s')               46800
```

The TIME() function provides two other options, used to control an elapsed timer facility:

E (“Elapsed”) If the elapsed timer has not been started in the current macro, starts the timer and returns 0. Otherwise, returns the elapsed time since the timer was started in the format *ssss.uuuuuu* (seconds, with no leading zeros, and microseconds, accurate to the resolution of the system clock).

R (“Reset”) If the elapsed timer has not been started in the current macro, starts the timer and returns 0. Otherwise, returns the elapsed time since the timer was started and resets the timer to 0.

Whenever execution of a macro begins, a new elapsed timer, local to that macro, is available. The value of the elapsed timer is preserved and restored across calls to internal routines.

```
* Time an operation with the elapsed timer
call time 'r'
do i = 1 to 10
   'get datafile.'i
end
say 'The elapsed time was' time('e') 'seconds'
```

TRACE([setting])

TRACE() with no arguments returns the trace setting currently in effect. That is, it returns 'A', 'C', 'E', 'F', 'I', 'L', 'N', 'O', or 'R', possibly preceded by '?'.

If specified, *setting* is a new trace setting to put into effect, and can be '?', or one of 'A', 'C', 'E', 'F', 'I', 'L', 'N', 'O', or 'R', optionally preceded by '?'.

TRANSLATE(string[,[tableo][,[tablei][,pad]]])

Informally, TRANSLATE() takes two lists of characters, an input list and an output list, searches a string for occurrences of characters in the input list and translates these occurrences to the corresponding character in the output list.

TRANSLATE() returns the value of *string* with the characters in *tablei* translated to the corresponding characters in *tableo*. If *tableo* is shorter than *tablei*, it is padded with

pad, which has a default value of blank. If *tableo* is longer than *tablei*, it is truncated to the length of *tablei*. The default value of *tablei* is the 256-character string beginning with '00'x and ending with 'FF'x. The default value of *tableo* is the null string. If *tablei* and *tableo* are omitted, *string* is translated to uppercase, as if you had used the UPPER() function.

```
translate('abcdwxyz')                              'ABCDWXYZ'
translate('abxyze', '12345', 'abcde')              '12xyz5'
translate('56/12/34', 'mmddyy', '123456')          'yy/mm/dd'
```

TRUNC(number[,n])

Truncates *number* to an integer followed by *n* digits after the decimal point and returns the result; *n* must be a non-negative number and defaults to 0. Before the truncation operation takes place, *number* is first rounded according to the current NUMERIC DIGITS setting.

```
trunc(12345.6789)                  12345
trunc(12345.6789,2)                12345.67
trunc(12345,3)                     12345.000
```

UPPER(string)

Returns the value of *string* with any lowercase letters translated to uppercase. The characters ‘‘a—z’’ are affected; accented characters with codes above 127 are not affected.

```
upper('abcdwxyz')              'ABCDWXYZ'
upper('90a1ff')                '90A1FF'
```

VALUE(name [,newvalue])

Returns the value of the symbol specified by *name*, and optionally assigns to the symbol the value specified by *newvalue*.

After

```
a = 's'
a9 = 17
s = '9'
```

the VALUE() function would yield:

```
value('a')                     's'
value(a)                       9
value('a'||s)                  17
value('a', 'Tom')              's'  /* and sets a = 'Tom' */
```

VERIFY(string,reference[,[option][,start]])

If *option* is "N" ("Nomatch") or is omitted, returns the position of the first character in *string* that does not match any character in the string *reference*, or returns 0 if there is no such character. If *option* is "M" ("Match"), returns the position of the first character in *string* that does match some character in the string *reference*, or 0 if there is no such character. Characters in *string* before the *start* position, which defaults to 1, are not considered.

```
verify('$1,000.00', '0123456789$,.')              0
verify('1000 Dollars', '0123456789$,.')           5
verify('1000.00', '$,.', 'm')                     5
verify('1000.00', '$,.', 'm' ,6)                  0
```

WORD(string,n)

Returns the *n*th blank-delimited word in *string*, or the null string if *string* contains less than *n* words.

```
word('red blue green yellow', 1)                  'red'
word('red blue green yellow', 4)                  'yellow'
word('   red   blue    green  yellow', 1)         'red'
word('red blue green yellow', 10)                 ''
```

WORDINDEX(string,n)

Returns the character position of the beginning of the *n*th blank-delimited word in *string*, or returns 0 if *string* contains less than *n* words.

```
wordindex('red blue green yellow', 1)                    1
wordindex('red blue green yellow', 4)                    16
wordindex('   red blue green yellow   ', 1)              4
wordindex('red blue green yellow', 10)                   0
```

WORDLENGTH(string,n)

Returns the length of the *n*th blank-delimited word in *string*, or returns 0 if *string* contains less than *n* words.

```
wordlength('red blue green yellow',1)             3
wordlength('red blue green yellow',3)             5
wordlength('red blue green yellow',5)             0
```

WORDPOS(phrase,string[,start])

If *phrase* is found in *string*, returns the word number in *string* of the first word of *phrase*. Otherwise, returns 0. *Start*, which defaults to 1, specifies the word within *string*

at which the search is to start. The blank-delimited words in *phrase* and *string* are compared, and the number of blanks separating the words does not affect the comparisons.

```
wordpos('red', 'blue green red yellow')                    3
wordpos('green red', 'blue green red yellow')              2
wordpos('green     red', 'blue green red yellow')          2
wordpos('green red', 'blue green red yellow',3)            0
wordpos('black', 'blue green red yellow')                  0
```

WORDS(string)

Returns the number of blank-delimited words in *string*.

```
words('in the heat of the night')                          6
words('   in     the heat of the    night   ')              6
words('intheheatofthenight')                               1
```

XRANGE([start][,end])

Returns a string consisting of the *start* character (which defaults to '00'x) and, in order, the characters whose codes follow it, through the *end* character (which defaults to 'FF'x). If *start* is greater than *end*, the sequence will wrap at 'FF'x, which will be followed by '00'x.

```
xrange(1,9)                    '123456789'
xrange('w','z')                'wxyz'
xrange()                       '00 01 02 ... fd fe ff'x
length(xrange())               256
xrange('fe'x,'01'x)            'fe ff 00 01'x
```

X2B(hex-string)

Converts a string in hexadecimal notation to binary notation and returns the result. Blanks may be included in *hex-string*, at byte boundaries.

```
x2b('0d')              '00001101'
x2b('E2')              '11100010'
x2b(d2x(4095))         '111111111111'
```

X2C(hex-string)

Takes a string containing a set of character codes expressed in hexadecimal notation and returns a string consisting of the corresponding characters. Blanks may be included in *hex-string*, at byte boundaries.

```
x2c('7a')              'z'
x2c(``)                ''
x2c('ff')              'ff'x
x2c('61 62 63')        'abc'
```

X2D(hex-string)

Converts a string in hexadecimal notation to decimal notation and returns the result. Blanks may be included in *hex-string*, at byte boundaries. An error occurs if the result cannot be expressed as a whole number according to the current NUMERIC DIGITS setting.

```
x2d('7a')              122
x2d(0)                 0
x2d('abcd')            43981
x2d('ab cd')           43981
```


6.8.2 Notes on I/O Functions

It is sometimes useful for a macro to directly read and write files on disk, as opposed to files that are in memory and being edited by KEDIT. KEXX includes I/O functions that you can use for this purpose. You can access disk files on a line-oriented basis by using the LINEIN(), LINES(), and LINEOUT() functions, or access files on a character-oriented basis by using the CHARIN(), CHARS() and CHAROUT() functions.

Line-oriented functions

The line-oriented functions assume that the files you are working with are organized into lines, with each line ended by a linefeed or by a carriage return-linefeed pair. Furthermore, the KEXX version of these functions works properly only with lines whose length is less than or equal to KEDIT's WIDTH value (which by default is 1024). For compatibility with other REXX implementations, these functions treat character code 26 as a special character indicating the end-of-file.

LINEIN(*fileid*) reads a line from the specified file and returns the contents of the line (with the end-of-line indicators removed) as its value. Data is read starting from the current read position for the file, which starts out at the beginning of the file and is updated after each input operation.

LINES(*fileid*) returns 1 if there are more lines of data in the specified file and 0 if there are not. That is, 0 is returned if the file does not exist, the read pointer is positioned at the end of the file, or the read pointer is positioned at an end-of-file character (character code 26).

LINEOUT(*fileid*,*string*) writes *string* (plus a carriage return-linefeed end-of-line sequence) to *fileid*. Output from LINEOUT() is normally appended to the end of the specified file; if you want to replace the existing contents of a file with new data, your macro should first use KEDIT's ERASE command to delete the existing file.

Character-oriented functions

The character-oriented functions treat the file as a continuous stream of characters and do no special processing of carriage returns, linefeeds, or end-of-file (character code 26) characters.

CHARIN(*fileid*[,[*start*][,*length*]]) reads data of a specified *length* (by default, 1 character) from *fileid*. If *start* is specified, data is read starting at that byte offset into the file; otherwise, data is read starting from the current read position in the file.

CHARS(*fileid*) returns the number of bytes of data remaining to be read in the specified file, starting from the current read position in the file, or 0 if you are positioned at the end of the file.

CHAROUT(*fileid*[,[*string*][,*start*]]) writes *string* to *fileid*. If *start* is specified, data is output starting at that byte offset into the file; otherwise data is written starting at the current write position in the file.

Read and write positions

Each file has a read position and a write position. The read position starts out at 1 (the beginning of the file) and is updated after each read operation to point just beyond the last character read. There is a single read position, updated by both CHARIN() and LINEIN(). You can use the *start* argument of CHARIN() to reposition the read pointer anywhere in the file.

The write position starts out at the end of the file, so that by default write operations will append to an existing file. If you instead want to replace the existing contents of a file with new data, your macro should first use KEDIT's ERASE command to delete the existing file. You can use the *start* argument of CHAROUT() to reposition the write pointer anywhere in the file, but repositioning the write pointer earlier in the file does not delete the existing contents of the file. There is a single write position, updated by both CHAROUT() and LINEOUT().

Opening and closing files

You do not need to explicitly open a file to begin using it; whenever you use an I/O function to refer to a file that is not currently open, the file will be automatically opened. Up to 10 files at a time can be open for use by the KEXX I/O functions.

Files are automatically closed when the macro in which they were opened terminates. You can explicitly close a file by calling the LINEOUT() or CHAROUT() functions and specifying the *fileid* argument but no other arguments. It is good practice for a macro to close a file explicitly when the macro has finished with it. In particular, you should be sure that any file that has been used with a KEXX I/O function within a macro is closed before you issue, from that same macro, any command (such as the KEDIT, GET, PUT, SAVE, or FILE command) that accesses the same file.

File size limits

The KEXX I/O functions cannot properly handle files larger than 2**32-1 (that is, 4294967295) bytes in size. Further, the CHARS() function will never return a value for the remaining bytes in a file larger than 999999999, and the largest allowable value for the *start* argument of the CHARIN() and CHAROUT() function, and for the *length* argument of the CHARIN() function, is 999999999.

Examples Here are two sample macros that illustrate how the I/O functions are typically used.

The first sample macro reads a disk file and inserts every non-blank line of that disk file into the current file:

```
fileid = 'test.fil'
do while lines(fileid) \= 0
    line = linein(fileid)
    if line \= '' then 'input' line
    end
call lineout fileid  /* close the file */
```

The second sample macro creates a disk file containing each of the words in the current file, with one word written to each line of the disk file:

```
fileid = 'words.fil'
'nomsg erase' fileid   /* erase file if it exists */
'top'
'down 1'
do while rc = 0
    line = curline.3()
    do j = 1 to words(line)
        call lineout fileid,word(line,j)
        end
    'down 1'
    end
call lineout fileid /* close the file */
```


6.8.3 Boolean Functions

Boolean functions are functions that test a condition within KEDIT and return 1 if the condition is true and 0 if it is false. An example:

```
if after() then 'sos firstch'
```

Here, if the cursor is located after the last nonblank character of the cursor line, the cursor is moved to the first nonblank character of that line.

The results from several Boolean functions depend on the contents of the current field. If the cursor is located on the top-of-file or end-of-file line, or on a shadow line, KEDIT will act as if the current field were empty.

Here are the available Boolean functions:

AFTER() Returns 1 if the cursor is positioned after the last nonblank character of the cursor field or if the cursor field is empty.

ALT() Returns 1 if the current file has been altered since the last SAVE (that is, returns 1 if the second number displayed by QUERY ALT is nonzero).

ALTKEY() Returns 1 if, at the time of the last keystroke or mouse action processed by KEDIT, either Alt key was down.

BEFORE() Returns 1 if the cursor is positioned before the first nonblank character of the cursor field or if the cursor field is empty.

BLANK() Returns 1 if the cursor field is completely blank.

BLOCK() Returns 1 if a block is marked in the current file.

BOTTOMEDGE() Returns 1 if the cursor is in the bottommost line of the file area.

BUTTON1() Returns 1 if mouse button 1 is down. Should only be used while processing a mouse click (that is, while the BUTTON1DOWN, BUTTON2DOWN, BUTTON1DBLCLK, or BUTTON2DBLCLK macros are active).

BUTTON2() Returns 1 if mouse button 2 is down. Should only be used while processing a mouse click (that is, while the BUTTON1DOWN, BUTTON2DOWN, BUTTON1DBLCLK, or BUTTON2DBLCLK macros are active).

CLASSIC() Returns 1 if INTERFACE CLASSIC is in effect.

CLIPTEXT() Returns 1 if any text is currently stored in the Windows clipboard for possible use in a Paste operation.

CMDSEL() Returns 1 if any text on the command line is currently selected.

COMMAND() Returns 1 if the cursor is on the command line.

CTRL() Returns 1 if, at the time of the last keystroke or mouse action processed by KEDIT, either Ctrl key was down.

CUA() Returns 1 if INTERFACE CUA is in effect.

CURRENT() Returns 1 if the cursor is on the current line.

DELSEL() If INTERFACE CUA is in effect, returns 1 when a selection or command line selection is marked, and when a persistent block is marked and no cursor movement has taken place since it was marked. Always returns 0 when INTERFACE CLASSIC is in effect. The default Delete and Backspace definitions use DELSEL() to determine whether they should use SOS DELSEL to delete the current block or command line selection, or should instead delete only a single character. DELSEL() returns 1 under the same conditions that the block indication on the status line is followed by an asterisk.

DIR() Returns 1 if the current file is the DIR.DIR file created via the KEDIT DIR command, or if .DIR is the extension of the current file.

END() Returns 1 if the cursor is on the last nonblank character of the cursor field.

EOF() Returns 1 if the cursor is on the end-of-file line.

FILELINE() Returns 1 if the cursor is not on the command line.

FIRST() Returns 1 if the cursor is positioned in column 1 of the cursor field.

FOCUSEOF() Returns 1 if the focus line is the end-of-file line.

FOCUSTOF() Returns 1 if the focus line is the top-of-file line.

FULLINP() Returns 1 if KEDIT is in Input Mode with INPUTMODE FULL in effect.

INBLOCK() Returns 1 if the cursor is in a marked block.

INITIAL() Returns 1 if the function is called while the profile for the first file in the ring is executing.

INPREFIX() Returns 1 if the cursor is in the prefix area.

INSERTMODE() Returns 1 if KEDIT is in Insert Mode.

INTRUNC() Returns 1 if the cursor is not to the right of the truncation column.

LEFTEDGE() Returns 1 if the cursor is in the leftmost column of the file area.

LINEINP() Returns 1 if KEDIT is in Input Mode with INPUTMODE LINE in effect.

MODIFIABLE() Returns 1 if the cursor is located in an area of the file that can be modified (that is, not on top-of-file, end-of-file, a shadow line, or to the right of the truncation column.)

MOUSEPOSMODIFIABLE() Returns 1 if, at the time of the last mouse button click or double-click, the mouse pointer was in the file area (that is, not in the prefix area or on the command line) and was in a position where it would be legal to type a character.

MOUSEPOSVALID() Returns 1 if, at the time of the last mouse button click or double-click, the mouse pointer was in a position where it would be legal to place the cursor.

MULTWINDOW() Returns 1 if more than one document window currently exists.

NOQUEUE() Returns 1 if no keystrokes are queued up within Windows waiting to be processed.

OEMFONT() Returns 1 if the font currently being used in KEDIT's document windows is an OEM font.

PENDING() Returns 1 if there are any pending prefix commands.

PREFIX() Returns 1 if PREFIX ON or PREFIX NULLS is in effect.

PREFIXLEFT() Returns 1 if the prefix area is displayed on the left of the window.

PROFILE() Returns 1 if the function is called while the profile for a file being added to the ring is executing.

REPEATING() Returns 1 if KEDIT is processing an auto-repeat keystroke. REPEATING() is 0 for the initial keystroke sent by Windows when you first press a key, but is 1 for each of the additional keystrokes repeatedly sent by Windows for as long as you hold the key down.

RIGHTEDGE() Returns 1 if the cursor is in the rightmost column of the file area.

SCROLLLOCK() Returns 1 if ScrollLock is in effect.

SHADOW() Returns 1 if the cursor is located on a shadow line.

SHIFT() Returns 1 if, at the time of the last keystroke or mouse action processed by KEDIT, either Shift key was down.

SHOWPRINTDLG() Returns 1 if the Print File toolbar button should show the File Print dialog box, and should not print immediately. That is, SHOWPRINTDLG() returns 1 when the ”Print File Toolbar Button Shows This Dialog” box in the File Print dialog box is checked.

SPACECHAR() Returns 1 if the character at the cursor position is blank.

TAB() Returns 1 if the cursor is located in one of the tab columns.

TOF() Returns 1 if the cursor is on the top-of-file line.

TOPEDGE() Returns 1 if the cursor is in the topmost line of the file area.

UNTITLED() Returns 1 if the current file is an UNTITLED file. That is, UNTITLED() returns 1 if the UNTITLED option was used when editing of the current file began, and the file's initial UNTITLED.*n* fileid has not been changed.

VERONE() Returns 1 if column 1 of the current file is being displayed in column 1 of the file area.

6.9 The PARSE Instruction

While PARSE is a keyword instruction, like DO, RETURN, or SAY, it is more complex than the other keyword instructions, so it is discussed in detail in this separate section.

PARSE takes character strings and assigns portions of their values to a set of KEXX variables according to a template that you provide. The format of the PARSE instruction is

PARSE [**UPPER**] *origin* [*template*]

UPPER is optional and specifies that the strings are to be converted to uppercase before being parsed according to the template.

Origin specifies where the PARSE instruction is to obtain the data to be parsed.

Template is a set of pattern specifications that control the parsing process, intermixed with lists of variables to which the parse data is to be assigned.

PARSE origins There are several ways to specify the origin of the string to be parsed:

PARSE [UPPER] ARG [*template***]**
When executed from a KEXX macro while no internal routine is active, PARSE ARG parses the argument string passed to the macro when it was invoked. When executed from an internal routine within a KEXX macro, PARSE ARG parses the arguments to that internal routine.

PARSE [UPPER] LINEIN [*template***]**
KEDIT reads a line of user input from the command line and the resulting line of input is parsed.

PARSE [UPPER] PULL [*template***]**
KEDIT reads a line of user input from the command line and the resulting line of input is parsed. In KEXX, PARSE LINEIN and PARSE PULL are equivalent. (Both are included in KEXX for compatibility with REXX, where PARSE LINEIN always reads a line of user input but PARSE PULL first tries to read from a REXX facility known as the ”external data queue”, and reads a line of user input only if this queue is empty. KEXX does not provide an external data queue, so PARSE PULL always acts as if it is empty and always reads a line of user input.)

PARSE [UPPER] SOURCE [*template***]**
KEXX parses a string that gives information about the currently executing macro. This consists of the system under which KEDIT is running (for KEDIT for Windows, this is “Windows”), the method used to invoke the macro (always “COMMAND” in this version of KEDIT), and the name of the macro.

For example, if you issue the command

```
macro test
```

and use PARSE SOURCE within the macro, it would parse the string “Windows COMMAND test”.

PARSE [UPPER] VALUE [*expression***] WITH [***template***]**
KEXX takes the value of *expression* and uses it as the string to be parsed. The keyword WITH is used to separate the *expression* from the *template*. For example,

```
parse value date() time() with template
```

would parse a string like “18 Jun 2007 11:37:52”.

PARSE [UPPER] VAR ***name*** **[***template***]**
KEXX takes the value of the specified variable as the string to be parsed. For example,

```
x = "tuna fish"
parse var x template
```

would parse the string “tuna fish”.

PARSE [UPPER] VERSION [*template*]

KEXX parses a string with information about the current version and release date of the KEXX language processor. This consists of the name of the language (''KEXX''), the KEXX version number (for example, ''5.61'', the version included in the initial release of KEDIT for Windows 1.6), and the date of this KEXX version (in the form ''*dd Mmm yyyy*''). For example, PARSE VERSION would parse a string like ''KEXX 5.61 15 Aug 2007''.

PARSE templates

PARSE templates consist of specifications that tell KEXX how to break the data being parsed into smaller substrings and lists of variables to which the data in these substrings is assigned.

The simplest templates consist only of a list of variables, to which all of the data in the string being parsed is assigned.

```
parse value 'Clark Kent is Superman.' with v1 v2 v3
```

In this example, ''Clark Kent is Superman.'' is the string being parsed and the template consists of a list of variables, *V1*, *V2*, and *V3*.

Variable lists

Data is assigned to variables in a list in this fashion: For all variables in the list except the last variable, leading blanks are removed from the data, the next word (that is, the next set of nonblank characters) is removed from the data and assigned to the next variable in the list, and then the first trailing blank (if any) after the word is removed from the data. When the last variable in the list is reached, all remaining data (which may include leading and trailing blanks) is assigned to it. If the end of the data is reached before all variables in the list have been assigned values, the remaining variables in the list are set to the null string.

This example, in which there is a single blank between each word of the string to be parsed, sets these variables:

```
parse value 'Clark Kent is Superman.' with v1 v2 v3
```

sets these variables:

```
v1    'Clark'
v2    'Kent'
v3    'is Superman.'
```

The next example has two blanks between each word of the string to be parsed:

```
parse value 'Clark  Kent  is  Superman.' with v1 v2 v3
```

It sets these variables:

```
v1    'Clark'
v2    'Kent'
v3    ' is  Superman.'
```

A list of variables can include periods that serve as dummy variables. The value that would normally be assigned to the variable at the period's position in the list is not assigned to any variable but is simply ignored, as ''Kent'' is in the following example:

```
parse value 'Clark Kent is Superman.' with v1 . v2

v1    'Clark'
v2    'is Superman.'
```

Parse specifications

To process a template, PARSE moves from left to right through the template, skipping over lists of variables but stopping at each parse specification. PARSE uses the parse specification to pick out a substring of the parse data. Then, if the parse specification is preceded by a list of variables, this substring is assigned to the list of variables by the process described above (that is, one word per variable except the last variable in the list, which gets the rest of the substring). If the last parse specification is followed by a list of variables, the remainder of the parse string is assigned to the variables in the list according to the same process.

Types of parse specifications are:

Absolute positional patterns

Absolute positional patterns are unsigned integers, optionally preceded by an equal sign (''='') , that specify a specific position in the parse data.

```
parse value 'Clark Kent is Superman.' with v1 10 v2 15 v3

v1    'Clark Ken'
v2    't is '
v3    'Superman.'
```

Relative positional patterns

Relative positional patterns are integers preceded by a plus sign (''+'') or by a minus sign (''-''). They specify a position a certain number of characters after or before the column matched by the last parse specification (or, when there is no preceding parse specification, from the beginning of the string).

```
parse value 'Clark Kent is Superman.' with v1 10 v2 +5 v3

v1    'Clark Ken'
v2    't is '
v3    'Superman.'
```

Positional patterns that specify a position beyond the end of the parse data are considered to match the end of the data. Positional patterns that specify a position to the left of the beginning of the parse data are considered to match the beginning of the parse data.

```
parse value 'Clark Kent is Superman.' with 0 v1 10 v2 40 v3

v1    'Clark Ken'
v2    't is Superman.'
v3    ''
```

When a position at or to the left of the current position in the parse string is specified by a positional pattern, it sets the new position in the parse data, and uses all data from the previous position through the end of the data to set any preceding variable list.

```
parse value 'Clark Kent is Superman.' with v1 10 v2 -3 v3

v1    'Clark Ken'
v2    't is Superman.'
v3    'Kent is Superman.'
```

Literal patterns

Character strings enclosed in quotes are literal pattern specifications. PARSE starts from its current position in the PARSE data and matches the first occurrence of the pattern string that it finds.

```
parse value 'Clark Kent is Superman.' with v1 'Kent' v2

v1    'Clark '
v2    ' is Superman.'
```

When a relative positional pattern follows a literal pattern, the new position is determined relative to the first character matched by the literal pattern. Also, the substring formed by the positional pattern and the preceding literal pattern includes all characters from the first character matched by the literal pattern up to the column specified by the relative positional pattern; in all other situations, the string matched by a literal pattern is not included in this substring.

```
text = 'Clark Kent is Superman.'
parse var text v1 'Kent' v2 +8 v3

v1    'Clark '
v2    'Kent is '
v3    'Superman.'
```

Variable patterns

Pattern information can come from variables rather than being directly embedded in the template. Variable patterns are indicated by variable names in parentheses. If the parentheses are preceded by an equal sign (“=”), the value of the variable must be an integer and its value is used as an absolute positional pattern. If the parentheses are preceded by a plus sign (“+”) or minus sign (“-”), the value of the variable must also be an integer and its value is used as a relative positional pattern. Parentheses alone, not preceded by an equal sign, minus sign, or plus sign, mean that the value of the variable is to be used as a literal pattern.

```
n = 3; s = 'Super'
text = 'Clark Kent is Superman.'
parse var text v1 =(n) v2 +(n) v3 (s) v4

v1    'Cl'
v2    'ark'
v3    ' Kent is '
v4    'man.'
```

Comma patterns

A comma (“,”) in a template matches the end of the string currently being parsed and causes processing to continue at the start of the next parse string. Except when processing PARSE ARG (and the ARG instruction), there is only one string to parse and a comma in a template is not useful. But with PARSE ARG and with the ARG instruction, when executed from inside an internal routine that was passed multiple parameters, all of the arguments to the routine can be processed with a single template that uses commas. For example, assume that an internal routine had been passed the three arguments “Clark Kent”, “Lois Lane”, and “Jimmy Olsen”:

```
parse arg v1, v2, v3 v4

v1    'Clark Kent'
v2    'Lois Lane'
v3    'Jimmy'
v4    'Olsen'
```

6.10 Conditions

Condition handling, discussed in this section, is an advanced feature of KEXX that most macro writers need not be familiar with.

You can have a KEXX macro automatically transfer control to a different location in your macro when certain special conditions occur. This allows your macro to, for example, display diagnostic information after a syntax error occurs, or ask a user of the macro who has pressed Ctrl+Break whether to terminate the macro.

The conditions that can be handled are:

The HALT condition, triggered when Ctrl+Break is pressed.

The FAILURE condition, triggered when a command yields a negative return code.

The ERROR condition, triggered when a command yields a positive return code, or when a command yields a negative return code and the FAILURE condition is not enabled. Note that the ERROR condition is not usually useful in KEXX macros, since several KEDIT commands (such as a DOWN command that reaches the End-of-File line) generate positive return codes in non-error situations.

The SYNTAX condition, triggered by a syntax error in your macro, or by invalid macro operations like division by zero or passing invalid arguments to a built-in function.

The NOVALUE condition, triggered by the use of an uninitialized variable in your macro.

By default, user-specified condition handling is disabled and the above conditions are handled as follows: Ctrl+Break and syntax errors cause termination of your macro. Error and failure return codes are ignored. Use of uninitialized variables is ignored if

SET NOVALUE OFF is in effect, and causes termination of your macro if SET NOVALUE ON is in effect.

Condition handling is most often enabled via this form of the SIGNAL instruction:

SIGNAL ON *condition* [NAME *trapname*]

When SIGNAL ON has been executed for a condition and that condition occurs, control is transferred, as if SIGNAL *label* had been executed, to the label specified by *trapname* or, if NAME *trapname* was omitted, to a label corresponding to the name of the condition.

For example, with

```
SIGNAL ON HALT
```

KEXX will transfer control to the label HALT if Ctrl+Break is pressed while your macro is executing.

With

```
SIGNAL ON HALT NAME BREAKER
```

KEXX will transfer control to the label BREAKER if Ctrl+Break is pressed while your macro is executing.

You can use

SIGNAL OFF *condition*

to disable user-specified condition handling and restore the default behavior.

With SIGNAL ON, control is transferred to the appropriate label when a condition is raised, but there is no convenient way of resuming execution of your macro at the point that the condition was detected. An alternative is to use

CALL ON *condition* [NAME *trapname*]

which calls your condition handler as a subroutine and, if your subroutine returns, resumes execution immediately following the point where the condition was triggered.

You can use

CALL OFF *condition*

to disable this form of user-specified condition handling.

CALL ON *condition* can be used with the HALT, ERROR, and FAILURE conditions, but cannot be used with the SYNTAX and NOVALUE conditions.

Notes

When a condition with SIGNAL ON handling in effect occurs, and before control is transferred to the condition handler, SIGNAL OFF is put into effect for that condition, so that the default behavior will take place if the condition occurs again and user-specified condition handling has not been re-enabled.

When a condition with CALL ON handling in effect occurs, and before control is transferred to the condition handler, the condition is put into a delayed state, and will be re-enabled on return from your handler. You can also use SIGNAL or CALL instructions within your handler to enable or disable condition handling while your handler is active.

While your handler is active, you can use the CONDITION() function to get information about the current trapped condition.

For the ERROR and FAILURE conditions, the RC variable is set to the command return code involved before control is transferred to the condition handler. For the SYNTAX condition, the RC variable is set to the message number for the error involved; the text of the message can be retrieved via the ERRORTEXT() function.

The SIGL variable is set before control is transferred to your condition handler to the line number of the last clause executed.

6.11 KEXX and REXX

KEDIT's macro language, KEXX, contains a large subset of the REXX language. Almost all of the features of REXX, as documented in *The REXX Language: A Practical Approach to Programming* by Michael Cowlishaw (Second Edition, Prentice-Hall, 1990) are available in KEXX. Here are the primary differences between KEXX and REXX:

KEXX clauses and comments must fit completely on a single line, and cannot be continued on additional lines.

Lines of a KEXX macro are limited to 1024 characters.

KEXX comment lines that begin with an asterisk ("*") are not valid in REXX programs. The other form of KEXX comment, delimited by slash-asterisk ("/*") asterisk-slash ("*/") pairs, are valid in REXX programs.

The maximum NUMERIC DIGITS value supported by KEXX is 1000.

KEXX does not support the following REXX instructions: ADDRESS, PUSH, and QUEUE.

Control variables in KEXX DO loops must be simple symbols, and the values of the control variable and the expressions involved in DO loop control must be whole numbers with at most 9 digits.

TRACE + and TRACE -, used to turn interactive tracing on and off, are supported in KEXX but not in REXX, which supports only TRACE ?, which toggles the status of interactive tracing.

The STREAM() built-in function is not available in KEXX.

KEXX does not support the NOTREADY condition.

Labels in KEXX programs are only valid when they are the first token on a line.

KEXX's LINEIN(), LINEOUT(), LINES(), CHARIN(), CHAROUT(), and CHARS() functions require that their first operand be specified. REXX lets you do I/O to the console by omitting this argument.

The following REXX features are supported in KEXX, but are not documented here because they are rarely-used and not of interest to most KEDIT users: the second argument ("*n*") of the C2D(), D2C(), and X2D() functions, the *expp* and *expt* arguments of the FORMAT() function, the NUMERIC FORM instruction, the FORM() function, SIGNAL VALUE, and TRACE VALUE. See the Cowlishaw book for details about these items.

For compatibility with older versions of KEDIT, KEXX accepts the characters "@", "#", and "$" as valid in symbols and lets you use "^" and "~" to indicate negation. These characters are not included in the latest REXX language definition and are considered invalid by some REXX implementations, so their use is discouraged.

Chapter 7. Built-In Macro Handling

7.1 Overview

While you can use the MACRO command to run macros directly from KEDIT's command line, KEDIT macros are most often executed in response to some interaction with KEDIT's user interface. Whenever you press a key, select a toolbar button or menu item, or click a mouse button, KEDIT runs a macro associated with that action. Most of the macros involved are built into KEDIT, but you can use the DEFINE command to redefine these macros and change KEDIT's behavior.

This chapter has information on the naming conventions used in the processing of keyboard, toolbar, menu, and mouse macros.

You can use the MACROS command to see the definitions of all currently-defined macros; this will include any macros that you have defined, as well as any default macros that you have not redefined. To see the default macros that are built into KEDIT, you can run KEDIT with the NOPROFILE option, so that only the default macros are defined, and then use the MACROS command.

Another way to see KEDIT's default macro definitions is to look at the file BUILTIN.KML, which is installed by the KEDIT for Windows SETUP program in the SAMPLES subdirectory of the main KEDITW directory. If you decide to change any of KEDIT's default macro definitions, we recommend that you use separate DEFINE commands or KML files for your modified definitions, and do not load the entire BUILTIN.KML file from within your profile. This is because loading the entire file would be time consuming and redundant, since the macros involved are already built into the KEDIT module, and because if any of KEDIT's default definitions change in future releases of KEDIT, you may be loading in obsolete macro definitions from your modified version of BUILTIN.KML.

7.2 Keyboard Macros

KEDIT has names for each key and key combination that it can read from the keyboard. Whenever you press a key or a key combination recognized by KEDIT, KEDIT determines the name of the key that was pressed and executes the corresponding macro.

For example, when you press the F1 key, KEDIT runs the F1 macro, which by default issues the command HELP, which in turn causes KEDIT to display its Help file. And when you press the Shift+F1 key, KEDIT runs the SHIFT+F1 macro, which by default issues the LOCATE command with no operands, causing KEDIT to re-execute the most-recent LOCATE command.

KEDIT even has built-in macros corresponding to the character keys on your keyboard. For example, when you press the "A" key, KEDIT runs a macro called "A", which issues the command "text a" to enter a lowercase "a" at the cursor position.

When you press Shift+A, KEDIT runs a macro called SHIFT+A, which issues the command ‘‘text A’’ to enter an uppercase ‘‘A’’ at the cursor position.

Naming conventions

The system used for key names is straightforward. For most keys, the key name is simply the name that appears on the key. For keys pressed in combination with the Shift, Ctrl, or Alt keys, key names are prefixed with ‘‘Shift+’’, ‘‘Ctrl+’’, or ‘‘Alt+’’ (which can also be given in shorter form as ‘‘S+’’, ‘‘C+’’, or ‘‘A+’’). For compatibility with older versions of KEDIT, you can use ‘‘-’’ instead of ‘‘+’’ as the separator within keynames. So, for example, ‘‘Ctrl+End’’, ‘‘Ctrl-End’’, ‘‘C+End’’, and ‘‘C-End’’ are all equivalent. You can give key names in any combination of uppercase and lowercase.

For example, ‘‘J’’ (or ‘‘j’’) names the alphabetic key that normally enters a lowercase ‘‘j’’ at the cursor position. ‘‘Shift+J’’ (or ‘‘Shift+j’’, or ‘‘S+J’’, etc.) names the shifted version of this key, which normally enters an uppercase ‘‘J’’ at the cursor position.

KEDIT also handles key combinations involving Shift and Ctrl pressed in combination with another key. For example, ‘‘Shift+Ctrl+A’’, ‘‘Ctrl+Shift+A’’, ‘‘C+S+A’’, and ‘‘S+C+A’’ are all valid and all refer to the same key combination. Similarly, KEDIT handles Alt and Ctrl pressed in combination with another key, as in ‘‘Ctrl+Alt+A’’ or ‘‘Alt+Ctrl+A’’.

The key names accepted by KEDIT are listed in the table on the next page. Some key combinations are not available, because they are given special handling by Windows. For example, Ctrl+Esc always brings up the Windows Task List window, and its behavior cannot be changed by KEDIT. And pressing Shift+Space (that is, pressing the Shift key in combination with the space bar) is treated the same as pressing Space alone.

A...Z	Shift+A...Z	Ctrl+A...Z	Alt+A...Z
0...9	Shift+0...9	Ctrl+0...9	Alt+0...9
'	Shift+'	Ctrl+'	Alt+'
-	Shift+-	Ctrl+-	Alt+-
=	Shift+=	Ctrl+=	Alt+=
[	Shift+[	Ctrl+[	Alt+[
]	Shift+]	Ctrl+]	Alt+]
\	Shift+\	Ctrl+\	Alt+\
;	Shift+;	Ctrl+;	Alt+;
'	Shift+'	Ctrl+'	Alt+'
,	Shift+,	Ctrl+,	Alt+,
.	Shift+.	Ctrl+.	Alt+.
/	Shift+/	Ctrl+/	Alt+/
Space			Alt+Space
Enter		Ctrl+Enter	Alt+Enter
Tab	Shift+Tab	Ctrl+Tab	
Esc			
Bksp		Ctrl+Bksp	Alt+Bksp
F1...F12	Shift+F1...F12	Ctrl+F1...F12	Alt+F1...F12
Home	Shift+Home	Ctrl+Home	Alt+Home
Pgup	Shift+Pgup	Ctrl+Pgup	Alt+Pgup
Pgdn	Shift+Pgdn	Ctrl+Pgdn	Alt+Pgdn
Curu	Shift+Curu	Ctrl+Curu	Alt+Curu
Curd	Shift+Curd	Ctrl+Curd	Alt+Curd
Curl	Shift+Curl	Ctrl+Curl	Alt+Curl
Curr	Shift+Curr	Ctrl+Curr	Alt+Curr
End	Shift+End	Ctrl+End	Alt+End
Ins	Shift+Ins	Ctrl+Ins	Alt+Ins
Del	Shift+Del	Ctrl+Del	Alt+Del
Center	Shift+Center	Ctrl+Center	
Plus		Ctrl+Plus	Alt+Plus
Minus		Ctrl+Minus	Alt+Minus
Slash		Ctrl+Slash	Alt+Slash
Star		Ctrl+Star	Alt+Star
NumEnter		Ctrl+NumEnter	Alt+NumEnter
App	Shift+App	Ctrl+App	Alt+App
Alt			

Notes on key names

The table does not specifically list Shift+Ctrl+*key* and Alt+Ctrl+*key* key combinations. Wherever Ctrl+*key* is valid, Shift+Ctrl+*key* is also valid. Wherever Alt+*key* is valid, Alt+Ctrl+*key* is also valid, aside from Alt+Ctrl+Del, which is given special handling by the operating system.

''Space'' is the name of the space bar. ''Bksp'' is the name of the Backspace key.

''Curl'', ''Curr'', ''Curu'', and ''Curd'' are the names of the cursor left, right, up, and down keys.

''Plus'', ''Minus'', ''Star'', ''Slash'', ''NumEnter'', and ''Center'' are the names of the numeric keypad ''+'', ''-'', ''*'', ''/'', Enter, and ''5'' keys.

''App'' is the name of the Application key found on some keyboards with Windows-specific keys.

The last keyname in the list, ''Alt'', refers to the Alt key, when pressed and released alone and not in combination with any other key.

When you define your own keyboard macros, be sure to use the key names given in the preceding table. For example, ''Esc'' is the name of the macro that is run when you press the key that is normally referred to as the Escape key. ''Escape'' is a valid name for a KEDIT macro, so it is not an error to define a macro with that name. But ''Esc'', and not ''Escape'', is the name of the macro that will be executed if you press the Escape key.

Most keyboards have a separate cursor pad and numeric pad. KEDIT does not normally distinguish between the two sets of keys. For example, if you redefine ''Home'' or ''Ctrl+Home'', your new definition will affect both the cursor pad Home key and the numeric pad Home key. Alt key combinations are an exception: when you press the Alt key in combination with a key on the cursor pad, the expected definition (for example, Alt+Home) is processed, while Alt in combination with the numeric pad is the "Alt key-numeric pad" method of entering special characters.

See SET RIGHTCTRL for information about how to use the right Ctrl key as the equivalent of the NumEnter key.

READV KEY and QUERY/EXTRACT LASTKEY always return key names in uppercase, with possible ''C-'', ''S-'', ''A-'', ''S-C-'', or ''A-C-'' prefixes; ''-'' is used in the prefix rather than ''+'' for compatibility with earlier versions of KEDIT.

Notes on non-U.S. keyboards

KEDIT's keyboard naming scheme is based on the layout of the standard U.S. keyboard, and users of non-U.S. keyboards should be aware of a few special considerations:

Special characters not found on U.S. keyboards, such as accented letters, are handled via the ASCII macro, which is discussed immediately following these notes.

Special characters that are located in different positions on U.S. and non-U.S. keyboards are mapped into their location on the U.S. keyboard. To redefine the behavior of one of these special characters you would need to refer to the character according to its location on the U.S. keyboard.

For example, the "#" character is present on both the U.S. and British keyboards. On the U.S. keyboard it is located on the same key as the digit ''3'', and is entered by pressing Shift+3. On the British keyboard, this character is on a key of its own, located next to the Enter key. In both cases, when KEDIT sees that you have pressed the key corresponding to the "#" character, KEDIT runs the macro that corresponds to this character's position on the U.S. keyboard: Shift+3. The default Shift+3 macro uses the TEXT command to enter the "#" character at the cursor

position. To redefine the behavior of the key that normally enters "#" into your file, you would therefore need to redefine the Shift+3 macro.

With non-U.S. keyboard drivers, Windows gives special handling to Alt+Ctrl, treating it as equivalent to the AltGr key found on non-U.S. keyboards. You should therefore not redefine Alt+Ctrl+*x* character keys that have additional characters normally accessed in combination with the AltGr key.

Special characters and the ASCII macro

Special characters not found on U.S. keyboards (such as accented letters found on most non-U.S. keyboards), digits entered via the numeric pad, and characters entered via the Alt key-numeric pad method of entering special characters do not correspond directly to any of KEDIT's defined key names. When KEDIT reads one of these characters from the keyboard, the character code involved, in decimal, is passed to a special macro called ASCII. (Many of the special characters involved are not defined in the ASCII character set and are in fact based on the ANSI character set, but for compatibility with earlier versions of KEDIT the macro is still called the ASCII macro.)

For example, if you use the Alt key-numeric pad method to enter character code 12, KEDIT will internally issue the command

```
MACRO ASCII 12
```

The default definition for the macro ASCII uses KEDIT's TEXT command to enter the character involved at the cursor location. If the character was entered via the Alt key-numeric pad method, or if you are using an non-OEM font, the character is entered directly. If you are using an OEM font, the default ASCII macro first translates the character from ANSI (the character set used by Windows for keyboard input) to OEM, and enters the resulting character.

7.3 Toolbar Macros

Toolbar buttons are defined via the SET TOOLBUTTON command. To define a toolbar button, you specify a name for the button, how the button will be displayed (that is, what bitmap or text will represent the button on the toolbar), any conditions under which the button will be disabled (for example, the Undo toolbutton is disabled when no undoable changes have been made to the file), and the help text to be displayed as popup toolbar help and on the status line.

You can then use the SET TOOLSET command to define the contents of KEDIT's top or bottom toolbar, or to add buttons to or remove buttons from the current contents of the toolbar.

See the descriptions of SET TOOLBAR and SET TOOLSET for full information on the operands for these commands, and for examples of their use.

Whenever you select one of the buttons on KEDIT's toolbar, KEDIT runs a macro called TOOL_*name*, where *name* is the name of the button, as defined in the SET TOOLBUTTON command. For example, whenever you select the Undo toolbar button, KEDIT runs a macro called TOOL_UNDO. If you add your own buttons to

KEDIT's toolbar, you will need to use the DEFINE command to supply KEXX macros to be executed when those buttons are selected.

There is one exception to this: the Quick Find toolbar item is not a button, but is a combo box that lets you work with recently-used search strings, and its behavior is hard-coded into KEDIT and is not controlled by a macro.

Built into KEDIT are TOOLBUTTON definitions for all of the buttons used on KEDIT's default toolbars, as well as bitmaps and TOOL_*name* macros for each of those buttons.

You can use the command QUERY TOOLBUTTON * to see a list of all current toolbutton definitions. You can use QUERY TOOLSET TOP, BOTTOM, or NOFILE to see the current SET TOOLSET values. Since the TOOLSET values may be too wide to be easily displayed; you may need to use MODIFY TOOLSET TOP, BOTTOM, or NOFILES to get the TOOLSET values to the command line where they can be more easily edited or preceded by INPUT and inserted into a file.

The file DEFTOOLB.KEX, in the SAMPLES subdirectory of the main KEDITW directory, has commands corresponding to the default SET TOOLSET and SET TOOLBUTTON definitions.

7.4 Menu Macros

The contents of KEDIT's menus are built into the program and cannot be changed. While you cannot change the menus themselves, you can redefine the macros that are executed when menu items are selected, although in practice the default menu macros rarely need to be changed.

When a menu item is selected, KEDIT runs a macro called MENU_*menu_item*, where *menu* is the name of the menu involved, and *item* is the text of the menu item involved, with all blanks removed. For example, when you select Save Settings from the Options menu, KEDIT runs the macro MENU_OPTIONS_SAVESETTINGS.

Exceptions to this are the document and frame window system menus, the recently-edited files list at the bottom of the File menu, and the window list at the bottom of the Window menu. The behavior of these menu items is hard-coded into KEDIT and is not controlled by macros.

7.5 Mouse Macros

Mouse clicks and double-clicks within a document window cause KEDIT to run a corresponding macro, called BUTTON1DOWN, BUTTON2DOWN, BUTTON1DBLCLK, or BUTTON2DBLCLK.

When you double-click a mouse button, KEDIT processes both the initial click and the double-click, and runs both of the corresponding macros. For example, if you double-click button 1, KEDIT runs BUTTON1DOWN and then runs BUTTON1DBLCLK.

Built into KEDIT are default definitions for each of these macros. These are by far the most complicated of KEDIT's built-in macros, and we recommend that they be modified only by advanced users of KEDIT's macro facilities. Note that these macros tend to change significantly between one version of KEDIT and the next as KEDIT adapts to new user interface conventions, so you should reprogram these macros only if you are willing to deal with changes in future versions of KEDIT.

Also built into KEDIT are some helper macros: BUTTONXDOWN is called by BUTTON1DOWN and BUTTON2DOWN, BUTTONXDBLCLK is called by BUTTON1DBLCLK and BUTTON2DBLCLK, and BUTTON2POPUP is called by BUTTONXDOWN.

Chapter 8. KEDIT Language Definition Files

KEDIT's syntax coloring facility uses different colors to highlight comments, strings, keywords, and other items in programs that you are editing. The rules that KEDIT uses to determine which text to treat as part of a comment, a string, a keyword, etc., are specified in special files, called KLD (''KEDIT Language Definition'') files, that are described in this chapter.

8.1 Loading KLD Files

KEDIT Language Definition files are loaded via the SET PARSER command:

```
[Set] PARSER parser fileid
```

Use the *parser* operand to specify the name of the parser you want to define.

The *fileid* operand specifies a file, with a default extension of .KLD, containing your language definition. KEDIT searches for the .KLD file in the same directories it uses when searching for macro files, as controlled by SET MACROPATH.

For example, if you were working with a hypothetical language called LANG and you had described the language in a KEDIT Language Definition file called LANGDEF.KLD, you could define a parser called LANG with the command

```
SET PARSER LANG LANGDEF.KLD
```

After issuing the SET PARSER command, you could then issue the command

```
SET COLORING ON LANG
```

to use this parser to control syntax coloring for the current file.

If files in your language always had an extension of, for example, .LNG, you could use the SET AUTOCOLOR command to tell KEDIT to always use the LANG parser for .LNG files:

```
SET AUTOCOLOR .LNG LANG
```

SET PARSER commands are typically executed from your KEDIT profile when KEDIT is initially loaded. For example:

```
* if first profile execution in a session,
* setup the LANG parser and then
* cause all .LNG files to be colored using the LANG parser
if initial() then do
    'set parser lang langdef.kld'
    'set autocolor .lng lang'
    end
```

Several language definitions are built into KEDIT, and when KEDIT is loaded it automatically issues SET PARSER commands that use these language definitions to set

up its default parsers. See the description of the SET PARSER command for a complete list of built-in parsers. To distinguish these internal language definition files from actual disk files, KEDIT uses an asterisk as the first character of their names. For example, the command

```
SET PARSER C *C.KLD
```

tells KEDIT to use *C.KLD as the Language Definition File associated with the C parser. The asterisk in the name tells KEDIT to use the special file *C.KLD, which is built into KEDIT, and not to look for the file on disk.

Copies of all of the KLD files built into KEDIT are included in the SAMPLES subdirectory of the main KEDITW directory. For example, there is a C.KLD file that is an exact copy of the *C.KLD file that is built into KEDIT. If you modify one of these copies you should save it in a different location (normally the "KEDIT Macros" subdirectory of your Windows Documents folder, which is sometimes known as the My Documents folder) and load it by issuing a SET PARSER command referring to the modified file.

Note that whenever you issue the SET PARSER command, the KLD file that you specify is loaded into memory, even if an identical SET PARSER command has previously been issued. This makes it easy to develop and test modifications to KLD files, because if you make changes to a KLD file you can simply reissue the appropriate SET PARSER command and KEDIT will load the updated version of the file. Any files whose syntax coloring is controlled by your parser will automatically be re-colored, so you can easily see the effect of the changes you have made to the KLD file.

8.2 KLD File Format

Here is a description of the format of KEDIT Language Definition files, which usually have an extension of .KLD. The best way to get started with KLD files is to look over this description briefly, and then to examine some of the KLD files that are included in the SAMPLES directory of the main KEDITW directory.

The rules given here for KLD files are flexible enough to describe a number of popular programming languages, to handle varying syntax conventions for comments, strings, numbers, etc., and to have user-configurable lists of keywords. The goal is to handle many common language variants with a relatively small number of parameters.

KLD files are divided into sections. Each section begins with a section header, consisting of a colon in column one followed immediately by the section name. Following each section header line are one or more lines of parameter information.

To improve readability, you can insert blank lines at any point in a KLD file. Additionally, any line whose first nonblank character is an asterisk ("*") is considered a comment line and is ignored by KEDIT. For example:

```
* Sample KLD contents
:case
 ignore

:identifier
 [a-z] [a-z0-9]

:keyword
 if
 then
 else
```

The above example starts with a comment line, followed by a :CASE section with one parameter line, an :IDENTIFIER section with one parameter line, and a :KEYWORD section with three parameter lines. Parameter information is usually indented from column one, as in this example, but it does not have to be.

Here are descriptions of each kind of KLD file section:

:CASE section

The :CASE section consists of a single line with the word RESPECT or the word IGNORE. RESPECT means that the language you are describing is case-sensitive (for example, ''else'' and ''ELSE'' are not considered identical), and IGNORE means that the language is case-insensitive.

An example:

```
:CASE
 respect
```

If the :CASE section is omitted, KEDIT assumes case insensitivity. If present, the :CASE section must precede the :IDENTIFIER section.

:OPTION section

The :OPTION section consists of a single line containing special options that are needed to properly process some languages and special situations. There are three possible options:

PREPROCESSOR char

PREPROCESSOR indicates that the language supports a C-like preprocessor mechanism, and that preprocessor keywords are preceded by the specified character. For example:

```
:OPTION
 preprocessor #
```

REXX

REXX indicates that the REXX language is being described. In REXX, certain identifiers are sometimes considered keywords and are sometimes considered variables, depending on the context in which they are used, and the REXX option tells KEDIT to do the special processing that this requires.

DEEPNESTING

KEDIT normally keeps track of :MATCH section items nested to a depth of 15. You can specify DEEPNESTING in the :OPTION section of a .KLD file to cause nesting to be tracked to a depth of 255.

:IDENTIFIER section

The :IDENTIFIER section consists of a single line that specifies what characters can appear within identifiers in the language you are describing. These characters are specified in the same way as character class specifications within KEDIT regular expressions. They consist of lists, enclosed in square brackets, of valid characters and/or ranges of valid characters (with the first character in the range, a minus sign, and the last character in the range). For example,

```
:IDENTIFIER
 [a-zA-Z]
```

specifies that any set of alphabetic characters is a valid identifier.

In many languages, there are different rules for what is valid as the first character of an identifier and for what is valid in additional characters in an identifier. To handle this situation, you can include two identifier specifications: first specify what is valid as the first identifier character and then specify what is valid in the remaining characters. For example, in C programs the first character of an identifier can be any alphabetic character or can be an underscore, while the remaining characters of an identifier can be alphabetic or can be underscores, but can also be numeric digits:

```
:IDENTIFIER
 [a-zA-Z_]     [a-zA-Z0-9_]
```

In some cases (BASIC programs are the main example), the last character of an identifier can be a special character that is not valid elsewhere in an identifier. For example, in BASIC, ABC@ is a valid identifier. To handle this, you can include a third item specifying the special characters acceptable only at the end of an identifier. For example:

```
:IDENTIFIER
 [a-zA-Z]  [a-zA-Z0-9_]  [%&!#@$]
```

You can also specify in the :IDENTIFIER section that identifiers that aren't keywords should be displayed in a special color. To do this, add the word ALTERNATE followed by a number from 1 to 9 that specifies the alternate ECOLOR to be used to for these identifiers. For example:

```
:IDENTIFIER
 [a-zA-Z_]  [a-zA-Z0-9_] ALTERNATE 2
```

In the above example, ALTERNATE 2 has been added to the usual specification for C identifiers. This will cause C identifiers that are not keywords to be displayed in the color specified by ECOLOR 2. An optional second number, also in the range 0 to 9, controls the color used to display preprocessor identifiers that aren't keywords, as in this example:

```
:IDENTIFIER
 [a-zA-Z_]  [a-zA-Z0-9_] ALTERNATE 2 3
```

The :IDENTIFIER section is required if you will be using the :KEYWORD section to give a list of the keywords in your language. The :IDENTIFIER section must appear before the :KEYWORD section.

:COMMENT section

Use the :COMMENT section to describe the rules for comments in your language. Each line of the :COMMENT section describes one type of comment; since some languages have multiple methods for specifying comments, there may be multiple lines in the :COMMENT section.

Some languages have single-line comments, which are introduced by some type of comment delimiter and cannot continue for multiple lines. Some languages have comments with both a starting and an ending delimiter. This kind of comment can usually continue for multiple lines, but in some languages may be restricted to a single line.

For example, C++ allows comments that are introduced by a pair of slashes (“//”) and continue until the end of the line. C++ also allows comments that can continue for multiple lines, introduced by a slash-asterisk pair (“/*”) and terminated by an asterisk-slash pair (“*/”). The corresponding :COMMENT section would be:

```
:COMMENT
 line     //     any
 paired   /*  */ nonest
```

Line comments are described using the format

LINE *delim* ANY|FIRSTNONBLANK|COLUMN *n*

where *delim* is the comment delimiter, which is followed by an indication of when the comment delimiter takes effect:

ANY

indicates that appearance of the comment delimiter anywhere on a line (except within a quoted string) starts a comment.

indicates that appearance of the comment delimiter anywhere on a line (except within a quoted string) starts a comment.

When ANY is used an additional option, NOTAFTER [*class*], is also allowed. It indicates that an occurrence of *delim* that immediately follows one of the specified *class* of characters should not be taken to indicate the start of a comment. For example,

```
:COMMENT
 line % any notafter [*]
```

would prevent the first line below from being taken as a comment:

```
*% This one is not a comment.
%  This one is a comment.
```

FIRSTNONBLANK

indicates that the comment delimiter starts a comment only if it is the first nonblank item on a line.

COLUMN *n*

indicates that the comment delimiter starts a comment only if it appears in column *n* of a line.

Comments with both starting and ending delimiters are described using the format

PAIRED *delim1 delim2* [NEST|NONEST] [MULTIPLE|SINGLE]

where *delim1* is the delimiter that starts a comment and *delim2* is the delimiter that ends a comment.

NEST|NONEST

NEST indicates that multi-line comments can be nested inside multi-line comments, with the comments ending only when as many comment end delimiters as comment start delimiters have been encountered. NONEST is the default and indicates that comments cannot be nested, and that a comment ends as soon as the next comment end delimiter has been encountered. For example, consider

```
/*
/* here is a comment */
x = 17
*/
```

In the REXX language, which allows nested comments, "x = 17" would be considered part of a comment. In the C language, which does not allow nested comments, "x = 17" would not be considered part of a comment, and the final "*/" in the example would be invalid.

MULTIPLE

indicates that the comments can continue for multiple lines; this is the default and need not be specified.

SINGLE

indicates that, even though paired delimiters are being used, the comments must begin and end on a single line.

:HEADER section

The :HEADER section describes header lines. Header lines are used to indicate the start of a new section in certain types of files; the section headers in .KLD files are examples of header lines.

Header lines are specified in the same way as single-line comments:

LINE ***delim*** **ANY|FIRSTNONBLANK|COLUMN** ***n***

As far as KEDIT's syntax coloring is concerned, the only difference between single-line comments and headers is that comments are displayed using ECOLOR A and headers are displayed using ECOLOR G. An example of a :HEADER section that describes .KLD file section headers:

```
:HEADER
 line : column 1
```

:STRING section

Use the :STRING section to describe the types of quoted strings used in your language. Each line of the :STRING section describes one type of string; since some languages have multiple methods for specifying strings, there may be multiple lines in the :STRING section. There are three possibilities:

SINGLE

This means that your language uses strings enclosed in single quotes.

DOUBLE

This means that your language uses strings enclosed in double quotes.

DELIMITER ***c***

Use this to specify that the character *c* is the string delimiter for your language.

SINGLE, DOUBLE, and DELIMITER *c* can optionally be followed by the words BACKSLASH, VERBATIM, or MULTILINE, by the combination BACKSLASH MULTILINE, or by the combination VERBATIM MULTILINE.

BACKSLASH

This means that, as is the case in the C and C++ languages, the backslash character serves as an escape character within strings and that quote characters following a backslash do not terminate a string.

VERBATIM

This is used to handle the "verbatim" strings available in the C# language. These are special C# strings that begin with an @ character in which backslash characters don't serve as escape characters, so that you can use strings like

```
@"C:\Windows\System32"
```

instead of

```
"C:\\Windows\\System32"
```

MULTILINE

Indicates that strings need not begin and end on the same line, but can continue across end-of-line boundaries.

If the :STRING section is omitted, KEDIT's syntax coloring does not recognize any strings in your files.

Another option, NOTAFTER [*class*], which must be the last option on the line, indicates that a quote that immediately follows one of the specified *class* of characters should not be taken to indicate the start of a string. For example,

```
:STRING
single notafter [a-zA-Z]
```

would prevent the first three quote characters here from being taken as the start of a string:

```
a' + b' = c'     'only this is a string'
```

The NOTAFTER option is not valid if the VERBATIM option has been specified.

:NUMBER section

Use the :NUMBER section to indicate the format of numbers in your language. The :NUMBER section is a single line long, with the word INTEGER, DECIMAL, C, COBOL, PASCAL, REXX, or ADA.

INTEGER means that numbers consist of strings of digits.

DECIMAL means that numbers consist of strings of digits and periods.

C is used for C language numbers. These can be integers, decimal numbers, or numbers in exponential notation, like 12.4E-2. Several other languages use numeric formats that are similar to those used by C.

COBOL is used for COBOL language numbers, which consist of digits and decimal points, except that trailing decimal points are not counted as part of a number, and digits immediately followed by COBOL identifier characters (for example, 1234-TEST) are not counted as numeric.

PASCAL numbers are like C language numbers, except that they cannot start with a decimal point. Also, hexadecimal values (for example, $abcd) are treated as numeric.

REXX handles REXX language constant symbols, which include REXX numbers and symbols like 12ABC and .XYZ.

ADA numbers are like C language numbers, except that underscores are allowed within the numbers.

If the :NUMBER section is omitted, KEDIT's syntax coloring does not recognize any numbers in your files.

:LABEL section

Use the :LABEL section to define what counts as a label in your language. The label section normally consists of a single line, but can involve multiple lines if your language has multiple ways of specifying labels. The label description has the format

```
DELIMITER delim FIRSTNONBLANK|ANY|COLUMN n
```

where *delim* is the delimiter that must follow the label and FIRSTNONBLANK indicates that the label must be the first nonblank item on a line, ANY indicates that the

label can appear anywhere on a line, and "COLUMN *n*" indicates that the label must begin in column *n* of a line.

Instead of a DELIMITER line, you can specify

```
COLUMN n
```

to indicate that any non-keyword identifier beginning in the specified column should be treated as a label, with no need for a delimiter following the label.

:MATCH section

Use the :MATCH section to specify the matching characters and identifiers that indicate nested structure within your language. For example, in most languages, left and right parentheses can be nested and must match up properly in a syntactically correct program. In some languages the same is true of keywords like BEGIN and END.

KEDIT's syntax coloring facility uses the information in the :MATCH section for two purposes:

First, items at different nesting levels are colored differently, so you can easily see which items match. For example, in the line

```
if (f(x + y + z) = 17)
```

KEDIT can display the inner parentheses and the outer parentheses in different colors.

Second, when you use the CMATCH command (assigned by default to Shift+F3) to find the matching item for the text at the cursor position, KEDIT can properly match any items described in the :MATCH section. With the cursor on the first DO in the following example, Shift+F3 can move the cursor to the second END in the example:

```
if a = 5 then do
  j = 17
  do i = 1 to 10
   say i*j
  end
end
```

Each line of the :MATCH section (which can have up to 200 lines) has either two or three items. The first item specifies the identifiers or character sequences that introduce a matchable construct. The second item specifies the identifiers or character sequences that end a matchable construct. The third item is optional, and is used to specify items that always appear inside of a matchable construct.

For example,

```
:MATCH
  (   )
  {   }
  #if  #endif  #else
```

Here, three matchable constructs are specified:

The first specifies that left parentheses will be matched with corresponding right parentheses.

The second specifies that left braces will be matched with corresponding right braces.

The third specifies that, as in the C preprocessor language, #if is matched with #endif and that within an #if/#endif construct there may be an #else item that should be colored in the same way as the corresponding #if and #endif.

KEDIT actually uses the following :MATCH section in its default C language parser:

```
:MATCH
 (      )
 {      }
 #ifdef,#if,#ifndef    #endif    #else,#elif,#elseif
```

This is because any of #ifdef, #if, and #ifndef can match up with #endif, with any of #else, #elif, and #elseif allowed between them. As in this example, you can specify multiple equivalent items in a :MATCH section, separated by commas.

You can specify multi-token keywords in the :MATCH section by using a plus sign ("+") to join the components of a multi-token keyword:

```
:MATCH
 do  end+do
 if  end+if
```

Some notes on using the :MATCH section:

The current scheme for handling matched items works only for items that do not contain blanks. That is, #if and #endif pairs or BEGIN and END pairs can be matched, but WHILE and END WHILE or IF and END IF cannot be handled, since they contain blanks.

An identifier or character sequence should only appear once in the :MATCH section; any additional occurrences of the same item will have no effect. So, for example, in a language that has DO—END and BEGIN—END constructs, you should not use

```
:MATCH
 DO END
 BEGIN END
```

but should instead use

```
:MATCH
 DO,BEGIN END
```

Any identifiers included in :MATCH specifications must also appear in the :KEYWORD section, or they will be ignored.

KEDIT can display up to 8 levels of nested items in different colors. But it keeps track of nested items up to a depth of 15 — for example, up to 15 levels of nested

parentheses. You can specify DEEPNESTING in the :OPTION section of a .KLD file to cause nesting to be tracked to a depth of 255.

If the :MATCH section is omitted, KEDIT's syntax coloring facility does not recognize any matchable constructs in your files.

:KEYWORD section

Use the keyword section to specify the keywords in your language. Each line of the keyword section has the form

***keyword* [ALTERNATE *n*] [TYPE *m*]**

where *keyword* must be a valid identifier in your language. (If you specified PREPROCESSOR in the :OPTION section, you can also include preprocessor keywords, which must consist of the preprocessor character followed by a valid identifier.)

Keywords are normally colored according to the current ECOLOR D setting, and preprocessor keywords according to the current ECOLOR F setting. It is sometimes useful to specify different types of keywords that will be colored differently. To do this, you can specify

ALTERNATE *n*

following a keyword, where *n* is a number from 1 through 9. When ALTERNATE 1 is specified, ECOLOR 1 is used to color the keyword; when ALTERNATE 2 is specified, ECOLOR 2 is used, etc.

TYPE *m*

is used only when REXX has been specified in the :OPTION section, and determines what to treat as a REXX keyword, subkeyword, etc. The number *m* is determined as follows: start with *m* equal to 0, then add 1 for a REXX keyword, add 2 for a REXX subkeyword, and add 4 for a REXX keyword that takes subkeywords. For example, SAY is a keyword that does not take subkeywords, so it is TYPE 1. ARG is a REXX keyword, is also a REXX subkeyword (as in PARSE ARG), and it takes subkeywords (as in ARG UPPER), so it is TYPE 7. For further examples, see the REXX.KLD file in the SAMPLES subdirectory of the main KEDITW directory.

A sample :KEYWORD section:

```
:KEYWORD
  if
  then
  else
  do
  end
  switch
  for
  procedure  alternate 1
```

If the :KEYWORD section is omitted, KEDIT's syntax coloring facility does not recognize any keywords. If the :KEYWORD section is specified, it must be preceded by the :IDENTIFIER section.

:MARKUP section

The :MARKUP section is used with HTML and similar markup languages. It can contain a TAG line and, optionally, a REFERENCE line.

Use the TAG line to specify the character string that initiates a markup tag and the character string that terminates a markup tag.

In an HTML file, where a typical line of text might be:

```
<H1>Level 1 header</H1>
```

"<" initiates a tag, and ">" terminates it. This would be specified in the :MARKUP section as

```
:MARKUP
 TAG <  >
```

Use the REFERENCE line to specify the character string that initiates a character or entity reference and the character string that terminates it.

HTML lets you use entity references like "<" or character references like "<" to refer to special characters. These references begin with an ampersand ("&") and end with a semi-colon (";"). This would be specified in the :MARKUP section as:

```
:MARKUP
 TAG       <  >
 REFERENCE &  ;
```

The following special rules apply if your KLD file contains a :MARKUP section:

Tags are highlighted, using ECOLOR T. For example, in the line

```
<P>This is a new paragraph.
```

"<P>" would be highlighted.

Quoted strings within tags are highlighted using ECOLOR B. For example, in

```
<A HREF="film_clip.jpg">
```

the quoted string is displayed using ECOLOR B, while the rest of the tag is displayed using ECOLOR T.

Similarly, numbers within tags are highlighted using ECOLOR C.

Numbers and quoted strings that are not within markup tags are not highlighted.

Character and entity references are highlighted using ECOLOR U.

:COLUMN section

Use the :COLUMN section to specify that the parser should ignore certain columns of your file. For example, in COBOL columns 1 through 6 of a file and all columns beyond column 72 of a file are ignored by the compiler. This would be specified as

```
:COLUMN
 EXCLUDE  1  6
 EXCLUDE 73  *
```

Each line of the :COLUMN section has the word EXCLUDE followed by the starting and ending column of a range of columns that the parser is to ignore. The ending

column can be given as an asterisk to indicate that all columns through the end of the line are to be ignored.

When the syntax coloring parser processes a line of your file, it will treat the excluded columns as if they were entirely blank. By default, the excluded columns will be displayed with no special highlighting, but you can specify that any of the 9 ALTERNATE colors be used. For example,

```
:COLUMN
 EXCLUDE 1 10 ALTERNATE 2
```

would display columns 1 through 10 of your file using ECOLOR 2.

:POSTCOMPARE section

The :POSTCOMPARE section is used to color character sequences that are not handled by any of the other sections of a KLD file. For example, you might want to color operators like "+", "-", and "=", or items like ".T." and ".F.", which indicate True or False in xBase programs but are not valid identifiers.

The :POSTCOMPARE can contain CLASS lines and TEXT lines.

CLASS lines specify a set of characters that you want to have colored, using the same regular expression character class notation that is used in the :IDENTIFIER section. For example,

```
CLASS [+-=/]
```

means that "+", "-", "=", and "/" characters are to be colored. KEDIT uses ECOLOR I by default, but you can instead specify any of the nine alternate keyword colors. For example:

```
CLASS [+-=/] ALTERNATE 2
```

TEXT lines specify a string of nonblank characters that is to be colored. For example,

```
TEXT .T.
```

would color the character sequence ".T.". KEDIT uses ECOLOR D by default, but you can specify an alternate keyword color. For example:

```
TEXT .T. ALTERNATE 3
```

You can specify any number of CLASS or TEXT lines in a :POSTCOMPARE section. When applying syntax coloring to your file, the :POSTCOMPARE section is processed last. That is, KEDIT first checks for identifiers, numbers, comments, tags, etc., and checks the items in the :POSTCOMPARE section only if none of these are found.

Note that it is not useful to include valid identifiers in the :POSTCOMPARE section, since the parser checks for identifiers before :POSTCOMPARE is processed, so identifiers, even identifiers that are not listed in the :KEYWORD section, will never be matched by :POSTCOMPARE. For this reason, any identifiers that you want to color should be included in the :KEYWORD section.

Chapter 9. Error Messages and Return Codes

The first part of this chapter lists all of KEDIT's numbered error messages, with descriptions of all messages that may not be self-explanatory. The return code associated with each of the errors is also given; the second part of the chapter discusses these return codes.

In many cases, KEDIT displays additional information with these error messages. For example, when KEDIT issues the "Invalid operand" message, it usually shows you the invalid operand.

Macros that you write can use the EXTRACT /LASTMSG/ command to examine the text of messages displayed by KEDIT. Note that the message numbers, message texts, and the return codes given may change from one release of KEDIT to the next.

9.1 Error Messages

Error 1: Invalid operand (RC=5)

Error 2: Too many operands (RC=5)

Error 3: Too few operands (RC=5)

Error 4: Invalid number (RC=5)
You entered a non-numeric value for an operand that must be numeric, or you entered a number that is not valid for the command you issued.

Error 5: Numeric operand too small (RC=5)
The number that you specified as an operand for a command is less than the lowest allowable value for that operand.

Error 6: Numeric operand too large (RC=5)
The number that you specified as an operand for a command is greater than the highest allowable value for that operand.

Error 7: Invalid fileid (RC=20)

Error 8: Invalid, protected, or locked file (RC=12)
KEDIT cannot access a file you have specified, possibly because the fileid you have given is actually the name of a directory, or because the file is locked. This error can also occur if you are trying to create a file in the root directory of a drive but the root directory is full.

Error 9: File not found (RC=28)

Error 10: Path not found (RC=28)
KEDIT tried to access a file, but the path component of the fileid specified a directory that does not exist.

Error 13: Invalid drive specifier (RC=24)
You specified an invalid drive letter in a fileid. One possible cause of this is that you are not properly logged on to a network drive that you are trying to access.

Error 14: Not same device (RC=99)
You issued the RENAME command specifying fileids on two different drives; RENAME requires that both fileids specify the same drive.

Error 15: Incomplete target specification (RC=5)

Error 16: Invalid target (RC=5)

Error 17: Target not found (RC=2)

Error 18: Invalid line name (RC=5)

Error 19: Line name not defined (RC=2)

Error 20: No line names defined (RC=3)

Error 21: Invalid command (RC=-1)

Error 22: File has been changed - QQUIT to quit anyway (RC=12)
You issued the QUIT command for a file that has been modified. If you are sure that you want to abandon the changes to the file, use the QQUIT command instead.

Error 26: Too many windows defined (RC=4)
You tried to use the SET SCREEN command to display more than its limit of 16 windows.

Error 27: Invalid delimiter (RC=5)
Only special characters are valid as delimiters for strings in string targets and in the CHANGE command and related commands, but KEDIT encountered an alphabetic or numeric delimiter.

Error 28: Invalid range (RC=5)

You issued the SET RANGE command, but specified a starting line for the range that comes later in the file than the ending line.

Error 29: Synonym table full (RC=95)

You tried to define more than KEDIT's limit of command synonyms, which is 1000, or KEDIT's limit of fifteen prefix synonyms.

Error 30: Memory shortage - FSA memory full (RC=94)

A command could not execute properly because KEDIT's FSA (File Storage Area) filled up and all available system memory was already allocated to it. To free up some space, you may need to remove some files from the ring by using File Close, or you may need to close some other applications. This error message usually indicates that the system swap file, or the disk on which it is located, is nearly full.

Error 31: File already exists - use FFILE/SSAVE (RC=3)

You issued a FILE or SAVE command to write a file to disk under a new fileid, but a file with the new fileid already exists on disk. If you are sure that you want to replace the existing file, use an FFILE or SSAVE command instead.

Error 32: Invalid hexadecimal or decimal value (RC=5)

Error 33: Too many ARBCHAR characters - maximum is 50 (RC=5)

A string that you specified had more than the maximum of fifty ARBCHARs that KEDIT can handle.

Error 34: Line not found (RC=2)

You issued a FIND, FINDUP, NFIND, or NFINDUP command that was unsuccessful.

Error 35: Truncated (RC=3)

Text that a KEDIT command would have placed beyond the truncation column was truncated by KEDIT, since KEDIT commands do not operate beyond the truncation column.

Error 36: No lines changed (RC=4)

A KEDIT command unexpectedly made no changes to your file. For example, the CHANGE command may have found no occurrences of the string that it was supposed to change.

Error 37: Maximum EDITV variable length exceeded (RC=95)

You tried to set an EDITV variable to a value that is longer than the WIDTH setting, which is the maximum allowable length for an EDITV variable.

Error 38: Improper cursor position (RC=2)

The cursor was not positioned properly for the action that you tried to carry out. For example, you tried to enter text when the cursor was on the top-of-file or end-of-file line or beyond the truncation column, or you tried to use a command like SPLIT or JOIN that is invalid when the cursor is in the prefix area or beyond the truncation column.

Error 39: No remembered operand available (RC=3)

You gave no operands for a command like LOCATE or CHANGE, and no remembered operands were available because the command had not previously been issued.

Error 43: Operation would truncate box block or one-line stream block(RC=2)

KEDIT cannot carry out a move, copy, or overlay operation on a box block or a one-line stream block, because to do so would affect data beyond the truncation column, and KEDIT commands cannot operate on text beyond the truncation column.

Error 44: No marked block (RC=2)

You are trying to carry out some block operation, but there is currently no marked block.

Error 45: Marked block not in current file (RC=2)

There is a marked block in some other file in the ring, but not in the current file. Many KEDIT commands that work with marked blocks require that the block be in the current file.

Error 46: Block boundary excluded, not in range, or past truncation column (RC=2)

KEDIT cannot handle block operations if the beginning or end of the block is an excluded line and SCOPE DISPLAY is in effect, or if the beginning or end of the block is outside of the current range or is located past the truncation column.

Error 47: Operation invalid for line blocks (RC=2)

The currently marked block is a line block, and the block operation you are attempting (such as OVERLAYBOX or FILLBOX) is invalid for line blocks.

Error 48: Operation invalid for box blocks (RC=2)

The currently marked block is a box block, and the block operation you are attempting (such as COMPRESS or DUPLICATE) is invalid for box blocks.

Error 49: Operation invalid for stream blocks (RC=2)

The currently marked block is a stream block, and the block operation you are attempting is invalid for stream blocks.

Error 50: Invalid MOVE, COPY, or OVERLAYBOX location (RC=2)

The destination of a block move, copy, or overlay is inside of the marked block or is the top-of-file line.

Error 51: No preserved settings to restore (RC=3)
You issued the RESTORE command, but there are no preserved values to restore.

Error 52: Disk is write protected (RC=12)

Error 53: Command valid only when issued from a macro (RC=-3)
The ALERT, DIALOG, EXTRACT, POPUP, and READV commands, along with some forms of the EDITV command, can only be issued from within macros and not from the command line. This is because they need to examine or set the values of macro variables.

Error 54: Device not ready (RC=100)
A device that you tried to use is not ready. This is most often the result of a diskette drive with no diskette in it.

Error 55: Printer out of paper (RC=100)
KEDIT could not complete a PRINT command because the printer returned an out-of-paper error.

Error 56: I/O error on file (RC=100)
KEDIT could not complete a command involving disk input or output because it encountered an input or output error while accessing the disk.

Error 57: Disk full error (RC=13)
KEDIT could not complete a FILE, SAVE, PUT, or AUTOSAVE because the disk involved became full. You will need to free up some space on the disk before reissuing the command that could not complete.

Error 59: No lines are named (RC=3)

Error 60: Line name not found (RC=2)

Error 61: Command ignored due to error loading file (RC=6)
If your profile issues a command that causes KEDIT to load your file into memory but KEDIT runs into errors that make it impossible to load the file, KEDIT executes your profile to completion but any additional commands that your profile issues yield this error message and are not otherwise processed.

Error 62: Operation invalid for multi-line stream blocks (RC=2)
The currently marked block is a multi-line stream block, and the block operation you are attempting (such as a SHIFT or CENTER command) is invalid for multi-line stream blocks.

Error 63: Invalid cursor line or column (RC=1)
You tried to use the CURSOR SCREEN command to move the cursor to a position on the screen not valid for a cursor (for example, to the ID line or the arrow on the

command line), or you tried to use the CURSOR FILE command to move the cursor to a line or column of the file not currently displayed.

Error 64: Line not reserved (RC=4)
You tried to use the SET RESERVED command to turn off a reserved line, but the line was not reserved.

Error 65: VERSHIFT would become too large (RC=5)
You tried to use the LEFT or RIGHT commands to offset the displayed columns to the left or right by more than the maximum value that KEDIT can handle, which is equal to the setting of the WIDTH initialization option.

Error 66: Invalid match position (RC=2)
You issued the CMATCH command, but the cursor was positioned outside of the current zone columns, or on the top-of-file or end-of-file line, or on a shadow line.

Error 67: Invalid match character (RC=2)
You issued the CMATCH command, but the character in the focus column was not one of the characters that CMATCH can handle (braces, brackets, angle brackets, or parentheses).

Error 68: Matching character not found (RC=2)
The CMATCH command could not find a character that matches the character in the focus column.

Error 69: Macro line exceeds 1024 character limit (RC=97)
Lines in KEDIT macros cannot be more than 1024 characters long.

Error 70: No lines marked (RC=4)
You tried to mark a block in an empty file.

Error 71: Macro not found (RC=-1)
You issued a MACRO command, but the macro you specified could not be located.

Error 74: Prefix area contains pending commands (RC=8)
KEDIT cannot process the SORT command when there are pending prefix commands.

Error 75: Too many sort fields - maximum is 50 (RC=95)
You issued a SORT command, but specified more than the maximum of fifty sort fields.

Error 76: Fileid already in ring (RC=4)
You used SET FILEID or a similar command to change the fileid of the current file, but the new fileid is the same as the fileid of some other file in the ring.

Error 77: Command or feature unavailable in KEDIT demo version (RC=5)
You are using the demo version of KEDIT, and the command that you issued or feature that you attempted to use is available only in the production version of KEDIT.

Error 78: HIT queue full (RC=95)
The HIT queue is full, and no more macros can be queued up by the HIT command until KEDIT processes some of the macros already in the queue. The HIT queue can have up to twelve entries.

Error 79: Invalid .KML file header line (RC=97)
KEDIT tried to load a .KML file, but the file began with something other than a comment or a .KML-format macro definition.

Error 80: Operation interrupted by Ctrl+Break or Alt+Ctrl+Shift (RC=96)
You pressed Ctrl+Break or Alt+Ctrl+Shift while a command was active.

Error 84: Invalid macro name (RC=97)
The length of the name of a macro exceeds the allowable maximum of 40.

Error 85: Files in ring would exceed maximum of 500 (RC=95)
You tried to add a new file to the ring, but the maximum allowable number of files is already in the ring.

Error 86: Unexpected null character encountered (RC=5)
Null characters (character code 0) cannot appear in the text of macros, and are valid only in certain contexts within command strings that KEDIT processes.

Error 87: Cursor line not in scope (RC=2)
Many KEDIT commands are invalid if issued from a macro when SCOPE DISPLAY is in effect and the cursor is on a shadow line.

Error 88: Macro has multi-line definition (RC=95)
You issued the MODIFY MACRO command for a macro with a multi-line definition, but MODIFY MACRO requires that the macro to be modified have a one-line definition. You can instead use the QUERY MACRO command to see the definition of the macro, or you can use the MACROS command to have the macro definition put into a file.

Error 89: Invalid column target (RC=5)

Error 90: Internal KEDIT stack full (RC=95)
A command could not execute properly because KEDIT ran out of room on its internal stack. This can happen if you are nesting your macros more deeply than KEDIT can handle. (That is, one macro calls another, which calls another, etc.)

Error 91: Memory shortage - ISA memory full (RC=94)

A command could not execute properly because KEDIT ran short of memory and all available system memory was already allocated to it. To free up some space, you may need to remove some files from the ring by using File Close, or you may need to close some other applications. This error message usually indicates that the system swap file, or the disk on which it is located, is nearly full.

Error 92: Command too long (RC=95)

The maximum allowable length for a KEDIT command is equal to the value of the WIDTH initialization option; a macro tried to issue a command that is longer than this.

Error 93: Invalid variable reference (RC=98)

An instruction of a KEXX macro that used a variable reference (the name of a variable in parentheses) did not have a right parenthesis as the first token after the variable name.

Error 94: Arithmetic overflow/underflow (RC=98)

An arithmetic expression in a KEXX macro yielded an extremely large or small result. This error is most often caused by an attempt to divide by zero.

Error 95: Invalid data on end of clause (RC=98)

Unexpected text was found at the end of a clause in a KEXX macro. You may have forgotten a semicolon.

Error 96: Invalid expression (RC=98)

An invalid expression was encountered in a KEXX macro. One possible cause of this error is a macro that issues a KEDIT command involving special characters, but does not have quotes around the command.

Error 97: Invalid hexadecimal or binary string (RC=98)

An invalid hexadecimal or binary string was encountered in a KEXX macro. Examples of valid hexadecimal strings are '124f'X and 'FE 03'X. Examples of valid binary strings are '10101111'b and '1100 0011'B.

This error is most often encountered when you aren't even intending to use a hexadecimal or binary string, but are instead intending to concatenate a normal character string with the value of a variable named B or X. For example,

```
val = 'something'b
```

is interpreted by KEXX as an invalid attempt to specify a binary string and not, as you probably intended, an attempt to concatenate the string "something" and the value of the variable B.

Unlike all other variables, the variables B and X cannot be specified immediately after a string to cause implicit concatenation; with B and X, you instead need to use explicit concatenation, for example

```
val = 'something' || b
```

Error 98: Invalid variable name (RC=98)
An invalid variable name was encountered in a KEXX macro.

Error 99: Invalid whole number (RC=98)
In certain contexts, KEXX requires whole numbers (numbers that can be expressed within the current NUMERIC DIGITS setting without using a decimal point or exponential notation). For example, 2**3.1 is invalid, because KEXX can raise numbers only to whole-number powers, and DO loop control variables must be whole numbers.

Error 100: Control stack full (RC=98)
DO, SELECT, and IF—THEN—ELSE constructs are being nested too deeply within the currently-active KEXX macros.

Error 101: Logical value not zero or one (RC=98)
The only acceptable operands for logical operators such as & and | in KEXX macros are 0 and 1.

Error 102: Not numeric value (RC=98)
An operand of an arithmetic operator in a KEXX expression could not be converted to a numeric value.

Error 103: Symbol expected (RC=98)
A clause with invalid syntax has been encountered in a KEXX macro.

Error 104: THEN expected (RC=98)
An IF instruction in a KEXX macro was not followed by a THEN clause.

Error 105: Invalid expression result (RC=98)
An expression result is not valid for the context in which the expression was evaluated. For example, the value specified for NUMERIC DIGITS or NUMERIC FUZZ was non-numeric or not in the allowable range. (The maximum allowable value for NUMERIC DIGITS is 1000, and NUMERIC DIGITS must always be larger than NUMERIC FUZZ).

Error 106: Unbalanced parentheses (RC=98)
An expression with unbalanced parentheses has been encountered in a KEXX macro.

Error 107: Unexpected or unmatched END (RC=98)
A KEXX macro has an END that does not terminate a DO instruction, or a DO instruction not terminated by an END.

Error 108: Unexpected THEN or ELSE (RC=98)
THEN or ELSE was encountered in a KEXX macro outside of the context of an IF clause.

Error 109: Unmatched ‘‘/*’’ or quote (RC=98)
An unmatched comment, single quote, or double quote was encountered in a KEXX macro. You may have tried to use a comment or quoted string spanning more than one line of a macro, which KEXX does not allow.

Error 110: Maximum KEXX expression result length exceeded (RC=98)
The length of an expression result, or of an intermediate result used in the calculation of an expression result, exceeded KEXX’s limit. The limit is equal to the current WIDTH value plus 100.

Error 111: Incorrect call to routine (RC=98)
A function call in a KEXX macro specified too many or too few arguments, or one of the arguments specified was invalid for the function involved.

Error 112: Routine not found (RC=98)
A function or subroutine called from a KEXX macro could not be located. KEXX looks for internal routines, built-in functions, implied EXTRACT functions, Boolean functions, and external routines.

Error 113: Macro interrupted by Ctrl+Break or Alt+Ctrl+Shift (RC=98)
Ctrl+Break or Alt+Ctrl+Shift interrupted execution of a KEXX macro.

Error 114: Interpretation error (RC=98)
An internal processing error was encountered during execution of a KEXX macro.

Error 115: Invalid LEAVE or ITERATE (RC=98)
A KEXX macro contains a LEAVE or ITERATE instruction outside of an iterative DO loop.

Error 116: Invalid DO syntax (RC=98)
A KEXX macro contains an invalid DO instruction, for example one which has two TO expressions.

Error 118: Unsupported REXX feature (RC=98)
A REXX feature that is not implemented in KEXX was used in a KEXX macro.

Error 119: Invalid TRACE request (RC=98)

Error 120: Invalid sub-keyword found (RC=98)
A KEXX macro used an invalid sub-keyword within a keyword instruction. For example, a PARSE instruction may have incorrectly specified the origin of the string to be parsed.

Error 121: Invalid PARSE template (RC=98)

Error 122: Function did not return result (RC=98)
A KEXX macro invoked an internal routine as a function, but the function did not return a result.

Error 123: Invalid label (RC=98)
Labels in KEXX macros that do not appear as the first token on a line are invalid.

Error 124: Unexpected PROCEDURE (RC=98)
A PROCEDURE instruction was encountered that did not follow a label, as is required in KEXX macros.

Error 125: Uninitialized variable; NOVALUE ON (RC=98)
NOVALUE ON is in effect, and your macro attempted to use the value of a variable to which you had not yet assigned a value. You might have spelled the variable name wrong, or might have mistakenly used a variable name instead of a quoted string.

Error 126: Unexpected WHEN or OTHERWISE (RC=98)
A WHEN or OTHERWISE instruction was encountered outside of a SELECT construct, or improperly embedded in a DO—END construct within a SELECT construct.

Error 127: WHEN or OTHERWISE expected (RC=98)
A SELECT construct was encountered that does not contain any WHEN clauses, or that contains clauses in a context where only WHEN and OTHERWISE clauses are valid. A common cause of this error is forgetting to use DO—END around multiple clauses associated with a WHEN—THEN construct.

Error 128: Incomplete DO/SELECT/IF (RC=98)
A DO or SELECT instruction without the matching END, or an IF instruction without a THEN clause, has been encountered.

Error 129: Machine resources exhausted (RC=98)
KEXX has run out of memory while trying to process a macro.

Error 130: Label not found (RC=98)
The label specified by a SIGNAL instruction, or required to handle a condition like SIGNAL ON NOVALUE, cannot be found. Note that labels in KEXX macros can only appear as the first token on a line.

Error 131: Error in called routine (RC=98)
A KEXX macro called an external routine, but there was an error loading the external routine or the external routine terminated with an error.

Error 132: Symbol or string expected (RC=98)
Within a KEXX macro, in a context where only a string or symbol is valid, either the end of a clause was reached or an invalid token was encountered.

Error 133: INTERPRET string exceeds 1024-character limit (RC=98)
You tried to use the INTERPRET instruction to interpret a string that was longer than 1024 characters.

Error 134: NUMERIC DIGITS must be > FUZZ and <= 1000 (RC = 98)
The maximum allowable value for NUMERIC DIGITS is 1000, and NUMERIC-DIGITS must always be larger than NUMERIC FUZZ

Error 135: Memory shortage - cursor field reset
You typed some text into the command line, the prefix area, or a line of your file, but when KEDIT tried to process the change, it did not have enough available memory to do so. The contents of the field you changed are reset to what they were before your text was entered.

Error 136: The file is already locked (RC=5)
You tried to use the LOCK command for a file that is already locked.

Error 138: Unexpected file timestamp change - use FFILE/SSAVE (RC=3)
You issued a FILE or SAVE command, but TIMECHECK ON is in effect and the file's timestamp has changed since you began editing the file or last saved it. This occurs when some other program, running either on your own computer or on another computer on your network, changes the file while you are editing it. If you are sure that you still want to replace the file, use an FFILE or SSAVE command instead.

Error 141: Read-only file (RC=12)
You tried to write to a file that is marked on disk as read-only.

Error 143: Network access error (RC=100)
A system function used by KEDIT failed due to a network-related error.

Error 144: Nothing to UNDO (RC=4)
You issued the UNDO command but there was nothing to undo. You may not have made any changes to your file, you may have already used UNDO to undo all changes to your file, UNDOING OFF may be in effect, or KEDIT may have discarded undo information due to memory limitations.

Error 145: Nothing to REDO (RC=4)
You issued the REDO command but there was nothing to redo. The REDO command is valid only after an UNDO command, and only when no undoable commands have been issued since the corresponding UNDO command.

Error 146: Out of memory - UNDO or REDO incompletely processed (RC=94)
KEDIT ran out of memory in the middle of processing an UNDO or REDO command. Some portion of the undo or redo may have been carried out.

Error 147: Out of memory - couldn't process UNDO or REDO (RC=4)
KEDIT ran out of memory in the middle of processing an UNDO or REDO command. The undo or redo was not carried out.

Error 148: Demo version save operations limited to *n*-line files (RC=5)
You are using the demo version of KEDIT and have attempted to save a file with more lines in it than the demo version allows. Note that the production version of KEDIT does not limit the size of the files that you can save.

Error 150: Invalid KEDIT module (RC=99)

Error 151: System error code *n* (RC=99)
An operating system function called by KEDIT unexpectedly returned with error code *n*.

Error 152: Macro not currently in storage (RC=3)
You issued a command like MACROS or QUERY MACRO and specified a macro name, but there is no currently defined in-memory macro with that name.

Error 154: JOIN would cause truncation (RC=4)
You are using the JOIN command to join two lines together, but the resulting text would extend beyond the truncation column. To avoid truncating your data, KEDIT cancels the join operation.

Error 155: JOIN would involve excluded line (RC=4)
You are using the JOIN command to join two lines together, but the second of the two lines is an excluded line that KEDIT cannot operate on.

Error 156: Command line unavailable (RC=1)
You tried to move the cursor to the command line but SET CMDLINE OFF is in effect.

Error 161: Out of system resources (RC=94)
KEDIT cannot complete a command because the supply of some system resource has been exhausted. Windows may be out of memory, or the data areas that Windows uses to keep track of fonts, windows, bitmaps, etc. may be full. To free up resources, you may need to remove some files from the ring by using File Close, or you may need to close some other applications.

Error 164: Cannot merge overlapping areas (RC=1)
You tried to use the MERGE command to merge together two groups of lines, but the MERGE command failed because the two groups of lines overlapped.

Error 165: Invalid use of colon in regular expression (RC=5)
You used a colon in a regular expression, but did not follow it with the letter that identifies one of the predefined regular expressions. If you do not want to use a predefined expression, but instead want the colon to be taken literally, precede it with the regular expression escape character, which is a backslash.

Error 166: Unknown predefined string specified in regular expression (RC=5)
You used a colon in a regular expression, but the character that follows it does not identify one of the predefined regular expressions. If you do not want to use a predefined expression, but instead want the colon to be taken literally, precede it with the regular expression escape character, which is a backslash.

Error 167: Invalid escape sequence in regular expression (RC=5)
Your regular expression used a backslash, which introduces an escape sequence, but either the escape sequence was invalid (for example, \x followed by an invalid hexadecimal value), or the escape sequence was missing (the backslash was the last character in the regular expression). If you are using a backslash in a regular expression and want to have it taken literally, you must actually use two backslashes.

Error 168: Unbalanced parenthesis in regular expression (RC=5)
Your regular expression contains an unbalanced parenthesis. If you are using a parenthesis in a regular expression and want to have it taken literally, you must precede it with the regular expression escape character, which is a backslash.

Error 169: Unbalanced brace in regular expression (RC=5)
Your regular expression contains an unbalanced brace character. If you are using a brace in a regular expression and want to have it taken literally, you must precede it with the regular expression escape character, which is a backslash.

Error 170: Power specified in regular expression exceeds maximum of 255 (RC=5)
You used the power operator (a caret, ''^'') in a regular expression, but the power you specified exceeds the limit of 255.

Error 171: Infinite closure loop in regular expression (RC=5)
You have specified a regular expression that can never be successfully matched. For example, ''(a*)@'' matches as many occurrences as possible of zero or more ''a''s, which amounts to matching zero ''a''s an infinite number of times.

Error 172: Invalid class specification in regular expression (RC=5)
Your regular expression contains an invalid character class specification. For example, the class may have an opening left bracket but no closing right bracket, or it may have an incomplete range specification. If you are using a bracket in a regular expression and want to have it taken literally, you must precede it with the regular expression escape character, which is a backslash.

Error 173: Invalid regular expression (RC=5)

Error 174: Too many tags in regular expression (RC=5)
Your regular expression contains more than the limit of nine tags, which begin with left braces (''{'') and end with right braces (''}''). If you are using a brace in a regular expression and want to have it taken literally, you must precede it with the regular expression escape character, which is a backslash.

Error 175: Illegal tag specification in regular expression (RC=5)
Your regular expression contains an invalid tag specification. If you are using a brace in a regular expression and want to have it taken literally, you must precede it with the regular expression escape character, which is a backslash.

Error 176: Incorrect tag reference in regular expression (RC=5)
You used an ampersand (“&”) in a regular expression. An ampersand signals a reference to a tagged expression, and must be followed by a digit indicating which tagged expression is involved, but the digit was missing in your regular expression. If you are using an ampersand in a regular expression and want to have it taken literally, you must precede it with the regular expression escape character, which is a backslash.

Error 179: Invalid use of cursor position operator (RC=5)
You used the cursor position operator (“\c”) in an invalid way within a regular expression. For example, it was immediately preceded by the not operator (“~”), or immediately followed by a closure operator.

Error 180: Unsupported closure or power subexpression in regular expression (RC=5)
KEDIT’s regular expression processor cannot handle expressions involving closure or power operators applied to closure or power operators. For example, KEDIT cannot handle this regular expression: “(x+)+”.

Error 186: Error accessing clipboard (RC=99)
KEDIT got an unexpected error when it attempted to access the system clipboard.

Error 187: No text in clipboard (RC=2)
An attempt to Paste text into KEDIT from the clipboard failed because there was no text in the clipboard.

Error 188: Command not available in this version of KEDIT (RC=-1)
You have issued a command that is valid in some versions of KEDIT but is not supported in this version of KEDIT.

Error 189: KEDIT Help file not found: KEDITW.CHM (RC=28)
Your tried to get access KEDIT's Help system but KEDIT's Help file, KEDITW.CHM, could not be located. This file is normally put into KEDIT's program directory by the KEDIT installation program. KEDIT looks for this file in the directory from which the KEDIT for Windows program file, KEDITW32.EXE, was loaded, and in C:\Program Files\KEDITW.

Error 190: String contains all possible delimiters (RC=5)
Some of KEDIT’s dialog boxes, such as Edit Find and Edit Replace, work by internally issuing commands like LOCATE and CHANGE to search for strings. The strings passed to LOCATE and CHANGE must be delimited, and KEDIT issues this error message in the very rare situation where no valid delimiters are available because the strings it is working with contain every valid delimiter character.

Error 191: Drive/directory specifiers not allowed in OPENFILTER (RC=5)
The fileids passed to SET OPENFILTER must consist of file names and extensions, possibly including wildcard characters, and they cannot include drive or directory specifiers.

Error 192: Button not defined via SET TOOLBUTTON (RC=5)
You have issued a SET TOOLSET command that specifies a toolbar button that has not yet been defined via SET TOOLBUTTON.

Error 193: Error creating bitmap (RC=99)
You have issued a SET TOOLBUTTON command that specifies a bitmap file on disk, but KEDIT encountered an error converting the bitmap into Windows' internal bitmap format. This most often occurs when the data in the bitmap file is not in the proper Windows bitmap format.

Error 194: Error loading bitmap (RC=99)
You have issued a SET TOOLBUTTON command that specifies a bitmap file on disk, but KEDIT encountered an error accessing the bitmap file or loading it into memory. This most often occurs when the data in the bitmap file is not in the proper format.

Error 195: Bitmap too wide or tall (RC=95)
You have issued a SET TOOLBUTTON command that specifies a bitmap file on disk, but the bitmap involved exceeds KEDIT's limits of 100 pixels high by 100 pixels wide.

Error 196: Invalid placement of Quick Find toolbar item (RC=5)
You have issued a SET TOOLSET command that refers incorrectly to the Quick Find toolbar item. Quick Find can only go on the top toolbar, and not on the bottom toolbar, and can only appear once on a toolbar, and not multiple times.

Error 197: TOOLBUTTON definition exceeds 400 character limit (RC=95)
The total length of the operands for the SET TOOLBUTTON command cannot exceed 400 characters.

Error 198: Multi-line clipboard text cannot be pasted to command line (RC=2)
You have attempted to Paste text from the clipboard to the command line. KEDIT allows this when the clipboard contains a single line of text, but not when the clipboard contains multiple lines of text. Multiple lines of text can be pasted into your file, but not to the command line. To Paste text into your file, you must position the cursor in the file area at the location where you want the pasted text to be inserted.

Error 199: Parser not defined: *name* (RC=5)
The parser referred to in a SET AUTOCOLOR or SET COLORING command has not been previously defined via a SET PARSER command.

Error 200: Too many items in target - maximum is 50 (RC=5)
You are trying to use and ("&") and or ("|") to combine more than the limit of fifty targets together.

Error 201: SET SCREEN unavailable in One-File-Per-Window mode; try Window Arrange (RC=2)

You are in One-File-Per-Window mode, with the default of OFPW ON in effect, and you tried to use the SET SCREEN command. SET SCREEN is supported in KEDIT for Windows for compatibility with text mode KEDIT, and is available only when OFPW OFF is in effect. With One-File-Per-Window mode, KEDIT's window handling is much more like that of other Windows applications, and because of this we recommend One-File-Per-Window mode for most users, even though it precludes use of the SET SCREEN command. One alternative for arranging your document windows is to use the Window Arrange dialog box.

For more information, see the descriptions of SET OFPW and of SET SCREEN.

Error 202: Operand invalid with INTERFACE CLASSIC (RC=5)

INTERFACE CLASSIC is in effect, and you used a command operand that is only valid with INTERFACE CUA. For example, since selections are available only with INTERFACE CUA, the SELECTION operand of the MARK command is invalid with INTERFACE CLASSIC.

Error 204: FILE/SAVE of UNTITLED file requires fileid (RC=5)

You cannot save an UNTITLED file (edited via File New or by starting KEDIT with no fileid specified on the command line) without assigning it a permanent name. Either use the SET FILEID command to assign a name, or specify a name for the file on your FILE or SAVE command.

Error 205: DOS command is too long (RC=99)

The length of a DOS command that you are trying to execute exceeds the limit. The command line that KEDIT passes to DOS can be up to 4096 characters long, but because of certain parameters that KEDIT itself must include on this command line, the practical limit for the length of your DOS command is typically closer to 4000 characters.

Error 206: REXX macro invoked; this version of KEDIT supports only KEXX (RC=99)

You tried to run a REXX macro but KEDIT for Windows only supports KEXX macros. A REXX macro is any KEDIT macro that begins with a REXX-style ("/* ... */") comment; KEXX macros cannot begin with REXX-style comments.

Error 207: REXX not available; this version of KEDIT supports only KEXX (RC=99)

You tried to use the QUERY REXXVERSION command to determine the version of REXX installed on your system, but KEDIT for Windows only supports KEXX macros.

Error 208: TOOLSET definition exceeds 400 character or 50 item limit (RC=95)

The total length of the operands for the SET TOOLSET command cannot exceed 400 characters, and the total number of items in a toolset, including periods used to indicate spacing, cannot exceed 50.

Error 209: Error updating registry (RC=99)
KEDIT encountered an unexpected error while updating the Windows registry during the processing of a REGUTIL command or of Options Save Settings.

Error 210: *Option* **is not one of the SET options kept in the registry** (RC=5)
You issued the REGUTIL SAVE SETTING command, but specified a SET option that KEDIT does not maintain in the Windows registry. If you want to put a new value for the option into effect for every KEDIT session, you can place the SET command involved in your profile.

Error 211: Unable to create the special file *fileid* (RC=4)
You issued a REGUTIL command that adds a temporary file to the ring to hold information from KEDIT's section of the Windows registry, but KEDIT encountered an unexpected error when it tried to create the file.

Error 212: SET *option* **unavailable in this version of KEDIT** (RC=5)
You issued a SET command for an option that is valid in earlier text mode versions of KEDIT but that is not available in KEDIT for Windows.

Error 213: QUERY *option* **unavailable in this version of KEDIT** (RC=5)
You issued a QUERY command for an option that is valid in earlier text mode versions of KEDIT but that is not available in KEDIT for Windows.

Error 214: MODIFY *option* **unavailable in this version of KEDIT** (RC=5)
You issued a MODIFY command for an option that is valid in earlier text mode versions of KEDIT but that is not available in KEDIT for Windows.

Error 215: WINEXEC failed with system error code *n* (RC=99)
You issued the WINEXEC command, but when KEDIT used the WinExec function to pass your command to Windows, system error code *n* was returned. The most common error codes are 2, which means that Windows could not find the file that you specified, and 3, which means that you specified a directory path for the file, and Windows could not find that directory.

Error 216: .KLD error - *info* (RC=97)
An error in a KEDIT Language Definition file was encountered while processing a SET PARSER command.

Error 218: Built-in file not found: *fileid* (RC=28)
A SET PARSER command used an asterisk in front of a fileid, but the KEDIT Language Definition file referred to is not built into KEDIT.

Error 219: NULL parser cannot be redefined (RC=5)
You tried to use a SET PARSER command to redefine the NULL parser, but the NULL parser is a special do-nothing parser that cannot be redefined.

Error 221: Bad network name or path: *fileid* (RC=28)
You used a UNC (Universal Naming Convention) name that is invalid or refers to a network resource that is not accessible to your machine.

Error 222: Internal control stack full (RC=95)
MACRO commands or calls by macros to external routines are being nested too deeply.

Error 223: Invalid DEFBUTTON value (RC=5)
You issued a DIALOG or ALERT command specifying a DEFBUTTON value that exceeds the number of buttons in the dialog box.

Error 224: Another process has exclusive access to *fileid* (RC=12)
Another program has exclusive access to the specified file. The other program could be another application running on your computer, or it might be running on another computer on your network.

Error 225: File not locked (RC=5)
You tried to use the UNLOCK command for a file that is not locked.

Error 227: QUICKFIND longer than 120-character limit (RC=9)
You issued the SET QUICKFIND command, but the string that you specified is longer than the maximum of 120 characters.

Error 228: Invalid network password specified (RC=100)
KEDIT is unable to access a file because a password is required to access the computer on which the file resides, but Windows does not have the correct password.

Error 229: Error creating printer device context (RC = 99)
KEDIT got an unexpected error while trying to communicate with the Windows printing subsystem.

Error 230: Printer margins exceed printable area (RC = 99)
The printer margins (controlled via the Margins... button in the File Print dialog box) that are in effect, or that you are trying to put into effect, are too large for the currently active Windows printer and paper size.

Error 231: File already exists: *fileid* (RC = 99)
You are trying to rename a file, but a file with the same name as the new filename involved already exists.

Error 232: PATH or MACROPATH setting exceeds 1000-character limit (RC = 95)
The values specified with SET PATH and SET MACROPATH have a 1000-character limit.

Error 233: *Command* **cannot have fewer ARCHAR chars on left than right** (RC = 5)
CHANGE and SCHANGE commands that use ARBCHAR characters must use at least as many ARBCHAR characters in the left string as in the right string.

Error 234: Manual not found in KEDIT program directory: *fileid* (RC = 28)
You tried to view the KEDIT User's Guide or Reference Manual, but the PDF files involved could not be found in the KEDIT for Windows program directory.

Error 235: Error accessing PDF viewer for *fileid* (RC = 28)
You tried to view the KEDIT User's Guide or Reference Manual, which are in PDF format, but either there is no PDF viewer, such as Adobe Reader, installed on your system, or KEDIT was not able to access it.

9.2 Return Codes

Every KEDIT command sets a return code that you can test in a macro to obtain information about the success or failure of the command. The special macro variable *RC* is automatically set equal to the value of the return code after a command completes. (KEDIT keeps separate track of the return code of the last command issued from the command line, which can be accessed via EXTRACT /LASTRC/.)

KEDIT commands set a return code of 0 if they execute successfully, and set a nonzero return code if an error is encountered. There is, however, one important exception to this. A return code of 1 from any command except the CURSOR command is not an indication of an error, but an indication that the command encountered the top-of-file or end-of-file line during its processing or, for commands like LOCATE or FORWARD, left you positioned at the top-of-file or end-of-file.

Macros can use the EXIT instruction to set their own return codes; the normal return code from macros is 0.

Here is a summary of the return codes given with KEDIT commands. Where possible, these return codes are modeled after those that XEDIT would return in a similar situation. Unfortunately, XEDIT does not follow a completely consistent pattern, so the following list is only approximate. The specific return codes set in connection with each KEDIT error message are given in the preceding section.

0 Command completed successfully.

1 Top-of-file or end-of-file encountered (not an error).

2 String or target not found.

3 Lines truncated by command.

4 A command unexpectedly had no effect. (For example, a CHANGE command found no occurrences of a string to change.)

5 An invalid command operand was encountered.

6 A command was issued from a profile after an error was encountered loading the file involved, or a command is valid only when there are files in the ring but was issued from a macro when the ring was empty. Commands that get a return code of 6 are otherwise ignored, and in most cases no error message is given.

8 There were pending prefix commands when you issued a command that cannot operate when prefix commands are pending.

12 The operating system denied access to some file, or you tried to QUIT from a modified file.

13 A disk full situation occurred.

20 Invalid fileid.

24 Invalid drive specifier.

28 File or path not found.

94 KEDIT did not have enough memory to process a command.

95 Some internal KEDIT limit has been exceeded.

96 Execution of a command was interrupted by Ctrl+Break or Alt+Ctrl+Shift.

97 KEDIT was unable to load or define a requested macro.

98 A macro language-related error occurred during the processing of a macro.

99 KEDIT encountered an error in accessing some system facility.

100 KEDIT encountered an Input/Output error.

-1 You asked KEDIT to execute a command or macro that could not be located.

-3 A command was issued from an environment that is inappropriate for that command. (For example, the EXTRACT command was issued from the command line.)

Index

C

D

E

F

J

K

N

O

P

Q

R

S

T

U

V

W

X

Z