BRIEF®
FOR DOS AND OS/2®

USER'S GUIDE

BORLAND

BRIEF® for DOS and OS/2

Version 3.1

User's Guide

BORLAND INTERNATIONAL, INC. 1800 GREEN HILLS ROAD
P.O. BOX 660001, SCOTTS VALLEY, CA 95067-0001

PRINTED IN THE USA.
10 9 8 7 6 5 4 3 2 1

R1

C O N T E N T S

I N T R O D U C T I O N

BRIEF is a text editor, running under either DOS or OS/2. You can use it to enter and modify programs and data in standard ASCII text form.

BRIEF focuses on the three most important areas for programming: speed, power, and safety. It provides an environment where you can edit an unlimited number of files without any loss of speed, display these files in windows that you create and size to your own choosing, and manipulate them with commands you design.

The tutorial in Chapter 2 introduces commands and concepts provided with BRIEF. As you work through it, concentrate on the features presented, but don't worry if you find that you might prefer a different keystroke for certain commands. BRIEF understands programmers and is built on the concept of programmability; you can, by changing keystroke assignments and modifying commands, make the editor fit your preferences.

BRIEF features

BRIEF can edit any standard ASCII program or data file. Key features include

- Full-screen editing and cursor movement
- Online context-sensitive help
- Editing multiple files
- Ability to undo most commands
- Copy and move block commands to transfer text within and between files
- Multiple windows viewing different files or different parts of the same file
- Multiple keystroke macros

- Completely reconfigurable keyboard
- Flexible macro language
- Powerful, full-screen, source-level macro debugger
- Extensive search and translate capabilities

Many more available features are discussed throughout this manual.

We've also added a number of new features to the latest release of BRIEF (version 3.1), the Professional Programmer's Editor. This helps you quickly get acquainted with what's new. A capsule description of each feature is provided.

This section provides information about

- Mouse support
- Redo command
- EMS support
- HPFS support
- New macro language functions
- Popup menu added
- Check boxes, radio buttons, and push buttons added
- PWB macro included
- New key assignments
- Compiler support
- Longer status line messages
- New Tech Support information

Mouse support

BRIEF supports 2- or 3-button Microsoft-compatible mice. Mouse button assignments are similar to Windows 3.0 button assignments. The mouse is also accessible from the macro language by using new macro language functions.

If you are updating to BRIEF 3.1 from a previous version of BRIEF, mouse support requires changes to the _init macro in your initials macro. These changes are done automatically when you run SETUP. Simply respond to the SETUP prompt that asks to update your initials macro: either allow the update or write the changes to another file and replace the _init macro in your old initials macro with the _init macro in the new initials macro.

Enabling mouse support outside SETUP

Upgrade users who do not want to run SETUP can enable mouse support by adding four of the new mouse functions to the STARTUP macro:

- **set_mouse_type (type, [btn1_loc], [reset_type], [double_click])** —Sets the type of mouse BRIEF expects:

`type`	0 (no mouse), 2 (two-button mouse), or 3 (three-button mouse)
`btn1_loc`	0 (if button 1 is left-most) or 1 (if button 1 is right-most)
`reset_type`	0 (if soft reset) or 1 (if hard reset)
`double_click`	A number 1 (shortest interval between clicks of a double click) through 9 (longest interval between clicks)

- **set_ctrl_state (control_id, state)**—Sets the state of mouse scroll bars and mouse buttons:

`control_id`	Identifies the control being set; value can be CLOSE_BTN, ZOOM_BTN, VERT_SCROLL, and HORZ_SCROLL, which are defined in win_ctrl.h.
`state`	Specifies the desired state; values can be ENABLE_CTRL, DISABLE_CTRL, HIDE_CTRL, and SHOW_CTRL, which are defined in win_ctrl.h. HIDE_CTRL and SHOW_CTRL should be reserved for macros.

- **set_mouse_action (name)**—Defines the macro to execute on all mouse actions. The default mouse action macro is called _mouse_action. A sample replacement mouse action macro can be found in the mouse.cb file in the \MACROS directory; it provides examples that show sub-classing and super-classing of the default mouse handler (see "Sample mouse handler" below).
- **set_btn2_action ([action])**—Sets the action setting of mouse button 2. Set to 0 for Quick-Edit; set to 1 for Quick-Menu (see "Popup menu added" below).

Mouse button assignments

Mouse button assignments for the various macro packages are listed in the READ.ME file on your BRIEF disk.

Sample mouse handler

A sample mouse event handler is available in mouse.cb. Examples of using the mouse inside macros can be found in

- buffers.cb
- dialog.cb
- dlg_list.cb
- dlg_menu.cb
- dlg_mous.cb
- errorfix.cb
- help.cb
- keys.cb
- prompt.cb
- pvcs.cb
- search.cb
- tlib.cb

Mouse support necessitated the addition of Close and Zoom buttons and scroll bars. If these controls are enabled and your macros create popup windows, the controls will appear on the windows. The controls may be hidden using **set_ctrl_state**. To use the controls with the mouse, your macro will require a mouse event handler. The macros listed above contain mouse event handlers. Also mouse.cb is a sample event handler. Pushing and popping keyboards also affect the current mouse event handler.

Redo command

A Redo command is now available. Redo allows you to redo previously undone commands until you edit the buffer. Redo is assigned to *Ctrl+u*, which was previously assigned to Scroll buffer up. Scroll buffer up is now assigned to *Ctrl+e*.

EMS support

If EMS memory is available, by default, BRIEF buffers files and macros there. EMS memory can be turned off by setting an environment variable of BEMS equal to zero.

HPFS support

BRIEF now supports the new High Performance File System (HPFS) available with OS/2 Version 1.2. HPFS support includes the capability to

- name a file and directory using up to 254 characters (259 with the path),
- use uppercase, lowercase, or mixed case when naming files, and
- assign extended attributes to a file.

New macro language functions

Each of the following functions is discussed in detail in the *Macro Language Guide.*

close_window	Collapses the current window.
copy_ea_info	Copies extended attribute information to a file (OS/2 only).
crunch_filename	Truncates a file name.
hpfs_to_fat	Converts an HPFS file name to a FAT file name (OS/2 only).
inq_btn2_action	Returns the default action setting of mouse button 2.
inq_ctrl_state	Returns the current state of mouse controls.
inq_mouse_action	Returns the name of the current mouse handler.
parse_filename	Parses a file name into its component parts.
read_ea	Reads extended attribute information for a file (OS/2 only).
redo	
set_btn2_action	
set_ctrl_state	
set_ea	Sets extended attribute information for a file (OS/2 only).
set_mouse_action	
set_mouse_type	

Popup menu added

BRIEF supports a popup menu that is actuated using mouse button 2. When actuated, the popup menu appears with its top left corner at the current mouse position. During SETUP, you can choose how you want to use mouse button 2 to display the popup menu, as well as to perform other edits such as cut, copy, and paste. When SETUP displays the `Default button 2 action` prompt, select from one of the two options shown here.

Popup menu—If you select this option, mouse button 2 works as follows:

Taking this action...	Causes this result...
Button 2 click	Display popup menu
Button 2 double click	Display popup menu
Ctrl Button 2 click	Execute last menu choice
Alt Button 2 click	Display last menu
Shift Button 2 click	Copy to scrap

Quick edit—If you select this option, mouse button 2 works as follows:

Taking this action...	Causes this result...
Button 2 click	Copy to scrap
Button 2 double click	Paste
Ctrl Button 2 click	Cut
Shift Button 2 click	Display popup menu
Shift+Ctrl Button 2 click	Execute last menu choice
Shift+Alt Button 2 click	Display last menu

Besides using SETUP, you can also change the mouse button 2 assignment by setting a parameter for **set_btn2_action**. This parameter, which can be either QUICK_MENU or QUICK_EDIT, is defined in win_ctrl.h. If **set_btn2_action** is not called, the default assignment for mouse button 2 is QUICK_EDIT.

The default popup menu that is displayed can be found in \brief\help\popup.mnu. You can customize this menu to suit your preferences. Processing for the popup menu can be found in \brief\help\popup.cb; it provides a multi-level menu structure similar to that of the Help menu.

Macro package support can be added to the menu by creating a sub-menu that can be called from the popup menu. To do this, add a line that contains the following:

- Popup menu choice for sub-menu followed by a semicolon
- "process_popup_menu"
- Sub-menu title (in quotes)

- File name of the sub-menu (in quotes)
- Height, width, line and column values (optional)

For example, the following line adds a new popup menu choice, "Display My Menu," that displays the sub-menu in my.mnu whenever it is selected. "New Menu," the title of the sub-menu, appears at the top when the sub-menu is displayed.

```
Display My Menu  ;process_popup_menu "New Menu" "my.mnu"
```

Check boxes, radio buttons, and push buttons

The Dialog Manager now supports check boxes, radio buttons, and push buttons.

PWB macro

As an alternative to restore.cb, BRIEF now provides a macro called pwb.cb that supports the Microsoft Programmer's Workbench. Restore.cb lets BRIEF save its state information in the PWB state file called current.sts. To use the PWB macro:

1. Set BFILE= to the drive and directory where current.sts resides. PWB keeps current.sts either in the current directory (if no INIT environment variable exists) or i n the first drive and directory specified in the INIT variable. For example, if INIT=c:\c600\init;d:\init, then set BFILE=c:\c600\init\current.sts.
2. Replace –mrestore in the BFLAGS variable with -mpwb. BRIEF creates private sections in the state file, and updates the [shared-] and [edit-] sections.

New key assignments

- *Ctrl+k*—Delete to beginning of line
- *Ctrl+u*—Redo
- *Ctrl+e*—Scroll buffer up in window

Compiler support

- Support for Borland C++ Version 2.0 has been added.
- Support for the following compilers has been upgraded to the latest versions: Lahey Fortran, F77L, Version 4.10, F77L-EM/32, Version 4.00, Alsys ADA, Version 4.4.2
- These upgrades required a change in the error handling macros.

Longer status line messages

Status line messages now can be 80 characters long. Long messages overwrite the Line:/Col: display and stay on the screen for at least three seconds, until the Line:/Col: display needs to be updated, or the time changes. File names and prompts displayed in the message area can also overwrite the Line:/Col: display.

About this manual

Whether you are an experienced BRIEF user or a beginner, we suggest you read through this chapter first, which tells you what's in this manual, all about the new 3.1 features, Borland Technical Support, and typographic conventions.

We also suggest you read through "Installing BRIEF" (Chapter 1) and the tutorial (Chapter 2), which offers an excellent introduction to the basic commands. The tutorial also shows you how to produce a macro to insert a copyright notice into your programs.

Once you are familiar with the basic commands, you can use the "Command overview" (Chapter 3), "Programming support" (Chapter 6), and "Command reference" (Chapter 8) to learn more complex commands.

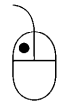
This manual is divided into the following chapters:

1	Installing BRIEF	Takes you through both a quick and complete installation process.
2	A tutorial	Walks you through some basic BRIEF commands and helps you produce a macro.
3	Command overview	Fully discusses BRIEF commands in a conceptual context.
4	Using special features	Discusses simple text formatting, reconfiguring the keyboard, the Restore feature, running other programs from within BRIEF, swapping for DOS users, and the dialog manager.
5	Flags and variables	Discusses customization flags that can be used when you start BRIEF.

6	Programming support	Discusses the many languages that BRIEF supports, and includes automatic indenting, template editing, compilation programs, and automatic error location.
7	Third-party packages	Includes instructions on running PVCS, TLIB, Microsoft Quick Help, PWB, and Sourcerer's Apprentice.
8	Command reference	Alphabetically lists all commands and associated macro names and keystrokes. Each item is cross-referenced to the command overview in Chapter 3.
9	Error messages	Alphabetically lists all error messages that could occur while running BRIEF, and discusses recovery solutions.
10	Questions and answers	Provides self-help for some common problems.
A	Help for new users	Provides instructions on the BRIEF screen, keyboard, and mouse, and overviews on commands, buffers, windows, scrap, and current position.

Typefaces and icons used in this manual

Here are the special typographic conventions used in this manual.

Type convention	Means
Keystroke	Key assignment
Name of Command	Descriptive command name, macro names.
`text onscreen`	Screen display, input/output
_(underscore)	The cursor location
DOS or *OS/2*	Operating system-specific instructions
	Mouse icon or mouse-specific instructions. The appropriate mouse button (1, 2, or 3) is indicated.

An example of the notation in the text might look like this:

> Press *End* to execute the **End of Line** command. The cursor should be located at the end of the first line, after the number 1:
>
> ```
> Line 1_
> ```

The *Quick Reference Guide* might come in handy while learning and practicing the commands presented in the tutorial.

How to contact Borland

Borland offers a variety of services to answer your questions about BRIEF.

Be sure to send in the registration card; registered owners are entitled to 60 days free technical support and will receive information on upgrades and supplementary products.

Borland Technical Support publishes technical information sheets on a variety of topics and is available to answer your questions.

800-822-4269 (voice) TechFax

TechFax is a 24-hour automated service that sends free technical information to your fax machine. You can use your touch-tone phone to request up to three documents per call.

408-439-9096 (modem)
File Download BBS
2400 Baud
9600 Baud

The Borland File Download BBS has sample files, applications, and technical information you can download with your modem. No special setup is required.

Subscribers to the CompuServe, GEnie, or BIX information services can receive technical support by modem. Use the commands in the following table to contact Borland while accessing an information service.

Online information services

Service	Command
CompuServe	`GO BORLAND`
BIX	`JOIN BORLAND`
GEnie	`BORLAND`

Address electronic messages to Sysop or All. Don't include your serial number; messages are in public view unless sent by a service's private mail system. Include as much information on the question as possible; the support staff will reply to the message within one working day.

800-851-9199
Technical Support
6 a.m. to 3 p.m. PST

Borland Technical Support is available weekdays from 6:00 a.m. to 3:00 p.m. Pacific time to answer any technical questions you have about Borland products. Please call from a telephone near your computer, and have the program running. Keep the following information handy to help process your call:

- product name, serial number, and version number
- brand and model of the hardware in your system
- operating system and version number—use the DOS `VER` command to find the version number
- contents of your AUTOEXEC.BAT and CONFIG.SYS files (located in the root directory (\) of your computer's boot disk)
- contents of your WIN.INI and SYSTEM.INI files (located in your Windows directory)
- daytime phone number where you can be reached

If the call concerns a software problem, please be able to recount the steps that will reproduce the problem.

408-461-9000 (voice)
Customer Service
7 a.m. to 5 p.m. PST

Borland Customer Service is available weekdays from 7:00 a.m. to 5:00 p.m. Pacific time to answer any nontechnical questions you have about Borland products, including pricing information, upgrades, and order status.

C H A P T E R

1

Installing BRIEF

SETUP automates the installation and configuration of BRIEF. After installation, you can also use SETUP to reconfigure BRIEF.

SETUP lets you perform the following basic tasks:

- Choose which files to copy from the distribution disks, and specify the drives and directories where they will be stored
- Choose from the options available for your display and keyboard hardware

DOS

- Ensure that DOS parameters are set for optimal performance
- Customize BRIEF support for the languages and compilers you use
- Set many BRIEF options to suit your preferences

How SETUP configures BRIEF

BRIEF configures itself by looking at information from the following sources:

- The operating system environments. These can be entered manually at the prompt or stored in one of the following files:

  DOS `autoexec.bat`

  OS/2 `os2init.cmd`, `config.sys`, or `config.os2`

  BRIEF uses the most recent setting to set options.

- Initialization files (compiled BRIEF macros). The "initials" macro, which is a file named with your initials, contains macros that tailor BRIEF to your preferences.

Modifying system files

As part of the configuration process, SETUP might modify the following files:

DOS	`initials macro, autoexec.bat, or config.sys`
OS/2	`initials macro, os2init.cmd, config.sys, or config.os2`

You will have the opportunity to preview changes before continuing.

If you let SETUP modify the files, backup copies of the original (unchanged) files will be saved in the same directory as the original files, with an extension of `.bak`. If you prefer, you can save the changes to a separate file and manually update the necessary files later.

Complete SETUP instructions are included in "Complete installation" later in this chapter.

Quick installation and preview

If you would like to use the tutorial (in Chapter 2), follow these instructions to install BRIEF, which ensures the settings are correct. Later, follow the instructions in the section "Complete installation" to reconfigure BRIEF for your preferences.

If you don't plan on using the tutorial, follow these instructions for making backup copies and then proceed to "Complete installation."

Backing up

Use `diskcopy` to make copies of your BRIEF diskette(s). Do *not* use the `copy` or `xcopy` commands, as they won't copy all of the required information. Put the originals in a safe place for emergency use.

Installation

If you are using DOS, you can install BRIEF on either floppy disks or your hard drive. If you are using OS/2, you can only install BRIEF onto your hard drive.

To install onto a 360K floppy disk, use the copy of the program disk you just made as your destination disk (install BRIEF onto the copy). You can only do a partial installation onto 360K floppies—only the executables, compiled macros, and **initials** macro can be copied.

To install onto any other floppy, you will need one that is blank and formatted.

To install onto a hard disk, you will need approximately 700K free. A partial installation takes less space.

The number of Program and SETUP diskettes may vary, according to your system and the package you purchased.

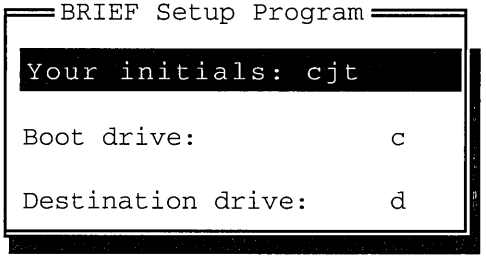
Starting SETUP

If you need help at any time during SETUP, press *Alt+H*. A screen describing the currently selected item will appear.

Insert the SETUP Disk into drive A or B, depending on your floppy drive configuration and diskettes purchased. Set this drive as the default (if it is not currently the default, type `a:` or `b:` at the prompt and press *Enter*).

Type `setup` and press *Enter*. An informational screen will appear. Read it, then press *Esc*. To exit SETUP at this point, press *Esc* twice. The following dialog box will appear:

Initial SETUP screen



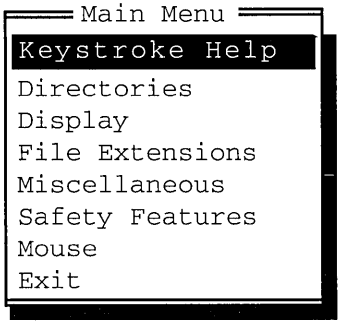
Type in

- your initials
- the boot drive
- the destination drive

Your initials will label your **initials** macro; the boot drive is where your computer boots; the destination drive is where BRIEF will be installed.

When the information is correct, press *F10* to continue. The SETUP Main Menu will appear.

SETUP Main Menu



Customizing for your environment

Minimal customization is required to run the Tutorial on your system (this includes the BRIEF subdirectory, display, keyboard, and mouse). For full setup instructions, see "Complete installation" later in this chapter.

Directories

SETUP installs BRIEF in the `\brief` subdirectory on the destination drive. Select `Directories` and the File and Directory Setup screen will appear. Press *Tab* to move into the destination directory column for the executables and type the directory name where these files should be stored. Press ↓ to move to the next directory, and continue the process for all directories on the screen. Press *F10* to save changes and return to the Main Menu or *Esc* to return without saving.

You can accept the default directories already assigned by pressing *F10* without changing anything. If you are doing a partial installation, you must be sure to change the `Copy Files` prompt to `No` for all files that are not to be copied.

Display

If you have a Monochrome Display Adapter, or a Hercules Graphics Card, Graphics Card Plus, or Incolor Card, skip to the next step. If you're not running DESQview, Ready!, SmartKey, Microsoft Mouse Menus (or other mouse pop-up menu programs), SideKick, SuperKey, or Turbo Lightning, also skip to the next step. Otherwise, select `Display`. Select `Video Page Test`. Type `y` to answer the question about resident programs that appears. Press *F10* twice to return to the main menu.

Keyboard compatibility

If your keyboard is fully compatible with the IBM PC standard, or if you are unsure about its compatibility, skip to the next step. Otherwise, select `Miscellaneous`. Select `no` at the question about keyboard compatibility and press *F10* to return to the Main Menu.

101-key keyboard

If you are using a 101-key keyboard and want BRIEF to recognize it, select Miscellaneous. Select Yes at the question about extended keyboard support and press *F10* to return to the main menu.

Mouse

If you don't have a mouse, skip to the next step. Otherwise, select `Mouse`. Enable the mouse by selecting `Yes` at the first prompt and answer the other questions. Press *F10* to return to the Main Menu.

Exit and update

Select `Exit`, then press *Enter*. Select `Install BRIEF` when the second menu appears.

SETUP will now copy the BRIEF files onto the destination disk. After copying the files, SETUP will prompt:

```
What should Setup do with changes to file name.ext:
Update file, Show onscreen, Write alternate file,
Discard [uswd]?
```

This prompt will appear for each `file name.ext` that needs to be updated.

DOS `config.sys, autoexec.bat`

OS/2 `os2init.cmd, config.sys, config.os2`

If you want to view the changes to the file, type `s`. Then type `u` to update the file (or `w` to write the changes to a different file, which you must incorporate manually). Type `d` to discard all changes.

DOS SETUP may modify `command.com` to expand the default environment size in certain versions of DOS prior to 3.1. If the environment size is not expanded, there may not be enough room for all of BRIEF's environment variables. The change has been thoroughly tested, and has few, if any, side effects. However, you will be given the option of refusing it, with the prompt `Patch command.com to change environment size [yn]?` If you have Compaq MS-DOS 3.0, be sure to answer `n`. Otherwise, answer `y` to increase the environment size or `n` to leave it unchanged.

Reboot Remove the diskette and reboot your computer to put the changes into effect.

Proceed to the tutorial in Chapter 2. When you have finished, return to this point to configure BRIEF to your preferences.

Complete installation

This section explains how to use SETUP in more detail. If you haven't performed a quick installation, be sure to make backup copies of the BRIEF disks before starting.

The sample screenshots used here are taken from a DOS system. OS/2 commands, when different, are explained in the paragraphs following the screenshot.

SETUP can be started in Install mode or Configure mode.

Install mode

Install mode lets you install and configure BRIEF and is used if SETUP is started from the SETUP Disk or, under DOS, a copy of the SETUP Disk. If you've already done a Quick Installation (which runs in Install mode), skip to "Configure mode" following. Otherwise, start SETUP in Install mode by following these steps:

- Insert the SETUP Disk into drive A or B, depending on your floppy drive configuration and diskettes purchased.
- Set this drive to the default (if it is not currently the default, type `a:` or `b:` at the prompt and press *Enter*).

Skip to "SETUP keystrokes."

Configure mode

Configure mode is used if SETUP is started from any disk other than the SETUP Disk. In Configure mode, you can configure BRIEF, but you cannot install it onto another disk. If you've performed a Quick installation or want to reconfigure your current BRIEF setup, follow these steps to start SETUP in Configure mode:

- Set the default drive to the destination drive from the previous installation (for example, if you installed onto drive C, type `c:` at the prompt and press *Enter*).
- Change to the `\brief` directory (or the directory where you installed the Miscellaneous files), by typing `cd \brief`.

SETUP keystrokes

During SETUP, you can use the following keystrokes while on a menu screen:

Using keys in menus

- ↑ and ↓

  Select the button above (or below) the currently selected button.

- *Enter*

  Completes the selection.

- *Esc*

  Returns to the previous menu (if any).

- Typing a letter

  Moves the cursor to the next menu item starting with that letter (if any).

- *Home*

  Moves to the first selection on the menu.

- *End*

  Moves to the last selection on the menu.

Using keys in dialog boxes

During SETUP, you can use the following keystrokes while in dialog boxes to make changes and move the cursor around the screen:

- *Tab*, ↓, or *Enter*

  Move the cursor to the next field on the screen. If the current field is the last one on the screen and *Enter* is pressed, the screen is automatically saved and the previous menu appears.

- *Shift+Tab* or ↑

  Move the cursor to the previous field on the screen.

- *Esc*

  Cancel the changes made in the current screen and return to the previous screen.

- *F10*

  Save the changes made in the current screen and return to the previous screen.

- ← and →

  Move the cursor to the previous (or next) character in character fields, or to the previous (or next) item in multiple choice fields.

- *Home*

  Move the cursor to the first character in the current field.

- *End*

  Move the cursor to the last character in the current field.

- *Ins*

  Change the current typing mode from overstrike to inserting, and vice-versa.

- *Del*

  Delete the character at the cursor.

- *Backspace*

  Delete the character before the cursor.

Other SETUP features

SETUP lets you install BRIEF and set built-in options. BRIEF, however, can be customized far more extensively with the Macro Language. With this, you can change the way existing commands work and write your own new commands. After reading this manual and becoming familiar with BRIEF, read the *Macro Language Guide* to learn how to do extensive reconfiguration.

SETUP does not reconfigure the BRIEF keyboard. See the **keys** command in the *Macro Language Guide* or "Using special features" (Chapter 4) for more information.

There are several infrequently used command line flags that SETUP does not change. See "Flags and variables" (Chapter 5) for a complete explanation of all flags.

Starting SETUP

Getting help

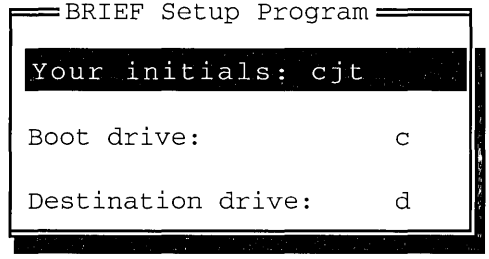
If you need help at any time during SETUP, press *Alt+H*. A screen describing the currently selected item will appear.

Type `setup` and press *Enter*.

An informational screen will appear. Read it, then press *Esc*. To exit SETUP at this point, press *Esc* twice.

The following dialog box will appear:

Initial SETUP screen



Type in

- your initials
- the boot drive
- the destination drive

Your initials will be used to label your initials macro. The boot drive is where your computer boots; the destination drive is where BRIEF will be installed.

If you are reconfiguring BRIEF after running the Tutorial, you must run SETUP from the floppy disk because any changes to these values will require re-installation.

When the information is correct, press *F10* to continue. The SETUP Main Menu will appear.

SETUP Main menu

The following instructions deal with each menu option, in the order given. The displayed values for each option will either be those already entered (if reconfiguring) or default (if installing).

Keystroke help

This option provides a complete listing of keystrokes available in SETUP, divided into "In menus," "In fields," and "Between fields."

Whenever you need help with keystrokes, or to refresh your memory, choose this option. Press *Esc* when finished.

Directories

This option is used to define the subdirectories used by BRIEF for the different types of files. The following screen will appear:

SETUP File and Directory Setup menu

```
File and Directory Setup

Class of Files        Copy Files   Destination Directory
Executables           Yes No Ask   c:\brief

CBRIEF macro source   Yes No Ask   c:\brief\macros

Compiled macros       Yes No Ask   c:\brief\macros

Help files            Yes No Ask   c:\brief\help

Miscellaneous files   Yes No Ask   c:\brief

Initials macro        Yes No Ask   c:\brief\macros

Class of Files        Move Files   Destination Directory
Orig. macro source    Yes No Ask   c:\brief\macros\old_src
```

These are the default values BRIEF uses during installation.

Executables are the executable program files, including the macro compiler, batch files, PIF files, device drivers, and font files, and are required for all installations.

CBRIEF macro source are the new version source files. They are only present when upgrading BRIEF and are not required, unless you want to modify the macros or the default key assignments. They cannot be copied for a 360K installation.

OS/2 The only ***device driver*** included is `101key.dll`.

Compiled macros are the macro files explained in the *Macro Language Guide* and contain many of the commands. They are required for all installations.

Help files are the on-screen help information. They are not required, but if they are not installed, you cannot use *Alt+H* for on-screen help.

Miscellaneous files contain special instructions for your system and preferences, and include SETUP and `read.me`. They are not required, but are recommended for all installations except 360K and 720K.

Initials macro contains your specific user instructions and is required.

Original macro source files are the old version source files. They are only present when upgrading BRIEF and are not required.

When installing, or if making directory changes when reconfiguring, you can control how BRIEF copies the files by moving to the "Copy Files" column and selecting `Yes` to copy without prompting, `No` not to copy, or `Ask` to prompt before copying.

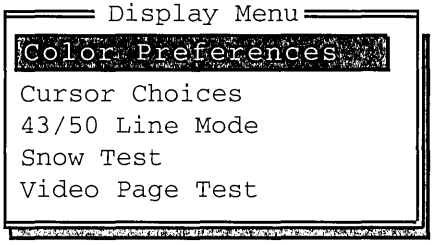
If you don't want BRIEF to use these directories, use the cursor keys to move to the Destination Directory column and change them.

When all information is correct, press *F10* to keep your changes and return to the Main Menu. Press *Esc* to return to the Main Menu without saving them.

Display

This option is used to select your display options. The following menu will appear:

SETUP Display menu



Color Preferences allows you to choose foreground, background, highlighted window titles, and normal and error message colors. It is also used to turn window borders on and off.

The current colors for foreground text and normal and error messages are displayed next to boxes showing the 16 available colors. The following prompts also appear:

```
Background color?           7
Foreground color?           15
Window title color?         7
Error message color?        15
Borderless background?      0
Do you want borders?      Yes No
```

Using the numbers under the boxes, enter the desired colors for each of the above. You are not allowed to enter the same colors for foreground and background, because the text will be invisible.

Choose `Yes` or `No` for window borders. If you are going to use a mouse, borders must be on.

When finished, press *F10* to return to the Display Menu.

Cursor Choices allow you to choose how BRIEF displays the cursor in four different modes:

- Normal insert mode, when you are in insert mode and the cursor is at an existing character (choice 1)
- Normal overstrike mode, when you are in overstrike mode and the cursor is at an existing character (choice 2)
- Virtual insert mode, when you are in insert mode and the cursor is in virtual space (past the end of a line or a file or within an expanded tab character) (choice 3)
- Virtual overstrike mode, when you are in overstrike mode and the cursor is in virtual space (choice 4)

You can choose to use the default cursors associated with each mode or change them around. You can also make them all the same. Press *F10* to return to the Display Menu.

The following cursor shapes are available (choices 1 to 4, from left to right):

43/50 Line Mode is available only if you have an Enhanced Graphics Adapter (EGA) or compatible, a PS/2 Video Graphics Array (VGA) or compatible, or a Hercules Graphics Card Plus/Incolor display adapter. It gives you the option of displaying 43 or 50 lines of text on the screen, instead of 25.

Choose whether or not you want 50 line mode. Press *F10* to return to the Display Menu.

Snow Test is only necessary if you have a Color Graphics Adapter (CGA) compatible monitor. It determines the maximum screen "refresh" time that doesn't produce "snow."

Video Page Test is used only if you don't have a Monochrome Display Adapter (MDA) or Hercules Graphics Card, Graphics Card Plus, or Incolor Card. Because of BRIEF's ability to shell out to DOS, several pages of memory are used for the screen. This enables you to exit to DOS, then return to BRIEF, and still see both screens. Some display adapters fail to provide extra pages of memory, so when you exit BRIEF, the cursor (sometimes, the entire screen) disappears.

This option will display a box where the cursor randomly moves around, and the following prompts:

```
Can you see the cursor moving?          Yes No
```

```
Do you use any resident programs
from the list of incompatibles
(press Alt-H to see the list)?          Yes No
```

If you can't see the cursor on the screen, select `No` at the first prompt. If you can't see anything, press *Esc* to choose `No` automatically.

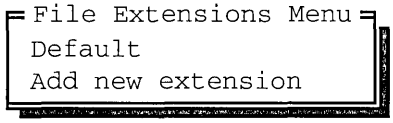
If you are using any of the memory-resident software on the list, select `Yes` at the second prompt. BRIEF will use only one page of display memory (it will not preserve your DOS screen).

Press *F10* to return to the Display Menu.

File extensions

The File Extensions Menu is used to customize BRIEF for the languages and file types you use. The following menu will appear:

SETUP File Extensions menu



Select `default` to see a screen containing information about how BRIEF handles any file type that does not match those listed on the File Extensions Menu.

Adding a language

To customize for your programming languages, select `Add new extension`. At the prompt, enter the file extension you want to add.

SETUP provides special support for the following file types:

- `.m` (BRIEF macro files)
- `.cb` (CBRIEF macro files)
- `.c` (C language files)
- `.h` (Header/include files)
- `.cpp`, `.cxx` (C++ language files)
- `.pas`, `.p` (Pascal language files)
- `.for`, `.ftn` (FORTRAN language files)
- `.cob`, `.cbl` (COBOL language files)
- `.mod`, `.def` (Modula-2 language files)
- `.prg` (dBase language files)
- `.pgm` (Eagle language files)
- `.sql`, `.prs` (SQL language files)
- `.lsp` (Lisp language files)
- `.pro`, `.dba`, `.ari` (Prolog language files)

- `.bas` (BASIC language files)
- `.ada`, `.pkg`, `.lib`, `.adb`, `.ads`, `.sub` (Ada language files)
- `.asm`, `.inc` (Assembly language files)
- `.txt`, `.doc` (Documents)

For each extension, you can set the following options:

- Tab stops. Note that the numbers represent the columns of the tab stops, not the number of spaces between tabs. Specifying a first column of 4 would put in 3 space tabs (moving the cursor to column 4).

  Also, the difference between the last two numbers will be used for the remaining tab stops. For example "4 7" is the same as "4 7 10 13". For more information, see "Tab stops" under "Editing text" in "Command overview" (Chapter 3).

- Language type. Allows you to choose the *language type* associated with an extension. This allows you to turn on language-specific features for extensions that BRIEF doesn't automatically recognize, but whose language is similar to one of the recognized extensions.

- Automatic indenting. Many file types include an automatic indenting option. For more information, move the cursor to this entry and press *Alt+H* or see "Programming support" (Chapter 6).

- Wordwrap (and margin, if wordwrap is on). This is set "on" for `.txt` and `.doc` files, which enables you to use BRIEF for memos and other documents. This is also explained in the help entry and in "Using special features" (Chapter 4) under "Simple text formatting."

- Compiler choice. Select your compiler (if any). Default configurations are provided for every major PC compiler of these languages. Once a compiler is selected, it can be customized by changing the command name field. If you don't find your compiler listed on the screen for its extension, choose `Custom Compiler` and fill in the invocation requirements yourself.

- Warnings setting. Most compilers consider warnings to be non-fatal and won't tell BRIEF they've occurred. This option lets you determine whether BRIEF should consider the compilation to have failed when warnings have been produced, even if the compiler says otherwise. It's useful if you want to ensure that your code doesn't produce any compiler diagnostics.

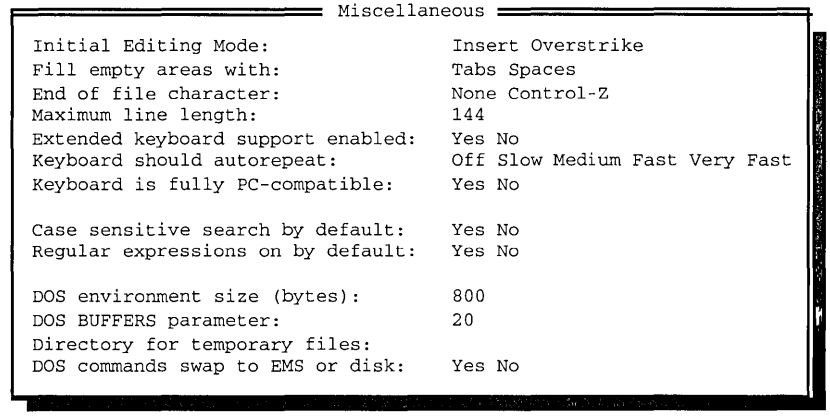
Any extensions not listed on the File Extensions menu will take the default behavior. If you want BRIEF to behave differently when editing files of a particular type, you must add that type to the File Extensions menu.

To remove an extension from the File Extensions menu, move the cursor to that extension and press *Del*.

Miscellaneous

This menu allows you to set some of your editing preferences and parameters. The following screen appears:

SETUP Miscellaneous menu



- The default typing mode when BRIEF is started (insert or overstrike)
- Whether blank areas should be filled with spaces or "hard" tabs
- Whether files should be terminated with an EOF (ASCII 26) character when they are written
- The maximum line length
- Whether extended keyboard support is enabled by using the 101-key keyboard driver, or not
- The rate of keyboard auto-repeat
- *DOS* Whether your keyboard is completely compatible with the IBM PC standard
- The default search characteristics when BRIEF is started (case sensitivity and regular expression sensitivity)
- *DOS* The maximum environment size
- *DOS* The number of DOS buffers
- The directory where temporary files should be stored

DOS

- Whether BRIEF should swap itself to EMS/EEMS/LIM expanded memory or disk or when running other programs

Choose your preferences and enter other options, if desired. Press *F10* to return to the SETUP Main Menu.

Safety features

This lets you choose safety options that can protect your work. The following screen appears:

SETUP Safety Features menu

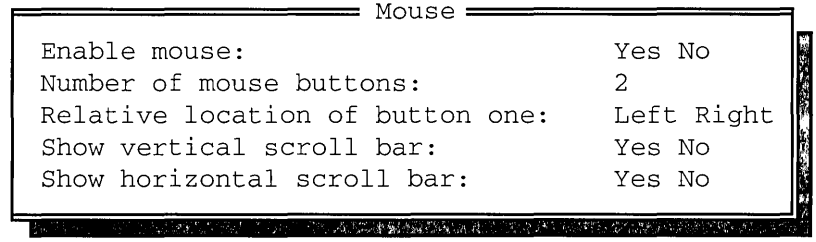
```
                    Safety Features

Automatic file saving when keyboard idle:          Yes No
Idle time (seconds) for autosave:                  120

Create backup files by default:                    Yes No
Put backup files in current directory:             Yes No
Put backup files in directory:     \brief\backup

Maximum number of undoable commands per buffer:    30

Restore session when no file name specified:       Yes No
```

- Whether to automatically save your files after you've been idle for a certain amount of time (autosave). Since most changes are kept in RAM, a power failure can mean a loss of work if the files have not been saved. It is recommended that you activate this feature and choose a reasonable idle time. We suggest using 120 seconds. These files use the `.asv` extension and are deleted when you exit BRIEF under normal conditions. If a file name exists with `.asv`, alternative extensions (`.as1` through `.as9`) will be assigned. You must still save your files before exiting. You can override this value using the `-i` or `-m` flags, or you can add a line to your startup file that sets BFLAGS. See "Flags and variables" (Chapter 5) for more information.
- Whether to save backup copies of files. You can set their directory or use the current one.
- The maximum number of commands (up to 300) that can be undone per buffer.
- Whether BRIEF should automatically restore your session when restarted without a file name (including the files you were editing, window layout, positions, search patterns, bookmarks, and the states of several other settings).

Mouse

This menu lets you enable the mouse and set its parameters. The following appears:

SETUP Mouse screen



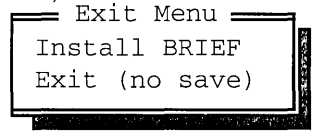
- Select `Yes` at the first prompt to enable the mouse.
- Select the number of mouse buttons (2 or 3). If you have a 2-button mouse, you press both buttons at the same time for a logical third button.
- Select the relative location of mouse button one (left or right). This enables you to use the mouse with either your right or left hand. Mouse button one is always the one you press with your index finger, regardless of which hand you're using.
- Select your preferences for vertical and horizontal scroll bars, which can only be enabled if borders are on. The right border is the vertical scroll bar and the bottom border is the horizontal scroll bar.

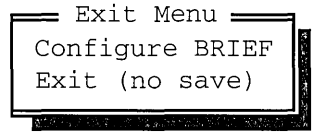
Press *F10* to return to the Main Menu.

Exit

This menu lets you exit without saving any of your choices or exit and install or configure BRIEF, using all specified parameters and directories. One of the following menus appears:

SETUP Exit menus





If you choose `Install` or `Configure`, the program will request the various Program and SETUP disks. Follow all screen prompts. Messages will be displayed as the different files are copied. If you requested `Ask` at the file directories screen, you will be prompted for a response before the files are copied.

The macro source code is stored in one file. SETUP automatically retrieves the code from this file and creates the separate source code files on the destination disk. If you want to copy these files yourself, they are located in the file `source.arc`. This file can be "unpacked" with most ARC-compatible utilities.

After copying the files, SETUP will prompt:

```
What should Setup do with changes to file
name.ext:
Update file, Show onscreen, Write alternate file,
Discard [uswd]?
```

This prompt will appear for each `file name.ext` that needs to be updated.

DOS `config.sys, autoexec.bat, initials macro`

OS/2 `os2init.cmd, config.sys, config.os2, initials macro`

If you want to view the changes to the file, type `s`. Then type `u` to update the file (or `w` to write the changes to a different file, which you must incorporate manually). Type `d` to discard all changes.

DOS SETUP may modify `command.com` to expand the default environment size in certain versions of DOS prior to 3.1. If the environment size is not expanded, there may not be enough room for all of BRIEF's environment variables. The change has been thoroughly tested, and has few, if any, side effects. However, you will be given the option of refusing it, with the prompt `Patch command.com to change environment size [yn]?` If you have Compaq MS-DOS 3.0, be sure to answer `n`. Otherwise, answer `y` to increase the environment size or `n` to leave it unchanged. If necessary, you can recopy `command.com` from your DOS disk.

If your initials macro is changed, SETUP automatically recompiles it. If you make your own additions to the initials macro, SETUP will preserve most changes automatically. The exceptions are changes made to the file extension macros and **_init** macro. To preserve your changes to the file extension macros (and prevent SETUP from changing any statements in them), delete the comment reading:

```
;**  Overwritable by Setup
```

located at the top of each file extension macro you modify. Do not make any changes to the **_init** macro, as SETUP will always overwrite it. Instead, put additions into the initials macro located below **_init** in the file. SETUP ensures that this macro is called when BRIEF starts.

DOS reboot

Remove the diskette and reboot your computer to put the changes into effect.

If SETUP updated `config.sys` and `autoexec.bat`, it will ask for permission to reboot:

```
Reboot now [yn]?
```

Type `y` if you want to reboot your computer (**all** information on RAM disks [virtual disks] will be lost). Type `n` to return to DOS.

If SETUP did not update `config.sys` and `autoexec.bat`, incorporate the changes from the alternate file yourself and reboot your computer.

Installation and configuration are now complete.

C H A P T E R

2

A tutorial

This chapter provides you with an overview of basic BRIEF instructions. You also will produce a customized macro, with an assigned keystroke, that will insert a copyright notice into your programs. It is designed to show you how quickly you can start producing with BRIEF and should take about 20 minutes to complete.

This chapter includes

- Help and undo
- Create, save, exit
- Cursor movement
- Editing text
- Blocks and marks
- Search and translate
- Customizing the tutorial file
- Compile and run

If you are familiar with other editors and want the keystrokes for the above commands (but don't want to complete the tutorial), skip to "Customizing the tutorial" at the end of this chapter. A complete command overview chart is included.

DOS Set the BCCB environment variable by entering

```
set BCCB="cb -d %%s"
```

At the prompt, type `cd c:\brief\macros`. If you have installed BRIEF somewhere other than C:, change to the correct drive and then enter that drive letter when typing this command.

To start BRIEF, type

```
b file name.ext
```

where

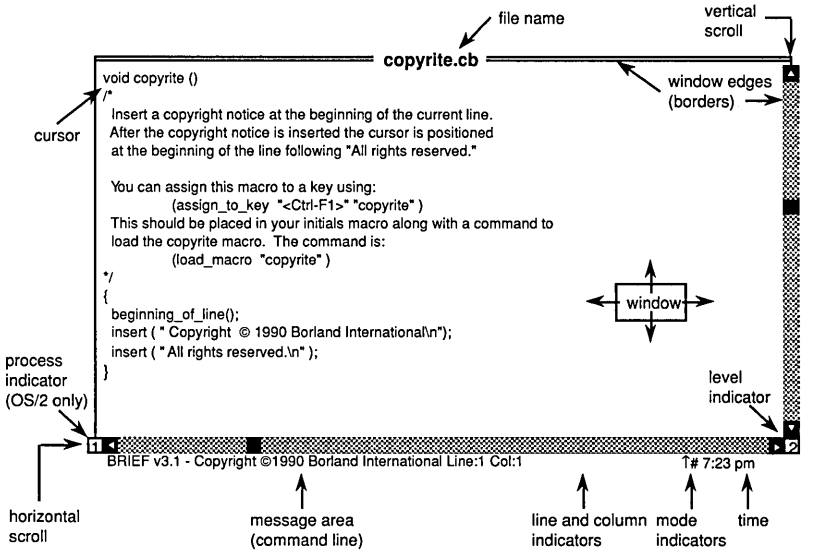
`b`	is the command you use to invoke BRIEF (the actual executable program is called `b.exe`).
`file name.ext`	is the name of the file to edit. You can specify an existing file or the name of a file to create. You can also specify multiple files (see "Flags and variables" for more information).

A macro file that inserts a copyright notice into your files has been included in the `\brief\macros` subdirectory. At the prompt, load the macro file by typing

```
b copyrite.cb <Enter>
```

The following screen appears:

BRIEF screen



If you type the file name incorrectly, you will see a blank screen. The file name you typed appears on the top line (where **copyrite.cb** is displayed above) and `New file (unable to open file name.ext)` appears on the command line.

To start over, press *Alt+X* to exit BRIEF. At the prompt, retype the line.

Once the tutorial file is displayed on the screen, you are ready to start.

Help and undo

These commands may become your favorites while running BRIEF. They allow you to get onscreen help and undo up to 300 commands.

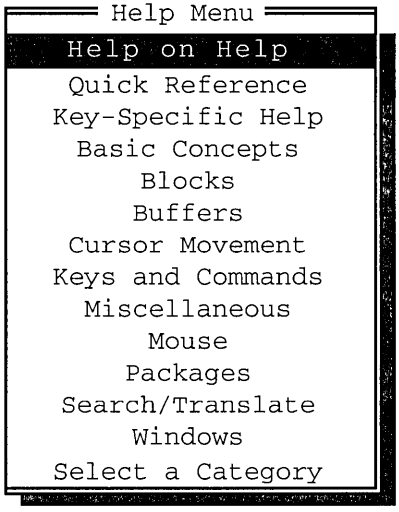
Help

During the tutorial, you can get help by

- Pressing *Alt+H* at any time for the general help menu
- Pressing *Alt+H* after entering a command for specific help

Press *Alt+H* and the general help menu is dispayed onscreen:

The help menu



Any topic can be selected by moving the cursor to it and pressing *Enter*. For example, move the cursor to the third topic in the menu, `Key-Specific Help` and press *Enter*. The following message is displayed:

```
Press the key you want help on, <Esc> to exit.
```

Press *Alt+D* to find out what the keystroke currently does. A help screen replaces the general help menu. The key-specific help tells you that *Alt+D* deletes the current line, moving the next line up to take its place.

When a help screen is being displayed, *Alt+H* displays a window that shows the key assignments for the command being described.

Press *Esc* once to return to the general help menu, and again to return to the buffer from this menu.

Help from a prompt

Sometimes specific help on how to respond to a prompt is needed. Rather than go to the help menu and select categories, you can get immediate help using the context-specific help feature.

For example, press *Alt+G*, the **Go to line** command. At the prompt:

```
Go to Line: _
```

Press *Alt+H*. A help screen appears that explains the command. The help screen is designed to help you provide a response to the prompt.

Now press *Esc* once to return to the **Go to line** prompt, and again to cancel the command.

Undo

BRIEF's **Undo** command protects you from *yourself*. It allows you to undo text manipulation commands, and in fact, each command you use to move the cursor or add and delete text can be undone (assuming you have not exited BRIEF or written the file to disk).

The undo stack

Many editors permit you to undo the last deletion, but in BRIEF you can keep pressing *Alt+U* to undo the last 30 commands (you can use SETUP to set BRIEF to undo up to *300* commands, but 30 is usually sufficient). BRIEF remembers the commands you execute so it can undo them in reverse order. The remembered list of commands is called the undo stack.

Save and exit

Saving the file

Text is not permanently saved on disk until you save it with the **Write** or **Exit** commands.

Any time you want to save a file without exiting, use the **Write** command, which is assigned to *Alt+W*. It is a good idea to save your work occasionally, as you edit, in case of a power failure or similar disaster.

You can also use Autosave, discussed in "Safety features" under "Complete installation" in Chapter 1.

Exiting BRIEF

To exit BRIEF at any point in the tutorial, use the **Exit** command. Press *Alt+X*.

If you execute the **Exit** command after text has been entered in the file, you will be given three choices:

```
1 buffer has not been saved. Exit [ynw]? _
```

You can press `w` to write the file to disk and exit to the operating system. If you press `y` at the prompt, the buffer is not saved, and you return to the operating system. If you press `n`, the command is cancelled and you return to the editor.

Cursor movement

When you started the tutorial and called the copyright macro, the following text appeared on your screen:

```
void copyrite ()
/*
   Insert a copyright notice at the beginning of the current line.
   After the copyright notice is inserted the cursor is positioned
   at the beginning of the line following "All rights reserved."

   You can assign this macro to a key using:
      (assign_to_key "<Ctrl-F1>" "copyrite" )
   This should be placed in your initials macro along with a command to
   load the copyrite macro.  The command is:
      (load_macro  "copyrite" )
*/
{
   beginning_of_line();
   insert ( " Copyright 1990 Borland International");
   insert ( " All rights reserved.\n");
}
```

You need to modify this macro file so you can use it.

Moving the cursor

The cursor should be under the `v` in `void`, since BRIEF always places the cursor at line 1, column 1 when starting.

By character

Arrow keys move the cursor either one character left or right or one line up or down.

Move the cursor to the `A` in `All rights reserved` on the fifth line using the arrow keys.

Next word
Previous word

Instead of moving the cursor character by character, hold the *Ctrl* key and press the left or right arrow keys to move word by word.

Press *Ctrl+* → to move the cursor to `rights`. Press *Ctrl+* ← to move the cursor back to `All`.

Beginning of line
End of line

To move the cursor quickly to either the beginning or end of a line, use *Home* and *End*.

Press *Home* to move the cursor to the beginning of line 5. Press *End* and the cursor moves to the space to the right of the quote signs after `reserved`.

Newline character

This space is actually the newline character, marking where *Enter* was pressed to end a line. If you delete the newline character by pressing *Del*, the blank line that follows disappears and the rest of the text moves up one line.

Press *Del* to delete the newline character. Press *Alt+U* to undo the deletion and restore the blank line.

Virtual space

The blank area on the screen to the right of the newline character is called virtual space. Press → to move the cursor right one space. The cursor will change shape.

Since the Tutorial is using default SETUP values, the cursor changes shape when it is moved into one of four areas: normal insert, normal overstrike, virtual insert, and virtual overstrike. See "Display" under "Complete installation" in Chapter 1 for more information on changing cursor shapes.

Go to line

If you know which line you want to move the cursor to, you can press *Alt+G* for **Go to line** and enter the line number.

Escape

Commands with prompts, such as **Go to line**, can be cancelled before they are executed by pressing *Esc*. Press *Alt+G*. When you get the prompt, press *Esc*. The message `Command cancelled.` is displayed and the cursor stays at its last position.

Mouse

If you have installed a mouse, you can use it instead of the cursor movement keystrokes. Press *HomeHome* to move the cursor to `void`. Position the mouse cursor on the `A` in `All` on line 5 and click *Mouse Button 1*. The text cursor will blink when it's in place.

Page up
Page down

PgUp and *PgDn* move the text screen by screen in the desired direction.

Press *PgDn*. The cursor changes shape, since it is in virtual space after the end of the file.

Press *PgUp* to return to the same position in the text.

Editing text

BRIEF is preset to insert text, not overwrite it, so any characters you type are inserted just before the cursor position. The exercises in this Tutorial are based on insert mode. Changing to overstrike mode is easy. See **Insert mode toggle** under "Editing text" in Chapter 3 for more information.

Splitting a line
Adding a line

Splitting or adding blank lines use the same format. Position the cursor as desired and press *Enter*. If the cursor was anywhere in the line, the second part will move to the next line. If the cursor is at the beginning or end of a line, a blank line will be inserted.

Delete a character

Earlier you learned how to delete the newline character using *Del*. This key will delete the character at the cursor. You can also use *Backspace* to delete the character to the left of the cursor.

Position the cursor at the `A` in `All`. Press *Del* and the `A` disappears. Press *Backspace* and the `"` disappears.

Retype the two characters to restore them.

Delete previous word
Delete next word

You can also delete by word in either direction. BRIEF varies word definitions according to the programming language being used. For example, press *Ctrl+* ← to move left one word, and the cursor will move to the `A` in `All`. Press the same keys again, and the cursor will move to the `"`.

Position the cursor on the `r` in `rights`. Press *Ctrl+Backspace* to delete the previous word `All`. If you chose the 101-key keyboard option during "Quick installation," you can press *Alt+Backspace* to delete the next word `rights`. Press *Alt+U* to restore the word(s) you deleted.

When entering lines of text, **delete previous word** will delete the word before the cursor, then the preceding space, then the next word, etc.

Delete line
Delete to end of line
Delete to beginning of line

If you want to delete the entire line where the cursor is positioned, press *Alt+D*. If you want to delete from the cursor to the end of the line, press *Alt+K*. If you want to delete from the cursor to the beginning of the line, press *Ctrl+K*.

Position the cursor at the `"` in `"All`. Press *Alt+K*. The rest of the line (`"All rights reserved."` disappears.) Press *Alt+D*. The whole line disappears.

Press *Alt+U* twice to restore the entire line.

Tabs

The tab positions used with this macro may not be ones you feel comfortable with. See **Tab** in "Cursor movement" and **Tab stops** in "Editing text" in Chapter 3 for more information.

Also, you can use Regular indenting, Smart indenting, and Template editing if you desire. See "Programming support" (Chapter 6) for more information.

Blocks

Any part of a file (other than a single line) that is to be cut, copied, or pasted must be marked into a block. The **Mark** command highlights the desired text for further action.

Mark

Position the cursor in column 1 of line 7. Press *Alt+M* to start marking the block. Use the right and down arrow keys to position the cursor at the end of line 11. The second paragraph should now be highlighted.

Copy to scrap

Copy this block to the scrap buffer by pressing *Keypad Plus*. The block is now unmarked and the highlighting will disappear. The message `Block copied to scrap.` appears on the command line.

Mark with a mouse

It is possible to use the mouse to mark blocks of text. Position the cursor in the same place as before (column 1, line 7). Hold the *Alt* and *Ctrl* keys and *Mouse Button 1* while you move the mouse to the end of line 11, then release it (this is called "dragging"). The text will be highlighted.

If you wanted to copy this block to the scrap with the mouse, you would click *Mouse Button 2*.

Cut to scrap with mouse

This time, however, you will cut the text to the scrap. Hold down the *Ctrl* key and click *Mouse Button 2*. The highlighted text will disappear and the other lines will move up.

If you wanted to cut this block to scrap without using the mouse, press *Keypad Minus*.

Paste from scrap with or without a mouse

Since the paragraph gives key assignment instructions for the copyrite macro, you need to put it back in place. You can paste text from the scrap in one of two ways: with *Ins* or by double-clicking *Mouse Button 2*. Position the appropriate cursor (text or mouse), then restore the paragraph. If the cursor is not positioned correctly, the text will be inserted in the wrong place. Use *Alt+U* to remove the paragraph, reposition the appropriate cursor, and paste again.

For more information on Blocks, Marks, and Scrap, see "Blocks and marks" and "Scrap" in Chapter 3.

Search and translate

When you edit an existing file, one that may be hundreds of lines long, you probably won't remember the line number you want. You could use *PgUp* and *PgDn* or the mouse to scroll through the buffer until you find the spot you're looking for, but a quicker and more accurate way would be to search for a symbol (word, comment, number, or command).

Press *HomeHome* to return to the top of the window. If you were in a longer file which didn't fit on one screen, pressing *HomeHomeHome* would place the cursor at the top of the file, or buffer.

Search forward

Press *F5* for **Search**. The following prompt appears on the command line:

```
↓ Search for: _
```

Type `All` and press *Enter*. Search is case-sensitive, unless you turn it off. See **Case sensitive toggle** in "Search and translate" in "Command overview" (Chapter 3). The command locates and highlights `All` on line 5 and the cursor moves to the `A`.

Search again

Press *Shift+F5* to continue to search for the same text. The next match on line 16 is found. If there were no more matches, `Pattern not found.` would be displayed on the command line.

Search backward

To search toward the front of the file, press *Alt+F5*. The word `All` is displayed, as BRIEF remembers the last entered text. Press *Enter* and the cursor stays at the `All` on line 16.

Translate forward

Translate lets you search for a pattern and replace it with something else.

Press *HomeHome* to position the cursor at the top of the window. Press *F6* for **Translate**. The following prompt appears:

```
↓ Pattern: All
```

Press *Backspace* three times, then type `a` and press *Enter*. The next prompt is:

```
Replacement: _
```

Type `one` and press *Enter*.

The cursor moves to the first `a` it finds on line 5 and another prompt appears:

```
Change [Yes|No|Global|One]? _
```

Be very careful using the Global change, which changes every `a` in the file to `one`. This would result in words like `onet` (instead of `at`) and `onessign` (instead of `assign`). The way to safeguard against this is to make the search pattern very specific by entering *Space*a*Space*, instead of just `a`.

Since the cursor is at the correct place, enter `y`. The line now reads `Insert one copyright notice....`

The other options at the prompt are

- `n` does not change the match and searches for the next pattern
- `o` changes this occurrence, then returns the cursor to its original position
- *Esc* cancels the command, leaving the cursor at the beginning of the search string

Press *Esc* to cancel the command.

Search and Translate commands are fully explained in the "Search and translate" section of "Command overview" (Chapter 3). It includes information on search forward, backward, and again; translate forward, backward, and again; simple search strings; regular expression characters; searching for words, programming language statements, and elements within matching quotes.

Customizing the tutorial

You have learned the following keystroke commands:

Command	Keystroke	Command	Keystroke
Add/split a line	*Enter*	Help	*Alt+H*
Beginning of line	*Home*	Mark	*Alt+M*
Cancel command	*Esc*	Next word	*Ctrl+→*
Copy to scrap	*Keypad Plus*	Page down	*PgDn*
Cursor movement	Arrows	Page up	*PgUp*
Cut to scrap	*Keypad Minus*	Paste from scrap	*Ins*
Delete character	*Del*	Previous word	*Ctrl+←*
Delete line	*Alt+D*	Search	*F5*
Delete next word	*Alt+Backspace*	Search again	*Shift+F5*
Delete previous word	*Ctrl+Backspace*	Search backward	*Alt+F5*
Delete to beginning of line	*Ctrl+K*	Top of buffer	*HomeHomeHome*
Delete to end of line	*Alt+K*	Top of window	*HomeHome*
End of line	*End*	Translate	*F6*
Exit	*Alt+X*	Undo	*Alt+U*
Go to line	*Alt+G*	Write	*Alt+W*

You have also learned the following mouse commands:

Command	Button	Modifier	Region
Copy to Scrap	Click 2		Window
Cut to Scrap	Click 2	*Ctrl*	Window
Cursor	Click 1		Window
Normal Mark	Drag 1	*Alt* & *Ctrl*	Window
Paste from Scrap	Double-Click 2		Window

The copyright line of code in the text (`Copyright 1990 Borland International`) has to be replaced with your company name and, if desired, the correct year.

You can choose to

- Position the cursor on the B in Borland and delete the company name, then retype it.
- Use Mark to highlight the company name, then delete it or cut to scrap and retype it.
- Use Translate to search for the company name and enter your information as the translation.

Choose one of these methods (or another, if you like) to correct the text.

Compile and run

Now that the macro text is complete, save the file to disk by pressing *Alt+W*. `Write successful.` should appear on the command line.

Next, compile the macro by pressing *Alt+F10*. If any errors occur during compilation, the error message is displayed on the command line and the cursor is moved to the line where the error occurred.

If the file compiled successfully, the message `Compilation successful.` appears on the command line.

If the file did not compile successfully, the error is displayed in the message area and the cursor is positioned on the line with the error. Correct the error, then enter *Ctrl+N* to proceed to the next error. When there are no more errors, a message is displayed in the message area. Recompile the macro.

Modifying your initials macro

If you would like, you can add the **assign_to_key** and **load_macro** statements to your initials macro. Open another window for your initials macro by pressing *F3→*. This window becomes active. Press *Alt+E* to open another file and enter the name of your initials macro (the initials you entered during "Quick installation," followed by `.m`).

Your initials macro appears, which has several parts. The first part is overwritten by setup, so no changes are made here. The second part includes the commented line `Put your changes here.` This area is where you will add the new instructions.

Position the cursor on the first column of `(return)` and press *Enter* to add a blank line. Add the following lines (tab as desired):

```
(assign_to_key "<Ctrl-F1>" "copyrite")
(load_macro "copyrite")
```

Press *Alt+F10* to compile the macro.

Make the **copyrite.cb** window active by pressing *F1←*. Position the cursor in the first column of line 12 and press *Enter* to open a line. Press *Ctrl+F1* to load your copyrite macro and the information you edited earlier appears on line 13.

From now on, you can press *Ctrl+F1* to load your copyright notice into any program.

Press *Alt+X* to return to the prompt. Enter `y` to save the copyright notice or `n` not to.

You have now finished the tutorial. Return to "Complete installation" in Chapter 1 to customize BRIEF to your preference.

C H A P T E R

3

Command overview

This chapter includes all commands in a logical sequence. Each command is fully explained and an overview chart is given at the beginning of each section. The following sections are included in this chapter:

- Help and undo
- Saving and exiting
- Cursor movement
- Mouse
- Editing text
- Buffers
- Blocks and marks
- Scrap
- Windows
- Search and translate
- Macros, playback and remember
- Special commands

Help and undo

These commands get you onscreen or context-sensitive Help text and allow you to undo commands as desired:

Command	Keystroke
Escape	*Esc*
Halt	*Ctrl+Break*
Help	*Alt+H*
Undo	*Alt+U, Keypad **

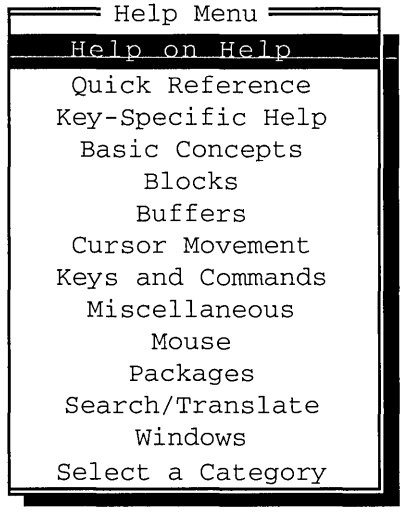
Help

BRIEF's online Help can be obtained:

- Through a general Help menu to select topics
- By specific, context-sensitive Help screens that describe possible responses at a command's prompt

Press *Alt+H* for the **Help** command. The general Help menu appears:

The Help menu



Context-specific Help screens

Sometimes specific help on how to respond to a prompt is needed. Rather than go to the Help menu and select categories, you can get immediate help using this feature. Enter a command. Before executing, press *Alt+H*. The Help screen relative to the command is displayed.

Possible responses

Menu prompt: `Select a Category`

- ↑ and ↓

  Choose a category in the window, which is highlighted.

- *Enter* or double-click *Mouse Button 1*

  Select the current category. Either another menu or the Help screen is displayed.

- *Keypad Minus*

  Return to the previous menu from a sub-category or Help screen.

- *Esc*

  Return to the current buffer.

Within a Help screen

- *PgUp* and *PgDn*

  Display additional pages of help information, if any.

- *Alt+H*

  Display a Help screen that shows the key assignments for the commands being described.

Within Help screens and menus

- *Keypad Minus*

  Return to the previous menu from a sub-category or Help screen.

- *Esc*

  Return to the current buffer. You cannot get help on this key under `Key-Specific Help` from the menu.

Undo

Allows you to undo any command that moves the cursor, adds and deletes text, or marks text, when *Alt+U* or *Keypad ** is pressed. The only commands that cannot be undone are:

- Window commands
- Writing files
- Returning to BRIEF after exiting

Undo can also be used if you want to temporarily look at some code in another section of a file. Move there, examine the code, and then **Undo** until you get back to your original position.

One at a time, you reverse the effect of the last `n` commands (or as many as your memory can hold) by pressing *Alt+U* or *Keypad **. The value of `n` is specified during SETUP, less than or equal to 300. Thirty is the default. If BRIEF runs low on memory, the undo information is purged.

Undoing text entries

When entering text, undo coalesces each time the keystroke is pressed. This means that first the previous word will be deleted, then the space before that word, then the word before that.

The language being used determines how BRIEF defines words. It accounts for punctuation, spaces, and operands. See "Programming support" (Chapter 6) for more information.

For example, if you typed in this code:

```
beginning_of_line();
insert (" Copyright 1992 Borland International\n");
insert (" All rights reserved.\n");
```

and then started pressing *Alt+U* to undo, the lines would be undone in the following manner:

```
");
```

`n`

```
.\
```

```
reserved
```

space after `rights`

```
rights
```

space after `All`

Undoing cursor commands

BRIEF tracks cursor movements, as long as the cursor has been moved at least 10 times (the count is changed if text is entered). The last five moves are undone individually, then the previous five moves are combined into one.

For example, if you move the cursor down 20 lines with ↓, then press *Alt+U*, BRIEF will undo moves 16 to 20 one at a time. Then, it will start to combine moves and will jump back to the tenth move, then the fifth, then the first.

Escape

Cancels commands when there is a displayed prompt, or inserts an escape character at the current position when *Esc* is pressed.

Assigning a different command to *Esc* doesn't change the meaning at a prompt.

Halt

Terminates the following commands when *Ctrl+Break* is pressed:

- **Search forward** (see "Search and translate")
- **Search backward** (see "Search and translate")
- **Translate** (see "Search and translate")
- **Playback** (see "Macros, playback and remember")
- **Execute command** (see "Special commands")

BRIEF attempts to terminate the command and return to the editing session. If the command cannot be terminated without exiting, you receive:

Prompt: `Save files before exiting (Esc to continue)?`

- `y`

  Writes all modified buffers before exiting. If any of the modified buffers are not written properly, BRIEF will not exit.

- `n`

  Exits without changing the files.

- *Esc*

  Continues the previous command.

Saving and exiting

These commands are used to write buffers to files, exit with or without saving, shell out to the operating system, and control backups.

Command	Keystroke
Backup file toggle	*Ctrl+W*
Change output file name	*Alt+O*
Exit	*Alt+X*
Suspend BRIEF	*Alt+Z*
Write	*Alt+W*
Write all and exit	*Ctrl+X*

Backup file toggle

Turns automatic backup on or off inside BRIEF when *Ctrl+W* is pressed.

Turning backups on or off does not alter the amount of space BRIEF needs on a disk to write the file successfully.

Automatic backup can be set on or off from outside BRIEF in two ways: during SETUP or with the **-b** flag. Backup files can be directed to a specific location with the BBACKUP environment variable. The default location is \brief\backup.

See the section "Backup file toggle" in Chapter 5 for information on the **-b** flag.

Change output file

Changes the output file name for the current buffer, which is used when a file is written to disk, when *Alt+O* is pressed. You can change the path as well as the file name; this is a useful command if the current disk you are working on does not have enough space to save the file. The name must differ from any input or output name used for the other buffers in the buffer list.

Prompt: `Enter new output file name:`

Type the new file name for the current buffer.

Suspend BRIEF

Exits temporarily to the operating system when *Alt+Z* is pressed. You can enter any command, including changing the current directory or the current disk drive. To return to BRIEF, type `exit`. You return to your editing session exactly as you left it.

DOS

Warning: This command creates a "shell process" that runs DOS and creates a new copy of `command.com`. Although this shell has a reduced amount of memory, most normal commands can be executed. Please remember that BRIEF cannot perform checks on your actions when you are in this shell. If you delete the files you are editing, or if a program you execute overwrites memory that is not allocated to it by DOS (and used by BRIEF), your editing session may be lost. This command is extremely useful, but please be careful.

Write

Writes either the current buffer or marked block of text to disk when *Alt+W* is pressed. If you write a marked block, you are prompted for a specific file name. When you write a buffer, its status changes to "unmodified."

BRIEF does not support writing column blocks.

Use the **Write** command to save a file without exiting. It is a good idea to save your work occasionally, as you edit, in case of a power failure or similar disaster.

When the current buffer is written to disk, the contents of the previous version are saved in a backup file, which resides in the directory `\brief\backup` unless directed to another location (or turned off) during SETUP.

If you receive the error message "Disk full, write file to another disk.," the write has failed. Execute the **Change Output file name** command (*Alt+O*) to redirect the file to another disk so you can save the changes you made during your editing session.

Because BRIEF always saves your original file until the new version is safely written to disk, you will need enough free space on your disk to store both the old and new versions of the file to successfully write it.

Write all and exit

Writes any modified buffers and exits BRIEF without prompting when *Ctrl+X* is pressed. This command is a shortcut to the **Exit** command (*Alt+X* followed by *w*) when you want to both exit and save all your files.

If any of the modified buffers do not write properly, BRIEF will not exit.

Exit

Exits BRIEF when *Alt+X* is pressed. If there are unwritten buffers, BRIEF displays the number and prompts you for an action.

Prompt: `n buffers have not been saved. Exit [ynw]?`

- `y`

  Exits without changing the files.

- `w`

  Writes all modified buffers before exiting.

- `n`, *Enter*, or *Esc*

  Cancels the command.

This exits BRIEF completely, returning you to the original copy of `command.com` rather than invoking a new copy. If you want to exit temporarily, see the previous entry **Suspend BRIEF**.

Cursor movement

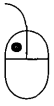
These commands allow you to move the cursor within the window or buffer as desired. The following commands are described:

Command	Keystroke
Back tab	*Shift+Tab*
Beginning of line	*Home*
Cursor movement (up, down, left, right)	↑, ↓, ←, →
End of buffer	*End End End* or *Ctrl+PgDn*
End of line	*End*

Command	Keystroke
End of window	*End End* or *Ctrl+End*
Go to line	*Alt+G*
Left side of window	*Shift+Home*
Next character	→
Next word	*Ctrl+→*
Page down	*PgDn*
Page up	*PgUp*
Previous character	←
Previous word	*Ctrl+←*
Right side of window	*Shift+End*
Scroll buffer down in window	*Ctrl+D*
Scroll buffer up in window	*Ctrl+E*
Tab	*Tab*
Top of buffer	*Home Home Home* or *Ctrl+PgUp*
Top of window	*Home Home* or *Ctrl+Home*

Arrows

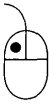
The arrow keys move the cursor in the desired direction. The command names are used here, but the appropriate arrow can also be used.



To use the mouse to position the cursor, place the mouse cursor in the desired area and click *Mouse Button 1*.

Up

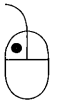
Moves the cursor up one line, staying in the same column, when ↑ is pressed. When the cursor is moved into virtual space, it will change shape if it has been modified in SETUP. See "Cursor choices" under "Complete installation" in Chapter 1 for more information.



To use the mouse to scroll up one line, place the mouse cursor on the up arrow at the top of the vertical scroll bar and click *Mouse Button 1*. See "Mouse commands" for more information.

Down

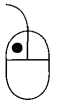
Moves the cursor down one line, retaining the column position, when ↓ is pressed. If the cursor is positioned on the last visible line in the window, the window is scrolled down over the buffer to make the next line visible.



To use the mouse to scroll down one line, place the mouse cursor on the down arrow at the bottom of the vertical scroll bar and click *Mouse Button 1*. See "Mouse commands" for more information.

Left

Moves the cursor one column to the left, remaining on the same line, when ← is pressed. When the cursor is moved into virtual space, it will change shape if it has been modified in SETUP.

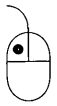


To use the mouse to scroll left one column, place the mouse cursor on the left arrow on the horizontal scroll bar and click *Mouse Button 1*. See "Mouse commands" for more information.

Right

Moves the cursor one column to the right, remaining on the same line, when → is pressed. If the cursor is moved into virtual space, the cursor changes shape.

If this command is executed at the end of a line that exceeds the maximum line length, the cursor moves at most one character past the last character that will be saved if the file is written. If the line is modified, the remaining characters in the line will be truncated, including the character at the cursor.



To use the mouse to scroll right one column, place the mouse cursor on the right arrow on the horizontal scroll bar and click *Mouse Button 1*. See "Mouse commands" for more information.

Next character

Moves the cursor to the next character in the buffer (if not at the end of the buffer), treating tabs as single characters and wrapping around line boundaries, when either → or *F10* **next_char** are pressed or entered.

Previous character

Moves the cursor to the previous character in the buffer (if not at the top of the buffer), treating tabs as single characters and wrapping around line boundaries, when either ← or *F10* **prev_char** are pressed or entered.

Next word

Moves the cursor to the first character of the next word when *Ctrl+→* is pressed. BRIEF varies the search pattern for a word from language to language to account for special characters.

Previous word

Moves the cursor to the first character of the previous word when *Ctrl+←*is pressed. BRIEF varies the search pattern for a word from language to language to account for special characters.

Tab

The way *Tab* works depends on the editing mode being used:

Insert mode inserts a tab into the buffer at the current cursor position, moving the cursor to the next tab stop.

Overstrike mode moves the cursor to the next tab stop.

If a block is marked and indenting is on, all of the lines in the marked area are moved right by one tab stop.

By default, the *Tab* key inserts a tab character. You can specify spaces instead of the tab character by:

- Choosing spaces as the fill character during SETUP
- Using the `-t` flag on the command line, or
- Executing the **Use tab characters toggle** in the current buffer.

If **Tab** inserts a tab character, the cursor will change shape over the tab area, indicating the area is virtual space, not a series of blank spaces.

You can set tab stops with the **Tab stops** command.

For example, if you want to set the first three tab stops to 5, 8, and 11, press *F10*, type `tabs 5 8 11`, then press *Enter*.

All subsequent tabs are three columns wide. BRIEF uses the difference between the last two tab stops (8 and 11) to set the rest. In this example, the next tab stops would be 14, 17, 20...

If only one tab stop is entered, BRIEF uses that number to set all tab stops. For example, if you enter `tabs 5`, the tab stops would be 5, 9, 13...

Back tab

Moves the cursor to the previous tab stop without erasing tabs or characters when *Shift+Tab* is pressed.

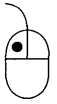
If the file extension of the current buffer is attached to a language package (see SETUP), the **Back tab** command changes behavior slightly. If a block is marked when *Shift+Tab* is pressed, all of the lines in the block are moved left one tab stop. If any lines in the block cannot be moved left without deleting a visible character, they are not moved; all other lines are moved.

Key assignment note

Back tab calls the **slide_out** macro when used to move blocks in an indenting package. See "Programming support" (Chapter 6) for information on language packages.

Beginning of line

Places the cursor at column 1 of the current line when *Home* is pressed.



To use the mouse to go to the beginning of the line, place the mouse cursor on the left arrow on the horizontal scroll bar and double-click *Mouse Button 1*. See "Mouse commands" for more information.

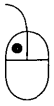
Key assignment note

Home calls the **_home** macro. The key assignment for *Home* can be reassigned, but the reassignment would affect the **Top of buffer** and the **Top of window** commands. The **_home** macro cannot be executed with **Execute command**.

End of line

Places the cursor at the last valid character of the current line when *End* is pressed. In a line with fewer characters than the maximum line length, the last valid character is the newline character.

If this command is executed on a line that exceeds the maximum line length, the cursor moves to the position where the final newline should appear. If the line is modified, the remaining characters in the line will be truncated, including the character at the cursor.



To use the mouse to go to the end of the line, place the mouse cursor on the right arrow on the horizontal scroll bar and double-click *Mouse Button 1*. See "Mouse commands" for more information.

Key assignment note

The *End* keystroke calls the **_end** macro. The key assignment for *End* can be reassigned, but the reassignment will affect the **End of buffer** and the **End of window** commands. The **_end** macro cannot be executed through **Execute command**.

Go to line

Moves the cursor to the specified line number when *Alt+G* is pressed and a number is entered. The status line displays the line number where the cursor is positioned.

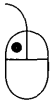
If you specify a line that is greater than the number of lines in the buffer, the cursor moves approximately one screen past the end of the buffer.

Prompt: `Go to line:`

Type the line number and press *Enter*.

Page up

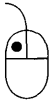
Moves the cursor up one page of text when *PgUp* is pressed, where a page equals the length of the current window.



To use the mouse to scroll up one page, place the mouse cursor on the vertical scroll bar above the scroll box and click *Mouse Button 1*. See "Mouse commands" for more information.

Page down

Moves the cursor down one page of text when *PgDn* is pressed, where a page equals the length of the current window.



To use the mouse to scroll up one page, place the mouse cursor on the vertical scroll bar below the scroll box and click *Mouse Button 1*. See "Mouse commands" for more information.

Top of window

Ctrl+Home moves the cursor to the top line of the current window, retaining the column position. *Home Home* moves the cursor to the top line and the first column of the current window.

Key assignment note

The *Home Home* keystroke calls the **_home** macro. The key assignment for *Home Home* can be reassigned, but the reassignment will affect the **Beginning of line** and the **Top of buffer** commands. The **_home** macro cannot be executed with **Execute command**.

End of window

Ctrl+End moves the cursor to the last line of the current window, retaining the column position. *End End* moves the cursor to the last character in the last line in the current window.

Key assignment note

The *End End* keystroke calls the **_end** macro. The key assignment for *End End* can be reassigned, but the reassignment will affect the **End of buffer** and the **End of line** commands. The **_end** macro cannot be executed through **Execute macro**.

Left side of window

Moves the cursor to the left side of the window when *Shift+Home* is pressed. This command is similar to *Home* (**Beginning of line**), but the cursor stops at the window border if the file has scrolled horizontally.

If the 101-key keyboard driver is loaded, press *Alt+Home*.

Right side of window

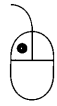
Moves the cursor to the right side of the window when *Shift+End* is pressed, regardless of the length of the line. For example, if the cursor is at line 1, column 1 in a window sized at 5 lines and 15 columns, *Shift+End* causes the cursor to move to column 15 on line 1.

If the 101-key keyboard driver is loaded, press *Alt+End*.

If this command is executed within one window width of the end of a line that exceeds the maximum line length, the cursor moves to the position where the final newline should appear. The remaining characters in the line will be truncated, including the character at the cursor.

Top of buffer

Moves the cursor to the first character of the buffer when *Ctrl+PgUp* or *Home Home Home* are pressed.



To use the mouse to go to the top of the buffer, place the mouse cursor on the up arrow above the vertical scroll bar and double-click *Mouse Button 1*. See "Mouse commands" for more information.

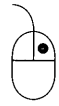
Key assignment note

The *Home Home Home* keystroke calls the **_home** macro. The key assignment for *Home Home Home* can be reassigned, but the reassignment will affect the **Beginning of line** and the **Top of window** commands. The **_home** macro cannot be executed with **Execute command**.

End of buffer

Moves the cursor to the last character in the buffer, which is always a newline character, when *End End End* or *Ctrl+PgDn* is pressed.

If the last line of the file exceeds the maximum line length, the cursor moves to the position where the final newline should appear. If the line is modified, the remaining characters in the line will be truncated, including the character at the cursor.



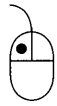
To use the mouse to go to the bottom of the buffer, place the mouse cursor on the down arrow below the vertical scroll bar and double-click *Mouse Button 1*. See "Mouse commands" for more information.

Key assignment note

The *End End End* keystroke calls the **_end** macro. The key assignment for *End End End* can be reassigned, but the reassignment will affect the **End of line** and **End of window** commands. The **_end** macro cannot be executed through **Execute command**.

Scroll buffer up in window

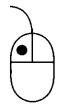
Moves the buffer, if possible, up one line in the window, keeping the cursor on the same line, when *Ctrl+E* is pressed. When the cursor reaches the top of the window, it remains there, but the buffer continues to move up.



To scroll up through the buffer using the mouse, place the mouse cursor on the vertical scroll bar above the scroll box and continuously click *Mouse Button 1* or position the mouse cursor on the scroll box and drag it up while pressing *Mouse Button 1*. See "Mouse commands" for more information.

Scroll buffer down in window

Moves the buffer, if possible, down one line in the window, keeping the cursor on the same line, when *Ctrl+D* is pressed. When the cursor reaches the bottom of the window, it remains there, but the buffer continues to move down.



To scroll down through the buffer using the mouse, place the mouse cursor on the vertical scroll bar below the scroll box and continuously click *Mouse Button 1* or position the mouse cursor on the scroll box and drag it down while pressing *Mouse Button 1*. See "Mouse commands" for more information.

Mouse

The mouse can be used in place of some commands to move the cursor, change windows, mark and extend text, and scroll. It can also be used to select and extend the selection to words, to produce a pop-up menu, to position the file both horizontally and vertically in relation to the scroll box, and to scroll the page left or right.

The following mouse commands described here are also covered in other sections:

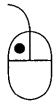
Command	Button	Modifier	Region
Beginning of line	Dbl click 1		Left scroll arrow
Bottom of buffer	Dbl click 1		Down scroll arrow
Change window	Click 1		New window or window title
Column mark	Drag 1	*Ctrl*	Window
Command mode	Click 1		Message area
Copy to scrap	Click 2		Window
Create window	Dbl click 1 or 2		Top/left edge
Cursor	Click 1		Window

Command	Button	Modifier	Region
Cut to scrap	Click 2	*Ctrl*	Window
Delete window	Click 1		Close button
End of line	Dbl click 1		Right scroll arrow
Extend selection	Click 1	*Shift*	Window
Extend word selection	Dbl click 1	*Shift*	Window
Line mark	Drag 1	*Alt*	Window
Noninclusive mark	Drag 1		Window
Normal mark	Drag 1	*Alt Ctrl*	Window
Paste from scrap	Dbl click 2		Window
Pop-up menu	Click 2	*Shift*	Window
Position file left/right	Move 1		Horizontal scroll box
Position file up/down	Move 1		Vertical scroll box
Resize window	Drag 1		Top/left edge
	Drag 2		Any edge
Scroll down—line	Click or hold 1		Down scroll arrow
Scroll down—page	Click or hold 1		Scroll bar below box
Scroll left—column	Click or hold 1		Left scroll arrow
Scroll left—page	Click or hold 1		Scroll bar left of box
Scroll right—column	Click or hold 1		Right scroll arrow
Scroll right—page	Click or hold 1		Scroll bar right of box
Scroll up—line	Click or hold 1		Up scroll arrow
Scroll up—page	Click or hold 1		Scroll bar above box
Select word	Dbl click 1		Window
Top of buffer	Dbl click 1		Up scroll arrow
Zoom/unzoom window	Click 1 Dbl click 1		Zoom button Title bar

The following commands are included only in this section:

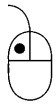
- Select word and extend
- Pop-up menu
- Position file left/right
- Position file up/down

Cursor



To change the position of the text cursor, move the mouse cursor to the position desired and click *Mouse Button 1*.

Command mode



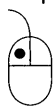
To execute a command, move the mouse cursor to the command line (status message) area and click *Mouse Button 1*.

Scroll

You scroll text by positioning the mouse cursor on either of the scroll bars (right or bottom) or the scroll arrows at the ends of the scroll bars and clicking or holding the mouse button as described below.

You control how quickly you move through the text by clicking or holding the mouse button. Clicking moves the text one unit at a time. Holding still moves the text one unit at a time, but the movement doesn't stop until you release the button.

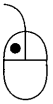
Up



To scroll up one line at a time, position the mouse cursor on the up arrow above the vertical scroll bar and click or hold *Mouse Button 1*.

To scroll up one page at a time, position the mouse cursor on the vertical scroll bar above the scroll box and click or hold *Mouse Button 1*.

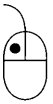
Down



To scroll down one line at a time, position the mouse cursor on the down arrow below the vertical scroll bar and click or hold *Mouse Button 1*.

To scroll down one page at a time, position the mouse cursor on the vertical scroll bar below the scroll box and click or hold *Mouse Button 1*.

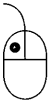
Left



To scroll left one column at a time, position the mouse cursor on the left arrow on the horizontal scroll bar and click or hold *Mouse Button 1*.

To scroll left one page at a time, position the mouse cursor on the horizontal scroll bar to the left of the scroll box and click or hold *Mouse Button 1*.

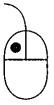
Right



To scroll right one column at a time, position the mouse cursor on the right arrow on the horizontal scroll bar and click or hold *Mouse Button 1*.

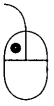
To scroll right one page at a time, position the mouse cursor on the horizontal scroll bar to the right of the scroll box and click or hold *Mouse Button 1*.

Beginning of line



To move the cursor to the beginning of the line, double-click *Mouse Button 1* on the left arrow on the horizontal scroll bar.

End of line



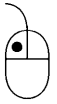
To move the cursor to the end of the line, double-click *Mouse Button 1* on the right arrow on the horizontal scroll bar.

Mark and extend

Text can be marked with the mouse in one of the following ways: normal (any block of text), line, column, and noninclusive (character at the end of the block is not included).

Normal mark and extend

Position the mouse cursor anywhere in the text, and hold *Alt*, *Ctrl*, and *Mouse Button 1* while moving the cursor to the desired position.



To extend the text selection, press *Shift* and *Mouse Button 1*. The highlighted area will move to the mouse cursor position. To continue marking text, drag the mouse or reposition the cursor and extend again.

Line mark and extend

Position the mouse cursor anywhere on the line, hold *Alt* and *Mouse Button 1* and move the cursor to the desired position.

To extend the text selection, press *Shift* and *Mouse Button 1*. The highlighted area will move to the mouse cursor position. To continue marking text, drag the mouse or reposition the cursor and extend again.

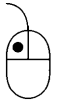
Column mark and extend

Position the mouse cursor anywhere in the column, hold *Ctrl* and *Mouse Button 1* and move the cursor to the desired position.

To extend the text selection, press *Shift* and *Mouse Button 1*. The highlighted area will move to the mouse cursor position. To continue marking text, drag the mouse or reposition the cursor and extend again.

Noninclusive mark and extend

Position the mouse cursor anywhere in the text, hold *Mouse Button 1* and move the cursor to the desired position.



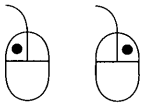
To extend the text selection, press *Shift* and *Mouse Button 1*. The highlighted area will move to the mouse cursor position. To continue marking text, drag the mouse or reposition the cursor and extend again.

Window

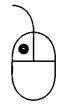
The mouse can be used to modify window placement by creating, deleting, resizing, or changing to another.

Create

Double-click *Mouse Button 1* or *Mouse Button 2* at the left edge (for a horizontal window) or the top edge (for a vertical window).

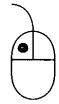


Change



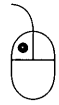
Move the mouse cursor to the window and click *Mouse Button 1* or click on the title of the desired window. Note that this action does not reposition the text cursor.

Delete

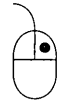


Click *Mouse Button 1* on the close window icon.

Resize

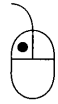


To resize a window from the top or left edge, position the mouse cursor on the top or left window edge to be moved. Hold *Mouse Button 1* to select the edge, then move the mouse until the window is the desired size.



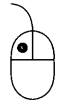
To resize a window from any edge, position the mouse cursor on any window edge to be moved. Hold *Mouse Button 2* to select the edge, then move the mouse until the window is the desired size.

Zoom/unzoom



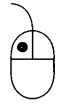
Position the mouse cursor in the zoom button in the top right corner and click *Mouse Button 1* or double-click *Mouse Button 1* in the title bar of the desired window.

Top of buffer



Position the mouse cursor on the up arrow above the vertical scroll bar and double-click *Mouse Button 1.*

Bottom of buffer

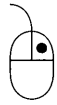


Position the mouse cursor on the down arrow below the vertical scroll bar and double-click *Mouse Button 1.*

Scrap

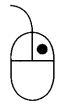
The mouse can be used to copy or cut selected text to the scrap or to paste the scrap into the current buffer.

Copy



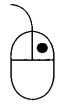
Click *Mouse Button 2* after the text has been marked.

Cut



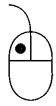
Hold *Ctrl* while clicking *Mouse Button 2* after the text has been marked.

Paste



Position the mouse cursor at the desired position in the text and double-click *Mouse Button 2.*

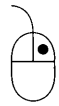
Select word and extend



To select a word, position the cursor anywhere in the word and double-click *Mouse Button 1.*

To extend the selection, position the mouse cursor at the desired word, hold the *Shift* key and double-click *Mouse Button 1.*

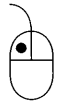
Pop-up menu



This command displays a menu specified in `\brief\help\popup.mnu`, which you can modify.

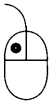
Hold the *Shift* key and click *Mouse Button 2* to display the menu.

Position file left/right



To position the window to the left or right of the current line, position the cursor on the scroll box on the horizontal scroll bar and hold *Mouse Button 1.* Move the mouse to position the scroll box in a new place. That area of the file is displayed.

Position file up/down



To move quickly through the file, position the cursor on the vertical scroll box, hold *Mouse Button 1*, and move the mouse to position the scroll box in a new place. That area of the file is displayed.

Editing text

These commands control insert/overstrike mode and modify existing text. The following commands are described:

Command	Keystroke
Backspace	*Backspace*
Center line horizontally	*F10* **center**
Delete	*Del*
Delete line	*Alt+D*
Delete next word	*Alt+Backspace*
Delete previous word	*Ctrl+Backspace*
Delete to beginning of line	*Ctrl+K*
Delete to end of line	*Alt+K*
Enter	*Enter*
Insert mode toggle	*Alt+I*
Margin	*F10* **margin**
Open line	*Ctrl+Enter*
Quote	*Alt+Q*
Reformat paragraph	*F10* **reform**
Tab stops	*F10* **stops**

Insert mode toggle

Switches between insert mode and overstrike mode when *Alt+I* is pressed. If you don't change the cursor shapes with SETUP, the cursor is thin in insert mode, and it is fat in overstrike mode. When BRIEF is in overstrike mode, characters typed into the buffer overwrite the characters present at the current position. By default, BRIEF starts in insert mode.

See SETUP in "Complete installation" in Chapter 1 for more information on cursor shapes and insert/overstrike mode.

Backspace, Enter, and *Tab* behave differently in insert mode than in overstrike mode. *Backspace* and *Enter* are described in this chapter. *Tab* is described in "Cursor movement."

Enter

Insert mode inserts a newline character at the current position when *Enter* is pressed, placing all following characters onto a newly created next line. The newline character can then be deleted (to join lines) by moving to the end of the first line to be joined and pressing *Del.*

Overstrike mode moves the cursor to the first column of the next line when *Enter* is pressed. No character is inserted. To open up a new line in overstrike mode, press *Ctrl+Enter.* To split the current line, use *Ctrl+M,* which is the ASCII equivalent of the newline sequence.

Open line

Inserts a blank line after the current line and places the cursor on the first column of this new line when *Ctrl+Enter* is pressed. If the cursor is in the middle of an existing line, the line is not split.

Center line horizontally

Centers the text on a line between the first column and the right margin when *F10* **center** is entered.

Quote

Causes the next keystroke to be interpreted literally (that is, not as a command) by inserting an ASCII sequence that would normally be interpreted as a command when *Alt+Q* is pressed.

For example, you might want to insert a literal backspace character (ASCII 8) into your file. Normally, pressing the *Backspace* key would invoke the **Backspace** command. Press *Alt+Q* first, then *Backspace*.

The **Quote** command overrides only those control sequences that have an ASCII equivalent. If a BRIEF command is pressed that does not have an ASCII equivalent (such as *Alt+D*), the keystroke is not overridden by the **Quote** command; the associated command executes as usual.

Margin

Resets the right margin for wordwrap, centering, and paragraph reformatting when *F10* **margin** is entered. The preset margin is at the seventieth character.

Prompt: `Margin:`

Enter a value indicating the column number for the right margin. The left margin is set at the first column and cannot be changed.

You could also press *F10* and enter **margin ##**, where `##` is the new right margin value.

Reformat paragraph

Reformats a paragraph, adjusting it to the current right margin, when *F10* **reform** is entered. The paragraph(s) to be reformatted can either be marked or unmarked. If nothing is marked, the current paragraph is reformatted. Otherwise, the marked areas are reformatted.

After you enter the **reform** command, you will notice a short delay while BRIEF adjusts the paragraph.

Tab stops

Sets the tab stops for the current buffer when *F10* **tabs** is entered.

For most file extensions, tabs are preset in BRIEF to 9, 17, 25, 33, and so on. You can change these defaults with SETUP.

Prompt: `Enter tab stop (return terminates):`

Type the new tab stop and press *Enter*. To terminate the **Tab stops** command, press *Enter* without entering a number. The last tab distance specified repeats for the rest of the line.

Tab characters are displayed as blanks. For example, if the first tab stop is set to 10, pressing the *Tab* key in column 1 moves the cursor to column 10. If you have specified that the tab character is inserted, there is actually only one character stored. If you have specified that spaces are inserted, the *Tab* key will insert nine spaces.

See **Tab** in "Cursor movement" for more information.

Backspace

Backspaces and erases the character preceding the cursor when *Backspace* is pressed. When the cursor is at the upper left edge of the window, the backspace is not performed to avoid erasing characters not shown on the screen.

Insert mode moves the remainder of the line after the erased character left to close the space.

Overstrike mode replaces the erased character by a space, unless the character is a tab. If there is a tab, the cursor moves to the left, as it does when ← is pressed. If the previous character to the left of the cursor is a newline, **backspace** moves the cursor to the left, but does not delete the newline character.

Key assignment note **Backspace** calls the **self_insert** macro, which in turn calls **backspace**.

Delete

Deletes the character at the cursor or, if a block is marked, deletes (and unmarks) the marked block when *Del* is pressed. If a block is deleted, the cursor is left at the character immediately following the block. **Delete** also functions as a join command, combining lines by deleting the newline character. When the cursor is at the end of a line, the newline character is deleted and the next line moves to the end of the current line.

BRIEF does *not* copy deleted blocks to the scrap. For this, use the **Cut to scrap** command.

Any deleted text can be recovered with the **Undo** command.

Delete next word

Deletes from the cursor position to the start of the next word when *Alt+Backspace* is pressed, if the 101-key keyboard driver is loaded. If the cursor is in the middle of a word, all of the characters from the cursor position to the beginning of the next word are deleted. The characters in front of the cursor are not deleted.

BRIEF varies the search pattern for a word from language to language to account for each language's special characters.

Delete previous word

Deletes from the cursor position to the beginning of the previous word when *Ctrl+Backspace* is pressed. If the cursor is in the middle of a word, the characters in front of the cursor are deleted, up to the beginning of the current word. The characters after the cursor are not deleted.

BRIEF varies the search pattern for a word from language to language to account for each language's special characters.

Delete line

Deletes the entire current line, regardless of the column position of the cursor, when *Alt+D* is pressed. The next line takes its place, and the cursor remains at the same screen location.

Delete to end of line

Deletes all characters from the current position to the end of the line, when *Alt+K* is pressed. If the cursor is at the beginning of a line, the entire line is erased except for the newline character. **Delete to end of line** never deletes the newline character.

Delete to beginning of line

Deletes all characters before the cursor to the beginning of the line when *Ctrl+K* is pressed. If the cursor is beyond the end of the line, the entire line is deleted, including the newline character.

These commands control the multiple editing buffers that BRIEF allows. To start BRIEF with multiple buffers, type the following at the prompt:

```
b filenam1.ext file name2.ext...filenamX.ext
```

BRIEF will look for the files in the directories. The first file on the list will be displayed. If any new file names are entered, the following prompt appears:

```
New file (unable to open filenamX.ext)
```

If you started BRIEF with only one file, but want to add more to the buffer list, use **Edit file** to load it.

The following commands are described:

Command	Keystroke
Buffer list	*Alt+B*
Compile buffer	*Alt+F10*
Delete current buffer	*Ctrl+Minus*
Display file name	*Alt+F*
Edit file	*Alt+E*
Next buffer	*Alt+N*
Previous buffer	*Alt+Minus*
Read file into buffer	*Alt+R*

Buffer list

Displays BRIEF's buffer list when *Alt+B* is pressed. The following screen appears:

Sample buffer list

```
Buffer List
1)  Buffer copyrite.cb has not been modified.
File:  c:\brief\copyrite.cb
2)  Buffer sample.doc has not been modified.
File:  c:\brief\sample.doc
3)  Buffer list.doc has not been modified.
File:  a:\list.doc
```

You can move the cursor through this list to delete, write, and edit buffers. Files that have been modified but not saved are labeled with an asterisk (*).

The current buffer is the first one on the list.

The following keystrokes are used to manipulate the buffer list:

- ↑ and ↓

  Arrow keys allow you to select buffers in the buffer list. The selected buffer is highlighted.

- *PgDn*

  Moves down one page in the buffer list, where a page equals six entries.

- *PgUp*

  Moves up one page in the buffer list, where a page equals six entries.

- *Home*

  Moves to the first entry in the buffer list.

- *End*

  Moves to the last entry in the buffer list.

- *E, Enter* or *Alt+E*

  Edits the selected buffer, placing it in the current window.

- *D* or *Ctrl+→*

  Deletes the selected buffer, unless it is being viewed in a window. If the buffer has been modified, a prompt appears to verify that you want to delete the modified buffer:

  ```
  This buffer has not been saved.  Delete [ynw]?
  ```

  For details on responses, see **Delete current buffer.**

- *W* or *Alt+W*

  Writes the selected buffer to disk. When you write a buffer, its status changes to "unmodified."

- *Esc*

  Exits the buffer list and returns you to the current buffer.

Buffer lists are created with the **buf_list** macro, which has both a standard and an expert mode. The standard mode is described above. Expert mode displays a one-line entry for each buffer, which contains only the full pathname of the file. If the file has been modified and not saved, an asterisk (*) is displayed after it. To run **buf_list** in expert mode, add the following to your initials macro:

```
(assign_to_key "<Alt-B>" "buf_list 1")
```

Read file into buffer

Reads a copy of the specified file into the current buffer, inserting it immediately before the current position, when *Alt+R* is pressed.

The file must be either a file that exists on disk or a file from the buffer list. If the file is in the buffer list, BRIEF reads the copy currently in memory (including modifications made during the editing session). If the file is read from disk, it remains in your directory in its original form, and the file name is not placed into the buffer list. You cannot read a file into itself.

Prompt: `File to read:`

Type the name of a file on disk or a file currently in the buffer list and press *Enter*.

If you frequently use a file as a base (or template) for new files, use the following procedure:

- Edit the template file with the **Edit file** command, assigned to *Alt+E*.
- Make changes to the template file.
- Rename the output file with **Change output file**, assigned to *Alt+O*.
- Write the file.

Do *not* use the **Read file into buffer** command to edit, save, or rename files. It is used to copy text from one file into another.

Display file name

Displays the name of the file associated with the current buffer on the status line when *Alt+F* is pressed. The name of the file includes the full file specification: the drive, path, file name, and extension. If the buffer has been modified, an asterisk appears after the name, as it does in the buffer list.

Edit file

Displays the specified file in the current window when *Alt+E* is pressed. When you specify the file name, BRIEF looks first to see if the file is already in the buffer list. If so, that buffer is viewed in the current window. If no buffer exists with the specified file name, BRIEF searches for the file on disk. If the file is found, it is displayed. If no file is found, BRIEF creates a new, empty file. A specified file that is not currently in the buffer list is added to the end of the list.

The cursor is placed at the beginning of the new buffer. The previous buffer remains in the same position in the buffer list. You can specify a full file specification, including disk, directory, and file name.

Prompt: `File:`

Type the file name and press *Enter*.

Pressing *Tab* at this prompt after entering a partial file name will perform file name completion on the current response. Inappropriate files, such as those with the extensions `.exe`, `.com`, `.cm` and `.obj`, are excluded from the list.

Next buffer

Moves the next buffer in the buffer list, if one exists, into the current window, when *Alt+N* is pressed. This window becomes the current buffer and the last remembered position becomes the current position. If there is only one buffer in the buffer list, BRIEF displays the message `No other buffers`.

Previous buffer

Displays the previous buffer in the buffer list in the current window when *Alt+Minus* is pressed. If there is only one buffer in the buffer list, BRIEF displays the message `No other buffers`.

Delete current buffer

Deletes the current buffer and makes the next buffer in the list the current buffer when *Ctrl+Minus* is pressed. If this buffer has been modified, you are asked if you want to delete it.

Prompt (if buffer is modified but not saved): `This buffer has not been saved.  Delete [ynw]?`

- `y`

  Deletes the buffer.

- `w`

  Writes it and then deletes it.

- `n`, *Enter*, or *Esc*

  Cancels the command.

You cannot delete a buffer if there is no other buffer to take its place.

Compile buffer

Compiles the file in the current buffer (and loads it if it's a BRIEF macro file) when *Alt+F10* is pressed. If any errors occurred during compilation, an error message is displayed and the cursor is moved to the line where the error occurred.

The file extension of the file in the current buffer must be associated with a language support package. Packages can be attached to file extensions during SETUP or by using the BCxxx environment variable. For more information, see "Programming support" (Chapter 6).

Blocks and marks

These commands mark sections of text, called blocks, so they can be used with other commands. The following commands are described:

Command	Keystroke
Column mark	*Alt+C*
Drop bookmark	*Alt+1* through *Alt+0*
Indent block	*Tab*
Jump bookmark	*Alt+J*
Line mark	*Alt+L*
Lowercase block	*F10* **tolower**
Mark/unmark	*Alt+M*
Noninclusive mark	*Alt+A*

Command	Keystroke
Outdent block	*Shift+Tab*
Print block	*Alt+P*
Swap cursor and mark	*F10* **swap_anchor**
Uppercase block	*F10* **toupper**

Mark and unmark

To mark a block, press *Alt+M*. The current cursor position becomes the first character in the block. As you move the cursor, the text is selected and highlighted as part of the block. To unmark the block without executing a command, press *Alt+M* again.

When a block of text is marked, several BRIEF commands can act on the entire block:

- **Cut to scrap**
- **Copy to scrap**
- **Delete**
- **Indent block** (in files with programming support)
- **Lowercase block**
- **Outdent block** (in files with programming support)
- **Print**
- **Search forward**, **Search backward**, and **Search again** (optionally; see the **Block search toggle** command)
- **Translate forward**, **Translate back**, and **Translate again**
- **Uppercase block**
- **Write**

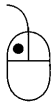
When the **Cut to scrap**, **Copy to scrap**, **Delete**, **Print**, or **Write** commands are executed on a block, the block becomes unmarked.

To mark text with a mouse, position the mouse cursor, hold *Alt+*, *Ctrl*, and *Mouse Button 1*, and move the mouse to the desired position.

To extend the text selection, press *Shift* and *Mouse Button 1*. The highlighted area will move to the mouse cursor position. To continue marking text, drag the mouse or reposition the cursor and extend again.

Line mark

Starts marking a line at a time when *Alt+L* is pressed.



To use the mouse to mark a line, hold *Alt* and *Mouse Button 1* and move the mouse to the desired position.

To extend the text selection, press *Shift* and *Mouse Button 1*. The highlighted area will move to the mouse cursor position. To continue marking text, drag the mouse or reposition the cursor and extend again.

Column mark

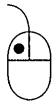
Starts marking text by column when *Alt+C* is pressed.

To use the mouse to mark a column, press *Ctrl* and *Mouse Button 1* while moving the mouse.

To extend the text selection, press *Shift* and *Mouse Button 1*. The highlighted area will move to the mouse cursor position. To continue marking text, drag the mouse or reposition the cursor and extend again.

Noninclusive mark

Equivalent to **Mark**, except that the marked area does not include the character at the end of the block, when *Alt+A* is pressed.



To do a non-inclusive mark with the mouse, press *Mouse Button 1* while moving the cursor to the desired position.

To extend the text selection, press *Shift* and *Mouse Button 1*. The highlighted area will move to the mouse cursor position. To continue marking text, drag the mouse or reposition the cursor and extend again.

Swap cursor and mark

Exchanges the current cursor position with the mark when *F10* **swap_anchor** is entered. For example, when *Alt+M* is pressed, the location where it is pressed becomes the mark. When the cursor moves through the text, the characters between the mark and the current cursor position become highlighted, or marked. When **Swap cursor and mark** is executed, the current cursor position becomes the mark and the position of the mark becomes the position of the cursor.

Indent block

When indenting is on (see Chapter 6) and a block is marked, the *Tab* key indents all the lines in the block to the next tab stop.

Outdent block

When indenting is on (see Chapter 6) and a block is marked, pressing *Shift+Tab* outdents all the lines in the block to the next tab stop.

Lowercase block

Converts the characters in a marked block to lowercase when *F10* **tolower** is entered. If no block is marked, the current line is converted.

Uppercase block

Converts the characters in a marked block to uppercase when *F10* **toupper** is entered. If no block is marked, the current line is converted.

Print block

Prints the marked block on the standard printer device when *Alt+P* is pressed. See **Mark**, **Column mark**, **Line mark**, and **Noninclusive mark** for information on marking blocks.

Tab characters in the marked region are expanded to the appropriate number of spaces before sending them to the printer. All other characters are sent without modification.

Drop bookmark

Bookmarks are markers you can insert in your files to quickly find specific areas in the text. Up to 10 can be entered in any edit session and they can be entered in any buffer. The areas are marked with **drop_bookmark** and returned to with **jump_bookmark**. If you are using the **restore** feature, all bookmarks are remembered when you return to the editing session. See "Using special features" (Chapter 4) for more information on **restore**.

Drop bookmark inserts a numbered bookmark at the current position. The bookmarks are numbered from one to ten: bookmark number one is dropped by pressing *Alt+1*, bookmark two by pressing *Alt+2*, etc. These bookmarks can be returned to with the **Jump to bookmark** command.

Prompt: `Drop bookmark [1-10]:`

Type the number of the bookmark you want to drop and press *Enter*.

Jump to bookmark

Moves the cursor to the specified bookmark (inserted by the **drop_bookmark** command) when *Alt+J* is pressed. If the bookmark is not in the current buffer, BRIEF checks to see if the bookmark's buffer is in any window on the screen. If so, that window is selected. If the bookmark's buffer is not in a window, the buffer is attached to the current window.

Prompt: `Go to bookmark [1-10]:`

Pressing *Tab* at this prompt will bring up a menu of the bookmarks you've already dropped. Bookmarks are saved by the **restore** macro; if you re-edit a file that had bookmarks in it, those bookmarks will be preserved.

Scrap

The scrap is a special buffer used for moving or copying blocks of text. A scrap buffer differs from a regular buffer because:

- Blocks of text can only be copied into it or pasted from it. The text itself cannot be edited (except through macros).
- It is automatically created whenever you cut or copy text.

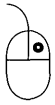
The following commands are described:

Command	Keystroke
Copy to scrap	*Keypad Plus*
Cut to scrap	*Keypad Minus*
Paste from scrap	*Ins*

Copy to scrap

Copies the block of marked characters (selected by pressing *Alt+M, Alt+C, Alt+A,* or *Alt+L* and highlighting the block with arrow keys or commands) to the scrap when *Keypad Plus* is pressed, replacing the contents of the scrap buffer and unmarking the block. If no block is marked, the entire line that the cursor is positioned on is copied.

The current position remains unchanged. If you **Undo** a copy to scrap, the previous scrap is restored.



To copy the block with the mouse, click *Mouse Button 2* while in the window.

Cut to scrap

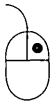
Deletes the block of marked characters to the scrap when *Keypad Minus* is pressed, replacing the previous contents of the scrap. If no block is marked, the current line is cut to scrap. If you **Undo** a cut to scrap, the previous scrap is restored.

To delete the block with the mouse, hold *Ctrl* while clicking *Mouse Button 2* in the text area.

Paste from scrap

Inserts (pastes) the current scrap buffer into the current buffer immediately before the current position when *Ins* is pressed, taking the type of the copied or cut block into account. The cursor remains at the same position, located after the inserted text.

The contents of the scrap consist of the most recently cut or copied text.



To paste text using the mouse, position the mouse cursor at the desired position in the text. Double-click *Mouse Button 2.*

Windows

BRIEF allows you to use multiple tiled windows when creating and editing files. Windows cannot overlap; your screen is subdivided as you request a new window. The minimum size of a window with borders on is 1 line deep by 15 columns wide.

The following commands are described:

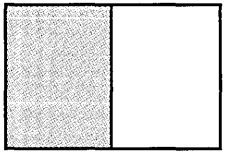
Command	Keystroke
Border toggle	*Alt+F1*
Center line in window	*Ctrl+C*
Change window	*F1+Arrow*
Create window	*F3+Arrow*
Delete window	*F4+Arrow*
Line to bottom of window	*Ctrl+B*
Line to top of window	*Ctrl+T*
Quick window switch	*Shift+Arrow,Alt+Arrow*
Resize window	*F2+Arrow*
Zoom window toggle	*Alt+F2*

Create window

Splits the current window in half either horizontally or vertically when *F3* is pressed, providing two views of the current buffer. The current position is located in the newly created window, making it the current window.

Prompt: `Select side for new window (use cursor keys).`

To create the window, press one of these keys:

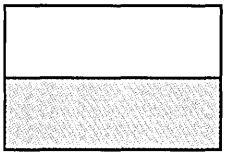


- ←

  Splits the current window, creating a vertical window to the left.

- →

  Splits the current window, creating a vertical window to the right.

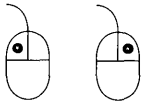


- ↑

  Splits the current window, creating a horizontal window above.

- ↓

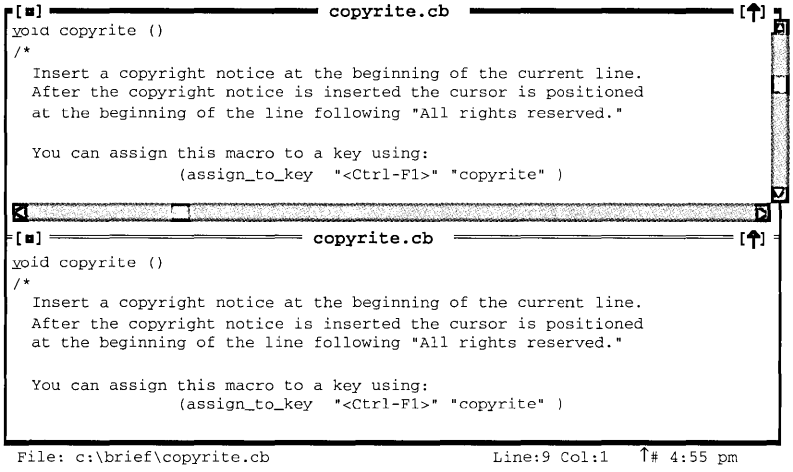
  Splits the current window, creating a horizontal window below.



To create a new window with the mouse, double-click *Mouse Button 1* or *Mouse Button 2* at the left edge (for a horizontal window) or the top edge (for a vertical window). The prompts will not appear.

When you create a new window, each one has a close button in the upper left-hand corner and a zoom button in the upper right-hand corner. The active window will display the scroll bars.

Split screen



These icons are used to delete the window (or close it) and to change the window back to normal size.

Border toggle

Toggles whether or not window borders are displayed when *Alt+F1* is pressed.

You can control the initial border setting two ways:

- During SETUP specify how you want window borders to be displayed.
- The `-Bnum` flag toggles the current default from the command line or from BFLAGS. See "Flags and variables" (Chapter 5) for more information on the `-Bnum` flag.

When borders are off, there is no visible distinction between the various windows on the screen, except that the color of each window is different. If you are using a monochrome display, this command is not useful for more than two windows.

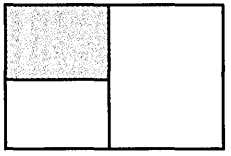
If you are using a mouse, borders must be on to take advantage of scrolling and window creation.

Change window

Initiates a switch from one window to another when *F1* is pressed.

Prompt: `Point to destination (use cursor keys).`

To complete the switch, press one of the arrow keys:

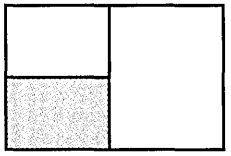


- ←

  Moves to the window sharing the left edge of the current window.

- →

  Moves to the window sharing the right edge of the current window.

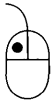


- ↑

  Moves to the window sharing the top edge of the current window.

- ↓

  Moves to the window sharing the bottom edge of the current window.



To position the cursor in a different window using the mouse, move the cursor to either the window area or the title bar and click *Mouse Button 1*. This activates the new window, but does not position the text cursor.

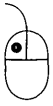
Instead of using *F1* and responding to the prompt, you can execute the **Change window** command faster by using **Quick window switch**.

Quick window switch

To quickly change windows, press *Shift+Arrow*, choosing the arrow key that points to the window you want.

If you have the 101-key keyboard driver loaded (see the section "Device driver flags" in Chapter 5), you must use *Alt+Dedicated Arrow* in place of *Shift+Arrow* to change windows.

The cursor returns to the last remembered position (the position it was in before it left the window). If you point in a direction where there is no window (for example, the editor's border), BRIEF displays the message `No window there.` You must point to a window edge that has been created during the editing session.



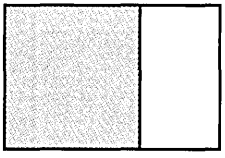
To position the cursor in a different window using the mouse, move the cursor to either the window area or the title bar and click *Mouse Button 1*. This activates the new window, but does not position the text cursor.

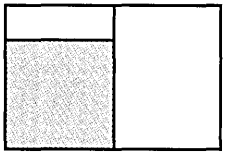
Resize window

Changes the dimension of a window by moving the window's edge when *F2* is pressed. The minimum size of a window is 15 characters wide and one line deep. The edge to be resized must be shared by two, and only two, windows.

Prompt: `Select an edge to move (use cursor keys).`

Point to the edge you wish to move.



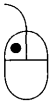


- ← Moves the left edge.
- → Moves the right edge.
- ↑ Moves the top edge.
- ↓ Moves the bottom edge.

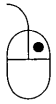
After you select an edge to move, the following prompt appears:

```
Move to new position and press Enter.
```

After you have selected the edge, you can make the window larger by moving the cursor outside the edge and pressing *Enter*. You can make the window smaller by moving the cursor further into the window and pressing *Enter*.



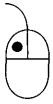
To resize a window from the top or left edge, position the mouse cursor on the top or left window edge to be moved. Hold *Mouse Button 1* to select the edge, then move the mouse until the window is the desired size.



To resize a window from any edge, position the mouse cursor on any window edge to be moved. Hold *Mouse Button 2* to select the edge, then move the mouse until the window is the desired size.

Zoom window toggle

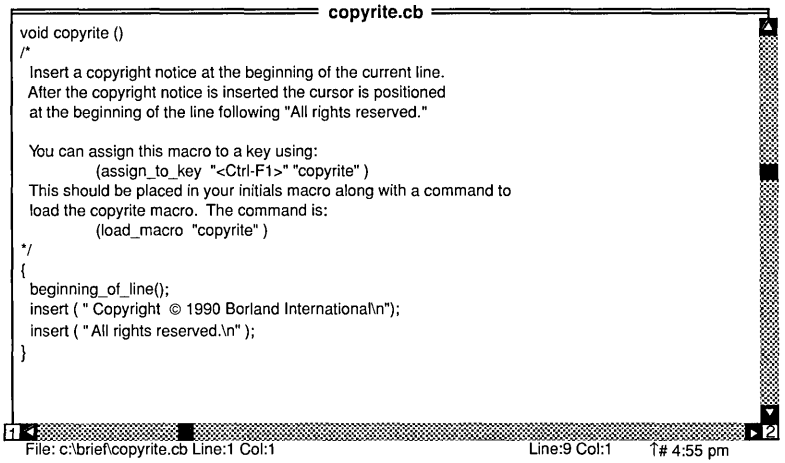
If there is more than one window on the screen, pressing *Alt+F2* or *Ctrl+Z* will enlarge the current window to a full screen window and save the previous window configuration. Pressing the keys again will restore the previous configuration if only one window is on the screen.



To enlarge the current window by using the mouse, position the mouse cursor in the top right corner on the zoom button and click *Mouse Button 1* or position the cursor on the title bar and double-click *Mouse Button 1*.

After you have used the zoom command, the window appears as

Zoomed window screen



The close icon disappears, since it only works when there is more than one window on the screen. The zoom icon changes shape. When you are finished with this screen, click on the zoom icon to return to the original split screen.

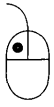
Delete window

Deletes a window when *F4* is pressed. The edge to be deleted must be shared by two, and only two, windows.

Prompt: `Select window edge to delete (use cursor keys).`

To delete the edge, press one of these keys:

- ←
  Deletes the window to the left.
- →
  Deletes the window to the right.
- ↑
  Deletes the window to the top.
- ↓
  Deletes the window to the bottom.



To delete a window with the mouse, double-click *Mouse Button 1* on the window's close button. The prompts will not appear.

Center line in window

Moves the current line, if possible, to the center (middle line) of the current window when *Ctrl+C* is pressed. This only affects the display; the line is not moved within the file.

Line to bottom of window

Scrolls the buffer by moving the current line, if possible, to the bottom of the window when *Ctrl+B* is pressed.

Line to top of window

Scrolls the buffer by moving the current line to the top of the current window when *Ctrl+T* is pressed.

Search and translate

Search commands locate occurrences of specific text patterns in the buffer. The translate commands search first, then replace the located text with new text.

The following commands are described:

Command	Keystroke
Block search toggle	*F10* **block_search**
Case sensitivity toggle	*Ctrl+F5*
Incremental search	*F10* **i_search**
Regular expressions toggle	*Ctrl+F6*
Search again	*Shift+F5*
Search backward	*Alt+F5*
Search forward	*F5, Alt+S*
Translate again	*Shift+F6*
Translate backward	*Alt+F6*
Translate forward	*F6, Alt+T*

Meaning	Regular expression
Cursor positioning	`\c`
Group expressions for searches	`{ }`
Match beginning of line	`<, %`
Match either preceding or following character	`|`
Match end of line	`>, $`
Match newline character	`\n`
Match one or more occurrences of previous character	`+`
Match tab character	`\t`
Match zero or more occurrences of previous character	`@`
Matches one character or range	`[ ]`
Specify range to search for	`[- ]`
Specify characters not to match	`[~ ]`
Numbers groups in search expressions	`\num`

Meaning	Regular expression
Override regular expressions	\
Specify numeric ASCII value	\x*hex number*
Wildcard matching—one character	?
Wildcard matching—zero or more characters	*

Search forward

Searches forward from the current position to the end of the current buffer for the given pattern after *F5* or *Alt+S* is pressed. If the pattern is found, the current position is set to the beginning of the matched string. If no pattern is specified, the previous pattern (displayed on the status line) is used. If located, the matched text is highlighted until a key is pressed.

Specify the pattern to search for.

Prompt: ↓ `Search for:`

When the pattern finds a match, the cursor moves to the first character of the match.

Message (when the search is successful): `Search completed.`

If no match is found, the cursor does not move.

Message (when the search is unsuccessful): `Pattern not found.`

Search backward

Searches backward from the current position to the beginning of the current buffer for the given pattern after *Alt+F5* is pressed. If the pattern is found, the current position is set to the beginning of the matched string. If no pattern is specified, the previous pattern (displayed on the status line) is used. If located, the matched text is highlighted until a key is pressed.

Prompt: ↑ `Search for:`

Enter the specific search pattern. For information on search patterns and regular expressions, see "About search and translate" later in this chapter.

Search again

Searches either forward or backward for the last given pattern, depending on the direction of the previous search, when *Shift+F5* is pressed. Before searching, the current position is moved either forward or backward one character in the buffer to prevent the command from finding the same match. If located, the matched text is highlighted until a key is pressed.

Incremental search

Searches for the specified search pattern incrementally, that is, as you type it, when *F10* **i_search** is entered. It ignores regular expression characters in the search pattern.

Block search toggle

Toggles whether or not **Search forward**, **Search back**, and **Search again** are restricted to blocks when *F10* **block_search** is entered. If block searching is on, search operations only search within the currently marked block. A forward search proceeds from the top of the block to the end, and a backward search goes from the bottom of the block to the top. If block searching is off, all search operations proceed from the current position to the end (or top, in the case of a backward search) of the file. By default, block searching is off when BRIEF is invoked. Block translate, however, is always on.

Case sensitivity toggle

Toggles upper and lowercase sensitivity when *Ctrl+F5* is pressed. If case sensitivity is off, the string `UPPERCASE` will match `UPPERCASE`, `Uppercase`, `uppercase`, and so on. If default startup values are chosen during SETUP, case sensitivity is on.

Regular expressions toggle

Toggles whether or not regular expressions are recognized in patterns by pressing *Ctrl+F6*. If regular expressions are on, certain characters have special meanings and must be preceded by an override character (\) if you want to search for them literally. If regular expressions are off, no characters have special meaning. By default, SETUP turns regular expressions on when BRIEF is invoked. When regular expressions are off, the search prompt contains the string (`RE off`).

Translate forward

Searches for the specified pattern from the current position to the end of the buffer, replacing it with the given string, when *F6* or *Alt+T* is pressed. If a block is marked, the Translate operation is restricted to the marked area.

Prompt 1: ↓ `Pattern:`

Type the string you are searching for and press *Enter*.

Prompt 2: `Replacement:`

Type the replacement for the string and press *Enter*. The cursor moves to the first match, if one exists.

Prompt 3 (when the match is successful): `Change [Yes|No|Global|One]?`

You can respond with

- `y`

  Changes the string at the cursor and searches for the next string.

- `n`

  Does not change the string at the cursor and searches for the next string.

- `g`

  Changes all subsequent matches in the file without prompting. The screen is refreshed every two seconds, allowing you to observe the progress through the file.

- `o`

  Changes this occurrence, then stops the translation and returns the cursor to its original position.

- *Esc*

  Cancels the **Translate** command from the prompt.

After all matches are translated, the following appears:

```
Translation complete, n occurrences changed.
```

If the match is unsuccessful, you'll see the following message:

```
Pattern not found.
```

Patterns are remembered and appear on the command line the next time you use **Translate**. The previous patterns are used by pressing *Enter* at the prompts. To cancel the command at a prompt, press *Esc*.

Translate backward

Searches for the specified pattern from the current position to the beginning of the buffer, replacing it with the given string, when *Alt+F6* is pressed. If a block is marked, the Translate operation is restricted to the marked area. For prompts and responses, see **Translate forward**.

Translate again

Searches again for the specified pattern in the direction of the previous **Translate** command, replacing it with the given string, when *Shift+F6* is pressed. If a block is marked, the Translate operation is restricted to the marked area.

About search and translate

The **Search** and **Translate** commands are case-sensitive. When case sensitivity is on, the matched text must exactly match the pattern specified at the prompt, including upper or lowercase. When case sensitivity is off, the match can consist of the same letters in either upper or lowercase. The default in SETUP is ON.

Regular expressions are special characters that provide flexibility in the pattern. The following examples assume case sensitivity is ON.

Simple search strings

The following sentence provides numerous opportunities for searching:

```
Today, she sells sea shells by the sea shore.
```

Type the sentence and move the cursor to the top of the buffer. If *F5* or *Alt+S* is pressed, you receive a prompt:

```
↓ Search for: _
```

To specify a search pattern, type in the pattern and press *Enter*.

```
↓ Search for: she_
```

The cursor moves to the first letter in the second word, `she`. The message line changes to `Search completed`.

To search again, press *Shift+F5*. The cursor moves to the first letter of the beginning of the word `shells`. The pattern, `she`, is matched by the word `she` and the first three letters of the word `shells`.

Block searching

Normally, the search commands ignore marked areas. By default, a search proceeds from the cursor position to the beginning (or end) of the file.

This behavior can be changed with the **Block search toggle** command. When block searching is on, the cursor position is ignored when a search is performed. Instead, forward searches start at the top of the block and move toward the end; backward searches move from the bottom of the block to the top.

The setting of the **Block search toggle** command is remembered by the **restore** macro.

The search pattern

A pattern is a text string that represents character sequences in a buffer. When you use the search commands, you need to specify a pattern to complete the search. Patterns can be simple or complex; a simple pattern is merely a character, such as `x`, or a string of characters, such as `xyz`. Complex patterns use regular expressions, which are special characters that extend the power of the search commands and allow you to perform complex searches and translations.

Some regular expressions are symbols that represent a character or a group of characters that may vary. Other regular expressions group characters or match special places in buffers (like the beginning and end of a line).

Most DOS users are familiar with one type of regular expression, the wildcard. A wildcard substitutes for any character that could possibly occupy that position. BRIEF uses the two wildcards familiar to DOS users, `?` and `*`. In the example above, you could have pressed *F5* (or *Alt+S*) and specified:

```
↓ Search for: s*e_
```

The first match of this pattern would still be `she`. If *Shift+F5* is pressed, the second match would be the `se` in `sells`.

The `*` in `s*e` finds every occurrence of `s` followed by `e`, regardless of any intermediate characters. In the preceding example, `s*e` would locate the following matches:

- `she` (from the word `she`)
- `se` (from the word `sells`)
- `s se` (from `sells sea`)
- `se` (from `sea`)
- `she` (from `shells`)

- `s by the` (from `shells by the`)
- `se` (from `sea`)
- `shore` (from `shore`)

Features of regular expressions

Other regular expressions allow you to

- Specify a set or range of characters in the pattern
- Specify a character or range of characters to exclude during the search
- Locate invisible characters, such as tab and newline

Regular expressions are powerful and, when used correctly, extremely useful. Each described expression is followed by simple examples that help you understand the purpose of the expression. In the section that follows the descriptions, the expressions are grouped together to illustrate more complex patterns.

Tips on using regular expressions

Regular expressions are the tools you need for precise searching, but you must determine the exact pattern you're searching for. If you want to search for "all of the occurrences of the letter `x` when used as a variable within a programming statement, but not when used as an element of a mathematical equation or as part of a word," you've got some decisions to make about the patterns that you are (and are not) searching for.

After listing the regular expression characters, the practice sessions provide opportunities to use the regular expressions, combine them, and recognize patterns in text.

Regular expression characters

Wildcard matching

? matches any single character. `?` does not match the newline character, so it will not match across lines (see "Matching a newline or a tab" to match the newline character).

Example: `t?p` matches `top` and `tip`, but not `trap`

The word `trap` is not matched because more than one character separates the `t` and `p`.

* matches any number of occurrences (0 or more) of any character except a newline. This character will not match across lines. Note that * will match as *few* occurrences as are necessary to make the rest of the pattern match—even zero.

Example: `t*o` finds matches in `too`, `trowel` and `sporty orange`

but not

```
smart
oranges
```

The `t` and `o` of `smart oranges` are not matched because they are on different lines, separated by a newline character.

Moving the cursor

\c positions the cursor at the character immediately following the `\c` in the pattern or replacement. If more than one `\c` is encountered in the pattern, the last one matched is used.

Example 1: `Response=\c`

Finds the phrase `Response=` and places the cursor at the first character after the equal sign.

If you do not use `\c` to specifically move the cursor to a position in the located string, the cursor is placed at the first matched character in the located string.

Example 2: `sho*\ce`

In the seashells example, this pattern locates `shore` and places the cursor at the `e`.

`\c` can also be used in a replacement string. Normally the cursor is left at the end of the replacement text, but if `\c` is used in the replacement string, the cursor is left at the character following the `\c`.

Specifying a numeric ASCII value

\xhex number: Most ASCII characters can be entered with ease directly from the keyboard. However, with some characters it may be preferable to specify a numeric ASCII value rather than the character itself.

To do this, use the `\xhex number` regular expression, where `hex number` is a two-digit hexadecimal ASCII value ranging from 01 to ff.

Example: `\x0c`

Locates a *Ctrl+l* character (decimal ASCII 12, also called form feed).

It is not always necessary to fully specify the hexadecimal ASCII value. If the second character is not a valid hexadecimal digit, only the first one will be used. To avoid unexpected results, however, we recommend specifying both digits.

Matching a newline or a tab

\n matches a newline character. Use `\n` to search for line endings or in patterns that cross line boundaries.

Example 1: `;\n`

Finds every occurrence of a newline following a semicolon.

Example 2: `;\nif`

Searches for a semicolon, a new line, and a line beginning with `if`.

\t matches a real tab character, ASCII value 9 (decimal).

Example 1: `\t8`

Finds every occurrence of a tab character followed by an `8`.

Example 2: `init\t`

Finds every occurrence of a tab character following `init`.

Matching the beginning or ending of a line

< or **%** matches the beginning of a line.

Example 1: `<(macro`

Finds any line that begins with `(macro`. You could also specify `%(macro` as the pattern.

Example 2: `<if`

Finds only those occurrences of `if` that begin on a new line.

> or **$** matches the end of a line, but does *not* match the newline character itself.

Example 1: `\t>`

Finds every occurrence of a tab at the end of a line.

Example 2: `<\t>`

Finds a line that contains only one character: a tab.

The "beginning of line" and "end of line" regular expressions never include the newline character when they match. Replacing a pattern with a < or > does not affect the corresponding newline character in any way. If you want the newline character to be included, you must use `\n`.

Matching zero or more occurrences

@ matches zero or more occurrences of the preceding character or expression. The fewest possible occurrences of a pattern will satisfy the match. For example, `Hel@o` will find `Heo`, `Helo`, `Hello`, `Helllo`, and so on.

If `@` is at the end of a search pattern, it will always match zero occurrences of the preceding character or expression.

Although BRIEF defaults to minimal closure as described above, it can be configured for maximal closure as well. See **search_fwd** in the *Macro Language Guide* for more information.

Example 1: `0@1`

Matches any one of the following lines:

```
1
01
001
0001
00001
000001
```

...and so on.

Example 2: `\t@y`

Matches the letter `y` with zero or more tabs preceding it.

+ does the same thing as `@`, but matches one or more occurrences of the preceding character or expression. The fewest possible occurrences of a pattern will satisfy the match. That is, `Hel+o` will find `Helo`, `Hello`, `Helllo`, and so on.

If + is at the end of a search pattern, it will always match one occurrence of the preceding character or expression.

Example 1: `0+1`

Matches any one of the following lines:

```
01
001
0001
00001
000001
```

...and so on.

Example 2: `\t+y`

Matches the letter `y` with one or more tabs preceding it.

Matching either/or

`|` matches either the preceding or following character or expression. An expression is a set of characters enclosed in curly braces (`{ }`) or brackets (`[ ]`) (described below).

Example 1: `a|b`

Finds every occurrence of `a` or `b`.

Example 2: `him|her`

This pattern doesn't do what you might expect; it finds all occurrences of `himer` or `hiher`. Since the `|` character operates on the characters immediately preceding and following it, it matches either an `m` or an `h` in this example. See below to find all occurrences of `him` or `her`.

Grouping expressions

{ } groups characters or expressions for searches. Groups can be used to

- Control the evaluation of the pattern (see "How regular expressions are evaluated" later in this chapter for details)
- Group text so you can refer to each group by number (see `\num` for details on using numbered groups)

Groups can be nested. The maximum number of groups in a pattern is 10, numbered from 0 to 9. Groups are numbered from left to right, based on the opening brace.

Example 1: `{him}|{her}`

This example, which uses curly braces as grouping characters, finds all occurrences of `him` or `her`.

Example 2: `{hello}+`

Finds one or more occurrences of the word `hello`.

Example 3: `{{if}|{else}}`

Searches for `if` or `else`, and sets up three groups:

- Group 0 begins with the left-most brace and ends with the right-most brace. Group 0 consists of `{if}|{else}`, which will contain either `if` or `else` (whichever is matched)
- Group 1 consists of `if` (or nothing, if `if` is not matched).
- Group 2 consists of `else` (or nothing, if `else` is not matched).

The grouping characters around the `if` and the `else` specify the exact expressions to look for. If the braces did not enclose the `if` and the `else`, the vertical bar would affect only the `f` and the `e` (searching for `iflse` or `ielse`).

Grouping characters into sets (called expressions) has an effect on several operators, not just the vertical bar. See "How regular expressions are evaluated" for a more complete explanation of the effect of grouping characters in search patterns.

Matching a set

[ ] matches any one character or range listed within the brackets (`[` and `]`). Brackets change the behavior of a small set of characters.

`[ - ]` specifies a range of characters to search for, such as `[a-z]`. The beginning character of the range must have a lower ASCII value than the ending character of the range. This expression is only meaningful within brackets.

`[~ ]` matches a single character if it is *not* any one of the characters between `[~` and `]`. Note that this pattern also matches a newline character unless, of course, the newline character is included in the brackets.

`\c` cannot be used inside the set brackets since a set locates only a single character; using it outside the brackets has the same effect.

`{ } | + ? @ * < % > $` do *not* represent regular expressions inside brackets; therefore you can use them without using the backslash (override) character (see "Overriding regular expression characters"). Every other regular expression, such as `\t` or `\n`, does not change behavior within brackets.

Example 1: `[AEIOU]`

Finds an uppercase vowel.

Example 2: `[<>?]`

Finds a literal <, >, or ?.

Example 3: `[A-Za-z0-9_]`

Matches an upper- or lowercase letter, a digit, or an underscore.

Example 4: `[~0-9]`

Matches any character except a digit.

Example 5: `[ \t\n]`

Matches a space, a tab, or a new line.

Example 6: `[\[\]]`

Matches a literal [or] in the buffer. For an explanation of how the backslash works as an override character, see "Overriding regular expression characters."

Sets cannot be nested.

Example 7: `[~A-Z]`

Matches any character except an uppercase letter.

Using groups in the replacement string

\num, where *num* is a number that corresponds to the number of a grouped expression in the search string. Groups are numbered 0 through 9 from left to right, based on the position of the {, with 0 representing the first (leftmost) group. This regular expression is only valid in the replacement string for a **Translate** command.

Example 1: `{[Hh]i} world`

If you specify this pattern at the **Translate** prompt, and use this as the replacement,

```
\0 there
```

the `\0` is replaced by the text matched by `{[Hh]i}`, which is group 0. Group 0 will match any occurrence of `Hi` or `hi`.

Thus, every occurrence that matches the pattern (`Hi world` or `hi world`) will be changed to `Hi there` or `hi there`.

Example 2: `{[lc]ook}{{out}|{in}}`

If you specify this pattern at the **Translate** prompt, and this as the replacement,

```
\1\0
```

the `\1` is replaced by the string matched by `{{out}|{in}}` (either the letters `out` or `in`), and the `\0` is replaced by either `look` or `cook`, depending on which was matched. The total effect is to translate all occurrences of `lookout` to `outlook`, `cookout` to `outcook`, `lookin` to `inlook`, and `cookin` to `incook`.

Overriding regular expression characters

\, the backslash, is the override character. If the character following the backslash is a regular expression character, it is treated as a normal character. Any time a regular expression character is not preceded by the backslash, it operates as explained in the preceding discussions. The backslash is ignored if it is followed by a normal (non-regular expression) character.

Example: `\|`

finds a literal vertical bar (|) in the text.

Note that this provides an "escape" from the special meaning of the vertical bar in search patterns. It works in this manner with all special characters.

To search for a literal backslash, use `\\`.

The override character in the BRIEF macro languages is also the backslash. This means that you need two backslashes (`\\`) in a macro file to get one backslash in a pattern. So if you want to search for a literal backslash from a macro, you need to specify 4 backslashes (`\\\\`). This pattern translates to two backslashes in the compiled macro, which is the pattern you need to search for a literal backslash.

How regular expressions are evaluated

Rule 1 Grouping has the highest precedence. If you enclose one or more characters within braces `{ }` or brackets `[ ]`, it is considered to be a single, grouped expression. Thus, if you search for

```
{100}|{101}
```

you are searching for `100` or `101`. But, if you search for

```
100|101
```

you are searching for `10001` or `10101`.

Rule 2 The regular expression characters `|`, `@`, and `+` have the next highest precedence. These operators are matched from left to right. For example, if you specify one of the following:

```
a|b   (a or b)
a@b   (b preceded by 0 or more a's)
a+b   (b preceded by 1 or more a's)
```

it is clear which characters are involved in the operation. However, if you specify

```
a|b@c
```

it may not be clear at first which operation takes precedence.

Patterns are evaluated from left to right. In other words, `a|b@c` is equivalent to

```
a|b@c ≡ {a|b}@c
```

The `or` expression is evaluated first, so it matches any number of `a`'s or `b`'s followed by a `c`.

If the pattern were rearranged:

```
a@b|c
```

it would be equivalent to

```
a@b|c ≡ a@{b|c}
```

which matches any number of `a`'s followed by a `b` or a `c`. If you have two of the same operator in the same expression, they also are evaluated from left to right:

```
a|b|c ≡ {a|b}|c
```

Rule 3 After both grouping and the `|`, `@` and `+` operators have been taken into account, the remaining characters are concatenated (matched sequentially). For example, the pattern

```
ab|cd ≡ a{b|c}d ≠ {ab}|{cd}.
```

Turning off regular expressions

Turning off regular expressions Sometimes, you may find that regular expression characters get in the way. Turning regular expressions off allows you to search for question marks, asterisks, at signs, etc. without using the override character.

Ctrl+F6 (**Regular expression toggle**) toggles regular expressions on and off. The search prompt contains the string (`RE off`) when regular expressions have been disabled.

Combining regular expressions in search strings

Examples of combined expressions

Correctly combining these search expressions into a pattern provides precise location of matches and avoids incorrect matches. Any of these expressions can be combined.

Searching for a word

One use of regular expressions is to teach BRIEF the specific punctuation of a language. For example, an English word occurs in the following situations (ignoring the effects of punctuation):

- At the beginning of a line
- At the end of a line
- With a tab preceding or following
- With a space preceding or following

Example: The search string to find a word is

```
<|[ \t]\c[~ \t]+[ \t\n]
```

- `<` searches for the beginning of a line.
- `|[ \t]` *or* a space or tab.
- `\c` places the cursor at the word (rather than at the space, tab, and so forth).
- `[~ \t]+` finds the word (one or more occurrences of any characters but a space, a tab, or a newline).
- `[ \t\n]` *if* it is followed by a space, a tab, or a newline character.

Searching for programming language statements

This example searches for an `if` construct in the C language and places the cursor at the first character of the condition.

Example: `if[ \t]@(\c*)`

This search pattern reads: search for the word `if`, match as many spaces or tabs as necessary, find the first opening parenthesis, place the cursor on the first character following the parenthesis, and match any characters except a newline up to a closing parenthesis.

Searching for elements within matching quotes

This example searches for the occurrence of a word or element between matching double or matching single quotes. The trick, however, is to avoid finding something between one double and one single quote.

Example: `{"*"}|{'*'}`

This search pattern reads: find an occurrence of two single quotes separated by zero or more characters, or two double quotes separated by zero or more characters. Note that neither * will match across a newline.

Macros, playback and remember

It is possible to save keystroke sequences in a macro file that can be used later to save time. The following commands are described:

Command	Keystroke
Delete macro file	*Shift+F9*
Load keystroke macro	*Alt+F7*
Load macro file	*F9*
Pause recording toggle	*Shift+F7*
Playback	*F8*
Remember	*F7*
Save keystroke macro	*Alt+F8*

Remember

Causes BRIEF to remember a sequence of keystrokes when *F7* is pressed. All keystrokes will be stored until *F7* is pressed again to stop remembering.

You can create any number of remembered sequences during an editing session. However, only one sequence is saved at a time. When *F7* is pressed to begin a new remembered sequence, you are asked if you want to overwrite the existing sequence. Any previously remembered sequence is discarded when you begin a new sequence.

The number of remembered keystrokes is controlled by the `-k` flag. See "Flags and variables" (Chapter 5) for more information.

Prompt (if remembered sequence exists): `Overwrite existing keystroke macro [yn]?`

The appropriate responses are

- `y`

  Overwrites existing remembered sequence and starts remembering all subsequent keystrokes until the next press of *F7*.

- n

  Returns you to the current buffer without overwriting the existing remembered sequence.

- *Esc*

  Cancels the command. In effect, this response is equal to n.

The remembered sequence can be played back by using the *F8* key. See **Playback** for more information.

Pause recording toggle

Temporarily stops recording the current keystroke sequence when *Shift+F7* is pressed. This feature is useful if part of the keystroke macro being remembered is different each time the macro is played back. Before the variable part of the macro, press *Shift+F7*. Perform the variable part and press *Shift+F7* to resume recording.

When the macro is played back, it will pause at the point *Shift+F7* was pressed to allow the user to perform the variable portion of the sequence. Pressing *Shift+F7* resumes playback.

The **Pause recording toggle** command cannot be reassigned because it isn't really a command. It is recognized internally by the **Playback** and **Remember** commands.

Save keystroke macro

Save the current keystroke macro in the specified file when *Alt+F8* is pressed. If no extension is specified, .km is assumed.

Prompt: `Save keystrokes as [directory]:`

Type the name of the file where you want to save the keystroke macro. The file will be saved in the `directory` displayed in square brackets, which is the first one on your BPATH. If you want to save the keystroke macro in another directory on your BPATH, press *Enter* before typing a response so another BPATH directory is displayed.

If you don't want to save the keystroke macro in the displayed directory, you can specify a path along with the file name when responding to the prompt. If a relative path is used, that path is relative from the displayed directory. If the path is absolute, it is used as is.

If the file already exists, and backups are on, a backup copy is made.

Keystroke macros are saved as normal ASCII text and can be edited just like any other file.

Load keystroke macro

Loads a keystroke macro into memory when *Alt+F7* is pressed, if the specified file can be found on the disk.

If you specify a full path name, BRIEF searches the specified directory. Otherwise, BRIEF searches for the file on the BPATH. If found, the keystroke macro is loaded into memory. For information on the BPATH environment variable, see "Flags and variables" (Chapter 5).

Prompt: `Keystroke macro file:`

Enter the name of the keystroke macro file you want to load.

Prompt (if another sequence exists): `Overwrite existing keystroke macro [yn]?`

The appropriate responses are

- `y`

  Overwrites existing remembered sequence.

- `n`

  Returns you to the current buffer without overwriting the existing remembered sequence.

- *Esc*

  Cancels the command. In effect, this response is equal to `n`.

The loaded keystroke macro can be played back by using the *F8* key.

Playback

Plays back the last keystroke sequence recorded with the **Remember** command when *F8* is pressed.

Load macro file

Loads a compiled macro file into memory, if the specified file can be found on the disk, when *F9* is pressed. To execute a macro from the file, see **Execute command**.

If you specify a full path name, BRIEF searches the specified directory. Otherwise, BRIEF searches the current BPATH for the macro file. If found, the file is loaded into memory. For information on the BPATH environment variable, see "Flags and variables" (Chapter 5).

Prompt: `Macro file:`

Enter the name of the macro file you want to load.

Pressing *Tab* at this prompt will perform file name completion on the current response. The list will be restricted to `.cm` files.

Delete macro file

Deletes the specified compiled macro file from memory when *Shift+F9* is pressed. You must specify the full path name of the compiled macro file you want to delete, unless the compiled macro file is in the current directory.

Prompt: `Macro to delete:`

Enter the name of the macro file.

Pressing *Tab* at this prompt will perform file name completion on the current response. The list will be restricted to `.cm` files.

Using remember and playback

BRIEF has the capability to remember a sequence of keystrokes and play them back on demand. This remembered sequence of keys is called a keystroke macro.

To tell BRIEF to begin remembering a sequence of keystrokes:

- Press the *F7* key (the characters `RE` will appear in the mode indicator area of the status line).
- Enter the keystrokes you want BRIEF to remember.
- Press *F7* again to signal the end of the sequence (the `RE` mode indicator will be removed).
- Press *F8* to play back the remembered sequence (while the sequence is playing back, the characters `PL` will be displayed in the mode indicator area of the status line).

If you want BRIEF to pause while playing back a sequence of keystrokes (so, for example, you could type in a different response to a prompt each time you play back the sequence), you can press the *Shift+F7* key while remembering at the point where you want to pause. When you want to resume recording, press *Shift+F7* again. During playback, BRIEF will pause at the appropriate point. To continue the playback, press *Shift+F7*. Note that the characters `PA` will be displayed in the mode indicator area of the status line while the keystroke sequence is paused.

You cannot press *F8* to play back the remembered sequence until the sequence has been ended with the second press of *F7*.

Multiple keystroke macros

Keystroke macros recorded with the **Remember** command can also be saved on disk for use at some other time. See the **Save keystroke macro** and **Load keystroke macro** commands for more information.

Repeat

Another useful command for repeating keystroke sequences is the **Repeat** command, which lets you repeat any command, including a **Playback** of a remembered sequence, an arbitrary number of times. To use it,

- Press *Ctrl+R*.
- Enter the number of times you want to repeat the command (if not the default of 1), but do *not* press *Enter* after typing the number.
- Invoke the command that you want to repeat.

For more information on **repeat**, see the next section, "Special commands."

Special commands

These commands don't fit into any of the previous categories. The following commands are described:

Command	Keystroke
Background compilation	*F10* **bgd_compilation**
Change directory	*F10***cd**
Color	*F10* **color**
Create key assignment	*F10* **assign_to_key**
Delete file	*F4*

Command	Keystroke
Display version ID	*Alt+V*
Execute command	*F10*
Go to routine	*Ctrl+G*
Next error	*Ctrl+N*
Pause on error	*F10* **pause_on_error**
Pop-up error window	*Ctrl+P*
Repeat	*Ctrl+R*
Use tab characters	*F10* **use_tab_char**
Warnings only complete toggle	*F10* **warnings_only**

Display version ID

Displays BRIEF's version number and copyright notice on the status line when *Alt+V* is pressed. You will need to provide this information when calling for technical support.

Go to routine

Pops up a window that lists the routines present in the current file, if any, when *Ctrl+G* is pressed. You can select one of the routines from the menu and BRIEF will move the current position to the declaration of the routine.

The following keystrokes are used to manipulate the list of routines:

- ↑ and ↓

  Allow you to select a routine in the routine list.

- *Enter*

  Removes the routine window and goes to the selected routine.

- *Esc*

  Exits the routine list and returns you to the current buffer.

Next error

Locates the next error in the current file, if an error exists, when *Ctrl+N* is pressed.

This command should be used after the file has been compiled. The cursor moves to the file and line number where the compiler located the error, and displays the error message on the status line. If *Ctrl+N* is pressed when there are no more errors, the status line reads:

```
No more errors.
```

Pop-up error window

Displays a window of error messages when *Ctrl+P* is pressed. Examine any message or go to the line where an error occurred. This command should be used after the current file has been compiled.

The legal keys in the pop-up error window are:

- ↑ and ↓

  Select the previous or next error message.

- → and ←

  Move the cursor left and right one character.

- *Home* and *End*

  Move to the beginning and end of the line.

- *Enter*

  Selects the current error message, displays it on the message line, removes the pop-up error window, and moves the cursor to the line where the selected error occurred.

- *Esc*

  Removes the pop-up error window.

Repeat

Repeats a command a specified number of times when *Ctrl+R* is pressed. By default, the repeat count is one.

When *Ctrl+R* is pressed, a prompt appears. You can change the repeat count by typing a specific value, or you can increase it by a factor of four by repeatedly pressing *Ctrl+R*.

Prompt: `Repeat count = 1: type count or command.`

If you type in a number, such as five, the display of the repeat count changes:

Prompt: `Repeat count = 5: type count or command.`

If *Ctrl+R* is pressed a number of times, such as three, it increases the repeat count automatically, which changes:

Prompt: `Repeat count = 64: type count or command.`

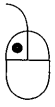
When the repeat count is correct, do not press *Enter*, but press the keys that execute the command. To cancel, press *Esc*.

Execute command

Executes the specified command when *F10* is pressed, which happens automatically if the command is built-in or a macro that is currently loaded into memory.

If the command is neither of the above, a compiled macro file by the same name is searched for, loaded, and executed. If the macro is not found, the execution fails.

This command is used to execute any command without a key assignment, such as the **Color** command, but cannot be used with commands that begin with an underscore, such as **_home**.



To activate the command line with the mouse, move the cursor to the bottom-left area of the status line and click *Mouse Button 1*.

Prompt: `Command:`

Type the command name and press *Enter*.

Color

Resets the colors used for the background, foreground, titles, and messages when *F10* **color** is entered.

Colors must be specified using the following assigned numbers. Only the first eight numbers (0-7) can be used for the background:

Value	Color
0	Black
1	Blue
2	Green
3	Cyan

Value	Color
4	Red
5	Magenta
6	Brown
7	White
8	Dark Grey
9	Bright Blue
10	Bright Green
11	Bright cyan
12	Bright Red
13	Bright Magenta
14	Yellow
15	Bright white

At the prompts, enter the number that identifies the color. Do not duplicate color selections of background and foreground; text will be invisible if placed on a background of the same color and you will receive an error message.

Prompt 1: `Enter background color number:`

This is the color of the area where text is entered and edited. This value is not used for the background color of windows that do not have borders, but is used as the background color of the status line.

Prompt 2: `Enter foreground color number:`

This is the color of the text entered into the editor and the editor's border, including the line, column and time indicators.

Prompt 3: `Enter selected title color number:`

This is the color of the title of the buffer that is centered at the top of the current window.

Prompt 4: `Enter normal message color number:`

These are the colors of the prompts and messages displayed when a command is successfully completed.

Prompt 5: `Enter error message color number:`

These are the colors of the messages displayed when a command does not complete successfully.

Change directory

Changes the current working directory when *F10* **cd** `directory_name` is entered, where `directory_name` is the name of the directory that should be the new current directory. If you don't specify any parameters, the current working directory is displayed on the status line. If you only specify a drive letter, that drive is made current.

Create key assignment

Adds a temporary key assignment to the current keyboard when *F10* **assign_to_key** is entered.

This is not ordinarily used as a command; see **assign_to_key** in the *Macro Language Guide* for more information.

Delete file

Deletes a file from disk when *F10* **del** `file_name` is entered, where `file_name` is the name of the file to be deleted. You cannot delete the file you are editing.

Warning: This command cannot be undone.

Pause on error

Tells BRIEF to pause when displaying run-time error messages when *F10* **pause_on_error** is entered. Otherwise, the messages flash by at a rapid rate.

When this is executed, one message at a time is displayed on the BRIEF command line. Each message is followed by three periods. For example:

```
New file (unable to open <file name>)...
```

Press any key to continue.

Use tab characters

Determines whether spaces or tabs are inserted when the *Tab* key is pressed. You can control whether the *Tab* key inserts tab characters or spaces three ways:

- During SETUP, specify that BRIEF should fill empty areas with tabs or spaces.

- Using the `-t` flag, which toggles the current default from the command line or from BFLAGS.
- Using this command (*F10* **use_tab_char**).

Warnings only complete toggle

This toggle forces the **Compile buffer** command to check the output from the compiler for messages when *F10* **warnings_only** is entered. If any warning or error messages are found, the compile is considered to have failed.

This is useful if your compiler doesn't tell BRIEF that the compile failed or if you want to ensure that your programs produce *no* compiler diagnostics.

Background compilation—*OS/2*

Toggles whether or not all compilations should be performed in the background when *F10* **bgd_compilation** is entered. Note that compilers can be controlled on an individual basis. See "Programming support" (Chapter 6) for more information.

C H A P T E R

4

Using special features

BRIEF contains some special features that are more complex than other commands. In some cases, SETUP automatically configures BRIEF to use these features, so you don't need to take the additional steps described here if SETUP has done them for you.

BRIEF's special features are

- Simple text formatting capabilities
- Reconfiguring the keyboard
- The `restore` feature, which saves and restores information from editing session to editing session
- Running other programs inside BRIEF

DOS

- Swapping
- Using the Dialog Manager

Each feature is discussed in this section, and includes opportunities for practice.

Simple text formatting

This feature provides simple commands to assist you in creating letters or writing short documents. The following commands are supported:

- Margin
- Reform
- Center
- Automatic wordwrap

All of the word processing commands except for automatic wordwrap can be used at any time in any file.

Margin

Sets the right margin for wordwrap and paragraph reformatting. The default is 70. To set the margin:

- Press *F10.*
- Type `margin` and press *Enter.*
- Enter the new margin. The number you type corresponds to the column in the editor where wordwrap occurs and where the lines of reformatted paragraphs end.

The left margin is set at the first column and cannot be changed.

Reform

Reformats a paragraph, using the current margin setting. A paragraph can be defined by marking it as a block, surrounding it with blank lines, or starting it with an indented line.

To reformat a paragraph

- Move the cursor into the paragraph.
- Press *F10.*
- Type `reform` and press *Enter.*

There is a short delay while the paragraph reformats.

Center

Centers a line of text between the first column and the right margin.

To center a line

- Move the cursor to the line.
- Press *F10*.
- Type `center` and press *Enter*.

Automatic wordwrap

Wordwrap is done automatically in `.txt` and `.doc` files if they've been added to SETUP's file extensions menu. If you want to turn wordwrap on manually through BPACKAGES (see "BPACKAGES" in Chapter 6 for more information), the name of the wordwrapping package is `wp`.

Reconfiguring the keyboard

Using the **keys** command allows you to completely reconfigure the keyboard to your preferences by assigning almost any key sequence to any command (see the Keyboard assignment chart in the *Macro Language Guide*). This command also allows you to see the current key assignments.

Press *F10* and type **keys**.

Two windows will appear on your screen. The left (command) window contains a list of BRIEF commands. The right (assignment) window contains the current key assignment(s) for the highlighted commands.

The following keys can be used in the command window:

- ↑ and ↓

  Select a command.

- *PgUp* and *PgDn*

  Move up or down one page.

- *Home*

  Move to the first entry in the window.

- *End*

  Move to the last entry in the window.

- Alphanumeric keys

  Moves to the first command beginning with that letter. Pressing the same key again moves to the next entry in the list, unless there aren't any more.

- *Ins* or →

  Moves to the assignment window so you can add or delete a key assignment for the highlighted command.

- *Esc*

  Saves your assignment and returns to BRIEF.

The following keys can be used in the assignment window:

- ↑ and ↓

  Select one of the displayed assignments.

- *Del*

  Deletes the selected assignment.

- *Ins*

  Add an assignment for the selected command. Press the assignment keys, then *Enter*. Press *Esc* to cancel the assignment.

- *Esc*, *F10*, or ←

  Move to the command window.

If you pass the **keys** command from the command line with nonzero parameters (for example, `<F10> keys 1 <Enter>`), the appropriate **assign_to_key** statements will be inserted into the current buffer. This feature is useful if you are writing your own macros to reassign the keyboard.

keys modifies a file called `keyboard.h`, which is located in the directory where the macro source files are stored. It also needs to access the help files.

If the macro source files are not installed, or if the compiled macro files are stored in a different directory than the sources, **keys** can only be used as a quick reference.

Restore

When you use the restore feature, BRIEF saves information about your editing session when you exit, and restores it when you start to edit again. This information includes

What BRIEF saves and restores

- The list of files you are editing, and the positions and bookmarks in those files
- The sizes and layout of the windows on the screen, and the buffers displayed in those windows
- The current search pattern
- The current translate pattern
- The current translate replacement string
- The search direction
- The regular expression setting
- The case sensitivity setting

OS/2

- The background compilation setting

To turn on **restore**, you can use SETUP or set your BFILE and BFLAGS manually to the correct values. See "Complete installation" in "Installing BRIEF" (Chapter 1) or "Flags and variables" (Chapter 5) for more information.

PWB

BRIEF provides support for Microsoft Programmer's Workbench, which can be used in place of **restore**. See "Third-party packages" (Chapter 7) for more information.

The state file

The information about your editing session is saved in a special file called the state file, which always has the extension `.rst` (Restore State), and is kept in the file and directory specified in the BFILE environment variable.

If `state.rst` becomes corrupted, the restore macro may not be able to recover the editing session. Delete `state.rst` and start BRIEF again with a file name.

Warning: If you're using BRIEF's Restore feature, you cannot use the BFILE environment variable for any other purpose. Doing so will cause Restore to work incorrectly or not at all.

Global vs. local state files

The state file can be kept *globally* or *locally*. Keeping global session information means that when you start BRIEF, your session information will always be saved to (and restored from) the same state file. To accomplish this, you set BFILE to a fully qualified path, including the drive letter.

For example, when you install BRIEF and turn on the Restore feature, SETUP automatically sets your BFILE so your session information is stored globally, and sets your BFILE to

```
set bfile=c:\brief\state.rst
```

This means BRIEF will always save and restore the state information using the file `state.rst` in the `c:\brief` directory, no matter what drive or directory you start BRIEF from.

Keeping project information in separate state files

When you're working on multiple projects, it may be desirable to have separate state files, depending on the directory or disk you happen to be on when you start BRIEF. Whenever you start BRIEF in the directory that corresponds to a particular project, the state information related to that project is restored.

To accomplish this, set your BFILE using a relative path or no path at all. For example, if you want your state information saved depending on the drive you started BRIEF from, set your BFILE as follows:

```
set bfile=\state.rst
```

BRIEF would then save and restore using the state file in the root directory of the current drive (of course, you could specify any pre-existing directory you liked). Alternatively, you could keep your session information in the current directory. In that case, leave the path off completely, and set your BFILE to

```
set bfile=state.rst
```

The restore macro

The final step in turning on the Restore feature is actually calling the macro that does the work. SETUP does this for you automatically when you install BRIEF. You can also do it manually using a command like

```
set bflags=-mrestore
```

The BFLAGS and BFILE environment variables are discussed in more detail in "Flags and variables" (Chapter 5).

Running other programs inside BRIEF

DOS commands and other programs in BRIEF

One of BRIEF's nicest features is its ability to run other programs without exiting to DOS or OS/2. The automatic compilation features discussed in "Programming support" (Chapter 6) take advantage of this, but you can use it to run almost any program you want without waiting for BRIEF to reload when it's done.

There are two ways to run a program in BRIEF. The easiest is to press *F10* and type `dos` followed by the name of the program or command you want to run. For example, if you want to get a listing of your files using the `dir` command, you could press *F10* and then type

```
dos dir
```

This would display a directory listing on your screen, then wait for you to press any key to resume BRIEF. If you want to run a command with parameters, like `type config.sys`, surround it with quotes:

```
dos "type config.sys"
```

Another way to run programs in BRIEF is to escape to the operating system by pressing *Alt+Z*. When you do this, the BRIEF screen will be replaced by a normal DOS or OS/2 screen, and you will be free to run any programs. When you want to return to BRIEF, simply type

```
exit
```

at the prompt.

DOS

If you find that you don't have enough memory to run a program inside BRIEF, you may be able to free some memory by enabling swapping (see next section), saving all of your files, or using the `-Mnum` startup flag. See "Flags and variables" (Chapter 5) for additional information.

Warning: Make sure you *never* delete or rename any files that you are editing, macros that are loaded, or temporary files while in the shell. *If you do so, you will lose your work.* Also, make sure that any programs you run are fully debugged, or they may clobber BRIEF and destroy your editing session.

Warning: TSRs or other programs that allocate memory but do not free it when you re-enter BRIEF can cause problems. For example, if you shell to DOS and run a memory resident program, then return to BRIEF, the memory resident program is still allocated memory. When you finally exit BRIEF, your system will have fragmented memory assignments. Sometimes, the memory can be freed. In most cases, however, you will need to reboot.

The BRIEF Macro Language lets you take full advantage of this interface. See the *Macro Language Guide* (the `dos` function) for further information.

Swapping—*DOS*

Normally, the programs you run from within BRIEF work with a greatly reduced amount of memory. This is because BRIEF only gives these programs the memory it isn't using for itself. If you're editing a lot of files or have made a lot of changes, your programs may not have enough memory to run.

BRIEF's swapping feature solves this problem. When swapping is enabled, BRIEF "steps aside" when other programs are run, freeing all but 8K of BRIEF's memory for use by the application. This happens automatically whenever you use **Suspend BRIEF**, **Compile buffer**, or any other command that runs a DOS program inside BRIEF.

Where BRIEF swaps to

BRIEF will swap to EMS memory, if present, or to disk. If BRIEF swaps to EMS memory, a RAM disk or a fast hard disk, the delay required to write the swap information is barely noticeable.

Warning: When swapped to disk, BRIEF saves crucial information in a file named `BR-xxxx.SWP` in the BTMP directory (or TMP directory: see "Environment variables" in Chapter 5), where `xxxx` is a four-digit hexadecimal number. The information in this file is *extremely* important. *Do not delete or modify this file for any reason. If you do so, your BRIEF session* ***will be lost.***

Swapping storage requirements

The storage device you're swapping to must have as much free space as CHKDSK reports you have free memory at the DOS prompt. This means that if CHKDSK reports 458,752 bytes free before BRIEF is loaded, you'll need about that much EMS memory or disk space for swapping to work.

When swapping is enabled, BRIEF first checks to see if you have enough (or any) EMS memory. If you don't, BRIEF tries to swap to the TMP directory (see "Environment variables" in "Flags and variables" [Chapter 5]). If BRIEF can't swap to either EMS or disk, it runs the program as if swapping were disabled.

Using the BRIEF Dialog Manager package

The BRIEF dialog manager is a macro package containing tools for generating pop-up menus and dialog boxes. Menus allow a user to perform complicated tasks by choosing from lists of possible actions. Dialog boxes allow a user to answer questions and fill in forms, while possibly restricting the set of valid answers. The user can use cursor keys (explained later) or the mouse to select options.

Many of BRIEF's standard macros (**help**, for example) use the dialog manager. You can use the dialog manager in your macros, too.

Source code for the package is located in the files `dialog.cb`, `dialog.h`, `dlg.h`, `dlg_butn.cb`, `dlg_fld.cb`, `dlg_list.cb`, `dlg_menu.cb`, and `dlg_mous.cb`; the compiled version is in `dialog.cm`.

The BRIEF dialog manager provides a standard user interface for macros. If you write a macro that uses this interface properly, it will have the look and feel of the standard commands that are shipped with BRIEF and any BRIEF user will be able to use it with minimal effort.

Using the dialog manager conserves memory and avoids reinventing the wheel. Suppose you frequently use Help and three other macro packages with separate user interfaces. BRIEF may spend a fair amount of time re-reading the disk because of the number of macros to load at once. They will take up far less memory if they reuse the same code.

The dialog manager is essentially just two macro calls, one to create and process a menu, and one to create and process a dialog box. Although the dialog manager contains dozens of other macros (as well as several global variables,) you should seldom need to deal with them. (Consult the source code for more information.)

Menus are pop-up windows, with one line ("button" always highlighted. Pressing *Enter* ("choosing") causes an action associated with the highlighted button to be executed. The up and down arrow keys move the highlight up or down. Windows scroll vertically as needed.

Menus offer a choice of several actions, which may invoke other menus or dialog boxes. In Help, for example, picking some buttons causes a window of help information on a particular topic to appear; picking others causes another menu (more detailed) to appear.

Dialog boxes

Dialog boxes, the other half of the dialog manager, are pop-up windows that contain descriptive text and any of several kinds of input fields. A user may move between fields and change the contents of any field. These changes can be saved or cancelled as a group.

The kinds of input fields are

- File names
- Lists
- Integers
- Nonblank strings
- Strings
- Buttons

Lists

Lists, like menus, are multiple-choice situations. However, lists are choices of values, not actions, and the values are usually mutually exclusive. One very common list has two choices: Yes and No.

Lists appear on the screen on a single line, with items separated by whitespace. A highlight is used to mark the currently selected item while the user's cursor is on the field. When the cursor is moved, the highlight changes to a pair of parentheses.

A list can be paired with a text string, which is usually a question:

```
Do you want to save your changes? (Yes)No
Line drawing style:               Single(Double)Mixed None
```

File names, integers, nonblank strings, and strings

The number of possible answers to a question is often so great that a list is awkward or impossible. In such situations, it's best to let the user enter an answer, then check it for validity. The user can be asked to re-enter an invalid answer.

File names, integers, nonblank strings, and strings behave identically except that they use different validation criteria:

- File name fields must contain a syntactically legal path name
- Integer fields must contain an integer value
- Nonblank strings may not be blank
- Strings are not checked at all

Fields are one line deep and may extend to the right edge of the window. The entire field is highlighted when the cursor enters it; the highlight disappears as soon as anything happens. When the cursor leaves, the field contents are validated.

Buttons

There are three types of buttons: push buttons, check boxes, and radio buttons. All three have toggle (on/off) relationships. When chosen, the current state toggles to the opposite state. When a radio button is chosen, its state is toggled on and all other radio buttons are toggled off.

Key assignments

The key assignments for moving around and between menus and dialog boxes are standardized. You should not change these key assignments, although you may add your own to the list.

In addition to these keys, the mouse can be used by clicking *Mouse Button 1* to achieve the desired action. In the following lists, *Mouse* is listed as the key assignment. The resulting action and cursor location are listed in the second column.

These key assignments are uniform throughout the dialog manager:

Esc	Exit all levels
Keypad Minus	Exit 1 level
Ctrl+Home	Top of window
Ctrl+End	End of window
Mouse	Drag thumb up the scroll bar: Top of window
Mouse	Drag thumb down the scroll bar: End of window

Within menus, the additional assignments are

Enter	Pick from menu
Mouse	Double-click: pick from menu
↑	Previous item
↓	Next item
Backspace	Previous item

Space	Next item
Mouse	Choose item
Home	First item
End	Last item
PgUp	Previous page
PgDn	Next page
Mouse	Click above thumb: previous page
Mouse	Click below thumb: next page
Alphanumeric characters	Next item beginning with characters

Within any dialog box field, the following keys are defined:

Enter	Next field, or save if on last field (== *Ctrl+M*)
F10	Save dialog box and exit
↑	Previous field
↓	Next field
Shift+Tab	Previous field
Tab	Next field
Mouse	Click on desired field

Within lists only, additional assignments are

←	Previous item
→	Next item
Backspace	Previous item
Space	Next item
Home	First item
End	Last item
Mouse	Click on desired item: previous or next item
Alphanumeric characters	Next item beginning with characters

Within fields (but not lists), additional key assignments are:

←	Left
→	Right
Backspace	Backspace
Space	Space
Home	Beginning of field
End	End of field

Mouse	Click on field: insert point is positioned at cursor
Del	Delete character
Alt+K	Delete to end
Ins, Alt+I	Toggle insert mode
Alt+D	Delete field

When a field is highlighted, typing causes the contents of the field to be replaced by the new characters.

Menus

Creating a menu

There are three steps involved in creating a menu.

1. Create a data file containing the names of the menu buttons and the actions you want the buttons to invoke.
2. Write a macro that calls the dialog manager package.
3. Write the action macros that will handle everything the user does in the menu.

Creating the menu data file

Menu data files are usually given a `.mnu` extension and kept in the same directory as the help files (specified by BHELP).

A menu file contains one line for each line of the menu. Each line consists of a button name and optional additional information. When the menu is displayed, the additional information should be invisible (just past the right edge of the window). The menu button is normally centered in the visible part of the line.

For fastest operation, each line is normally pre-formatted. In this example (taken from `help.mnu`), spaces are used to center the buttons in the visible part of the window, which is delimited by the semicolons. Everything to the right of the semicolons is considered "additional information":

```
   Help on Help     ;display_help "help"
  Quick Reference   ;display_help "keyboard layout"
Key-Specific Help ;key_specific_help
```

You can also have the menu automatically formatted. If you do, you must use a semicolon as the delimiter, and leading/trailing whitespace around the buttons will be ignored:

```
Help on Help;display_help "help"
Quick Reference;display_help "keyboard layout"
Key-Specific Help;key_specific_help
```

Action macros, described below, are free to do whatever they want with the additional information. In this case, the information is a macro call (complete with a string parameter), and it is passed to **execute_macro**.

Calling the menu

The call to create and process a menu is as follows:

```
_process_menu (lx, by, rx, ty, title, msg, file name,
buf_id,
  action, fast, picked_line, picked_text);
```

Parameters 0 through 3

`lx`, `by`, `rx`, and `ty` are parameters 0 through 3. They are integers, the coordinates of the left (`lx`), bottom (`by`), right (`rx`), and top (`ty`) edges of the window. These are absolute screen coordinates; the top left corner of a 25x80 screen is (0, 0) and the lower right is (24, 79).

If `lx` equals `rx` or `ty` equals `by`, the menu is positioned at (`lx, ty`) and the size will be computed automatically based on the number of lines on the menu and the column position of the first semicolon. (Be careful not to position menus so close to the bottom and left edges of the screen so that there's no room for automatic formatting.)

Parameter 4

`title` is the title of the menu, which is a string to be displayed in the top edge of the window. It can be 12 characters long maximum.

Parameter 5

`msg` is a message to be displayed in the bottom edge of the window. Make sure it is shorter than the edge itself.

Parameters 6 and 7

`file name`, parameter 6, is the file name of a menu data file (in the BHELP directory). `buf_id`, parameter 7, is the identifier of a buffer containing a menu data file. Use `buf_id` if you have an `.mnu` file already in memory and want to keep it there; otherwise, use `file name` and the dialog manager will delete the buffer when you're done with it. (If you use `buf_id`, the title you passed in `title` will be ignored and the buffer name already in existence will be used.)

You should supply one, but not both, of these parameters. Supply NULL in the place of whichever parameter you omit.

Note that `file name` is assumed to be the name of a file in the BHELP directory unless an absolute path name is supplied. The dialog manager assumes that any name containing the characters `/`, `\`, or `:` is an absolute path.

Parameter 8

`action` is a string containing the name of your main action macro, described below.

Parameter 9

`fast` may be omitted. If it is passed and TRUE, your menu file is not formatted in any way, saving time when it pops up. If you use this parameter, make sure to pre-format your `.mnu` file.

Parameter 10

`picked_line` is an optional integer variable which will be set to the number of the menu button that the user picks. It is only valid when the call to **_process_menu** returns TRUE. If the user presses *Esc* or *Keypad Minus* to escape from a menu, this parameter will be zero.

Parameter 11

`picked_text` is an optional string variable which will be set to the text of the menu button that the user picks. It is only valid when the call to **_process_menu** returns TRUE and `picked_line` is nonzero.

Warning: The dialog manager assumes you are passing it correct values, so be careful.

When the call to **_process_menu** is encountered, the menu will pop up and processing of the user's keystrokes will begin immediately. Processing will stop when the user presses *Esc* (or *Keypad Minus*), and control will return to your macro, just after the **_process_menu** call.

_process_menu returns TRUE if it was able to create and process a menu, FALSE if an error occurred.

Events

The dialog manager has been written to be as flexible as possible. Hence, the system leaves a lot of decisions and a lot of work up to your action macro. The dialog manager is event-driven; it quietly goes about its business until an "event" of significance occurs, at which point it taps your macro on the shoulder, informs it of the situation, and waits for a response before continuing.

Here's a list of all significant events for both menus and dialog boxes. Matching **#define** statements are contained in the file `dialog.h`. You should **#include** this file in any macro file you write that calls the dialog manager, so that you can refer to the event types by symbolic name rather than by number.

DIALOG_INIT	The dialog manager has just been invoked
DIALOG_TERM	The dialog manager is exiting
DIALOG_ENTER_FIELD	The cursor has just entered a non-list field
DIALOG_EXIT_FIELD	The cursor has just left a non-list field
DIALOG_ENTER_LIST	The cursor has just entered a list
DIALOG_EXIT_LIST	The cursor has just left a list
DIALOG_ALTER_LIST	The current item in a list has just been changed
DIALOG_ALTER_MENU	The current item in a menu has just been changed
DIALOG_MOVE_MENU	(Same as DIALOG_ALTER_MENU)
DIALOG_PICK_MENU	The user has just selected a menu button with *Enter*
DIALOG_ESCAPE	The user has just pressed *Esc* to exit all levels
DIALOG_F10	The user has just pressed *F10* to save a dialog box
DIALOG_GREY_MINUS	The user has just pressed *Keypad Minus*
DIALOG_CREATE_MENU	A menu buffer has just been made current
DIALOG_CREATE_DBOX	A dialog box data buffer
DIALOG_ENTER_CHECK	The cursor is moved into a check box
DIALOG_PICK_CHECK	The user has selected a check box
DIALOG_EXIT_CHECK	The cursor has moved off a check box
DIALOG_ENTER_PUSH	The cursor has moved onto a push button
DIALOG_PICK_PUSH	The user has selected a push button
DIALOG_EXIT_PUSH	The cursor has moved off a push button
DIALOG_ENTER_RADIO	The cursor has moved onto a radio button
DIALOG_PICK_RADIO	The user has selected a radio button
DIALOG_EXIT_RADIO	The cursor has moved off a radio button

Writing the menu action macros

When you're working with menus, only the following events are pertinent: invoke/exit, *Esc*/*Keypad Minus*, and changing or selecting menu items. Your job is to provide a single macro that responds to these events The name of this macro is parameter 8 in the call to **_process_menu**, and the macro is called by **_process_menu** whenever one of these events occurs.

Suppose you call your action macro **action**. When called, action will be passed three parameters:

```
action (event_type, line, text);
```

The first parameter is an integer matching one of the above **#defines**. The second is the number of the current line in the menu (note that this is also the number a user will see as `Line:` in the BRIEF message area). The third is the text of the entire button, including the delimiter and additional information (if any).

In the BRIEF macro language, parameters in the calling function are only evaluated when the called function explicitly requests the parameter. This means that, for best performance, your action macro should never get a parameter it does not need.

The normal strategy for processing events within a menu action macro is as follows:

1. Get the event type parameter.
2. If the event type is DIALOG_PICK_MENU, then get the line number or text parameters and process them as desired. Note that the dialog manager does *not* execute an action associated with a button automatically. Your action macro must parse the button text and execute the command.

   Commonly, a pick event will cause the action macro to pop up a window of text (such as Help) and enter a process under the user's control. The window will remain visible until the user presses *Esc* or *Keypad Minus* to signal he's done. If the user pressed *Esc*, the action macro should push the *Esc* back into the keyboard buffer so that a DIALOG_ESC event is generated as soon as the action macro returns.
3. If the event type is DIALOG_CREATE_MENU, the unformatted menu is the current buffer and its name is the text parameter. You can use this event to add buttons to the menu.

(If the event type is anything else, you usually don't need to do any more. However, if you want to add key assignments to the normal menu keymap, you can use DIALOG_INIT as a signal to add them and DIALOG_TERM as a signal to remove them. If you want to prevent the user from moving to a particular line of the menu (for example, a heading), then you use DIALOG_MOVE_MENU.)

4. Return TRUE. The only time you should return FALSE is if the event was DIALOG_MOVE_MENU and the line number or text was such that you don't want the cursor to be allowed to move to it (other lines will be tried until EOF or until DIALOG_MOVE_MENU succeeds).

Example

The following macro, when called by the dialog manager, will handle two types of events. First, the user will not be allowed to move to line 1 or line 4. Second, when the user picks a button, this macro will use the additional information present in the `.mnu` file to call another macro.

```
action (...)
{
 (int  event_type,
    line_no,
    retval;

 string   button_text

 get_parm (0, event_type);
 retval = TRUE;

 switch (event_type)
    {
    case DIALOG_MOVE_MENU:
       {
       get_parm (1, line_no);

       if (line_no == 1 || line_no ==4)
       retval = FALSE;
       }

    case DIALOG_PICK_MENU:
       {
       get_parm (2, button_text);

       /* Trim the button name and treat the additional
       */
       /* information as a command that can be executed.
       */

       execute_macro (substr (button_text, index
```

```
            (button_text, ";") +1));
            }
        }
    returns retval;
}
```

Dialog boxes

Creating a dialog box

As with menus, there are three steps to creating a dialog box: creating a data file, writing a macro that calls the dialog manager, and writing action macros.

Data files for dialog boxes

Dialog box data files must be kept in the BHELP directory. Like menu files, they contain one line for each item displayed. Each line contains the type of the input field, the row and column coordinates (relative to the window's origin) at which the field should be placed, and the field's initial contents. The number of fields is unlimited because dialog boxes, like menus, can scroll vertically.

The row and column coordinates must be surrounded by parentheses and separated by a comma. The initial contents must be surrounded by double quotes. Whitespace may go just about anywhere and you can have comments.

The following example shows a number of legal ways of defining fields, and descriptive text, in a dialog box:

```
;**  Get the speed using a 3-item list.
Text at (1, 1) is "Speed:"
List ( 1, 8 ) = "(Slow)Medium Fast"

;**  Get the user's name.  Make sure they enter one.
text (3,1) "Name:"
  nonblank(3,9)  "Your name goes here"
;**  Prompt for user's age.  Make sure it's numeric.
T    (5, 1)   "Age:"
I(5,9)"24"
```

The first non-whitespace character on a line defines the type of the field. A `T` means that this line is a descriptive text string that can be displayed only. The letters `F`, `L`, `I`, `N`, and `S` denote File names, Lists, Integers, Nonblank strings, and Strings.

All fields, even lists and integers, are really strings. Hence, you should format the initial contents of every field as a quoted string, as in the example above.

List strings have a special format. Items must be separated by single tab characters (since spaces are allowed in item names). The item that is to be initially highlighted should be surrounded by parentheses instead of tabs.

Position lists as you would any other field. If a list begins with a `(`, the dialog manager will compensate for you.

If you omit the current item in a list, or if you supply a field that's too wide for the window, the invalid field will not be displayed, but an error message will.

The dialog manager does not check to see if any fields overlap. Hence, only one input field is allowed per line. Text can be placed anywhere except to the right of an input field.

Calling a dialog box

The call to create and process a dialog box is nearly identical to the call to create a menu, but the macro name is **_process_dialog_box**:

```
_process_dialog_box (lx, by, rx, ty, title, msg, file
name,
buf_id, action);
```

All parameters have the same meaning as for **_process_menu**, except:

1. There is no automatic sizing for dialog boxes, so `lx` must not equal `rx`, and `by` must not equal `ty`.
2. Parameter 4, `title`, is always used, even when a buffer ID is passed for parameter 7.
3. There are no parameters 9, 10, or 11.

This call may return when the user presses *F10, Esc,* or *Keypad-Minus.* **_process_dialog_box** returns TRUE if it was successful, FALSE if an error occurred.

Writing the action macros

The first two parameters passed to your action function are the event type and line number. The third parameter is either (for lists) the text of the current list item, not counting separator characters, or (for other field types) the full text of the field, not counting the newline at the end.

Any event may occur in a dialog box except DIALOG_ALTER_MENU or DIALOG_PICK_MENU. However, the only events you normally have to worry about are DIALOG_EXIT_FIELD, DIALOG_EXIT_LIST, and DIALOG_F10.

You can perform additional validation on the contents of a field when the user wants to move to another field (DIALOG_EXIT_FIELD). After the dialog manager has performed its built-in type checking on the field, your action macro will be called. If your action macro returns FALSE for any reason, the user will not be allowed to leave the field. You can use a String field coupled with your own validation function to check for almost any value. Lists are not validated, since all the possible answers are known in advance.

When the user leaves a field or a list (DIALOG_EXIT_FIELD or DIALOG_EXIT_LIST), keep track of the value of that input field. There is no way to determine the value of an input field at any other time. We recommend you use a separate buffer, where each line represents the contents of one input field, for dialog boxes containing more than three or four input fields; otherwise, use string variables to store the current values.

Treat DIALOG_F10 as a signal that the user is satisfied with the changes and wants to save them. The current list or field will be exited before DIALOG_F10 occurs. When it does, process the values you have saved in your macro. (DIALOG_ESC and DIALOG_GREY_MINUS should just ignore the saved values.)

Example

This data file defines a dialog box containing two lists.

```
Text (1, 3)  = "Do you want case-sensitive search?"
List (1, 38) = "(Yes)No"
Text (3, 3)  = "Do you want regular expressions?"
List (3, 38) = "(Yes)No"
```

This macro puts up a dialog box containing the above information and defines two string variables that are used by the action macro. Both variables are set to `Yes`, since the data file's defaults are `Yes`.

```
put_up_box (...)
{
 string    case_sens,
           reg_exp;

 global    case_sens,
           reg_exp;

 case_sens = "Yes";
 reg_exp = "Yes";
 _process_dialog_box (10, 15, 60, 10, "Search", "Set
 search parameters", "example", NULL, "action");
}
```

The following macro will process the dialog box and save the current values when *F10* is pressed:

```
action (...)
{
 int   event_type,
       line_no;

 string button_text;

 get_parm (0, event_type);

 switch (event_type)
       {
       case DIALOG_EXIT_LIST:
            {
            get_parm (1, line_no);
            get_parm (2, button_text);

            if (line_no == 1)
                case_sens = button_text;
            else
                reg_exp = button_text;
            }
       case DIALOG_F10:
            {
            search_case (case_sens == "Yes" ? 0 : 1);
            toggle_re (reg_exp == "Yes" ? 1 : 0);
            }
       }
 returns TRUE;
}
```

A more sophisticated macro could keep the data file in a buffer and continuously alter it to make sure the correct values of case sensitivity and regular expressions were always displayed.

Buttons

Check boxes

Check box control is displayed as checked or unchecked. When checked, it looks like this:

```
[√] Checkbox label
```

When unchecked, there is no checkmark between the brackets.

The specification syntax for a check box control includes the control's row and column position within the dialog box and the control's label.

```
Check (2,3) "Checkbox label"
Check (3,3) "Another checkbox"
Check (4,3) "The third checkbox"
```

The user chooses a check box by pressing *Spacebar* or clicking the mouse when the mouse cursor is over the check box. The Dialog Manager will then call your action function with the DIALOG_PICK_CHECK event and the label of the check box that was picked. The Dialog Manager does not automatically toggle the state of a check box. Your action function must toggle the state by making use of the two functions below.

```
void dialog_check_button (string label, int check)
```

This function places a checkmark next to or removes a checkmark from a check box control. The label specifies which button to change. If the parameter check is nonzero, a checkmark will be placed. Otherwise, if a check exists, it is removed.

```
string dialog_is_checked (string label)
```

This function determines whether the control specified by label has a checkmark next to it. If the return value is nonzero, the control has a checkmark. Otherwise, it does not.

Push buttons

A push button control is displayed as selected or unselected. When selected, it looks like this:

```
«Push Button»
```

When unselected, it looks like this:

```
<Push Button>
```

The specification syntax for a push button control includes the control's row and column position within the dialog box and the control's label. A push button control may also be specified as the "default" push button, which is always selected unless another push button has the focus.

```
Push (2, 15) "OK" DEFAULT
Push (4, 15) "Cancel"
```

The user chooses a push button by pressing *Spacebar* or clicking the mouse when the mouse cursor is over the push button. The Dialog Manager will then call your action function with the DIALOG_PICK_PUSH event and the label of the push button that was picked.

The push button text has "(space)<(space)" on the left and "(space)>(space)" on the right. Allow for these in the row and column specification.

Radio buttons

A radio button is displayed as on or off. When on, it appears as:

```
(•) Radio button label
```

When off, there is no dot between the parentheses.

The specification syntax for a radio button includes the button's group name, row and column positions within the dialog box, and the control's label.

```
Group "Group1"
  Radio (5, 3) "Radio 1"
  Radio (6, 3) "Radio 2"
  Radio (7, 3) "Radio 3"
EndGroup
```

The Group/EndGroup directives and the group label are required. The group label identifies a group of radio buttons and must be unique for each separate group of radio buttons within a dialog box. A group label can use the characters `A` to `Z`, `a` to `z`, `0` to `9`, `$` and `_` and may be up to 32 characters long. It must be enclosed in double quotes.

The user chooses a radio button by pressing *Spacebar* or clicking the mouse when the mouse cursor is over the radio button. The Dialog Manager will then call your action function with the DIALOG_PICK_RADIO event, the label of the radio button that was picked, and the group label of the radio button. The Dialog Manager does not automatically turn radio buttons on and off. Your action function must turn the chosen radio button on by using one of the following two functions.

```
void _dialog_check_radio (string button, ~string group)
```

This function checks the radio button specified by the button parameter and removes the checkmark from all other radio buttons in the group specified by the group parameter.

```
string _dialog_radio_checked (~string group)
```

This function determines which radio button of the group specified by the group parameter is on. The return value is the button label of the "on" button.

C H A P T E R

5

Flags and variables

The startup features in this section let you change some of the defaults in BRIEF. You can specify how you want BRIEF to handle tabs, backup files, the amount of memory allocated to files, and more.

Most of the options discussed in this chapter are automatically defined by SETUP. This explanation is provided if you need better control than SETUP provides or if you want to understand what SETUP does.

For complete instructions for all SETUP screens and prompts, see "Complete installation" in Chapter 1.

Flags and the command line

Flags are specified on the command line:

```
b file name.txt -t -kb -M50 -m"goto_line 10"
```

You can also set flags with the BFLAGS environment variable, which causes BRIEF to automatically use the flags specified in BFLAGS each time BRIEF is invoked. Environment variables are discussed following the flag descriptions.

It is unlikely that you will ever need to use *all* of these flags, but you should glance over them to see if any are relevant to you. You will notice that some flags adjust BRIEF to your personal preference, while other flags are required with specific hardware or software.

Remember that SETUP will handle most of these flags for you. For normal use, you won't have to set them yourself.

Flags are used with the `b` command when invoking BRIEF. Unlike the choices you make during SETUP, flags specified on the command line alter the editing environment only for one editing session; they do not remain in effect when you exit the session and invoke a new one.

Flags can be divided into the following categories:

- Editing environment flags
- Hardware and software compatibility flags
- Device driver flags
- Macro flags

Examples and descriptions of flags are provided in the following sections. All of these examples assume that no BFLAGS variable is set unless otherwise specified.

Flags are case-sensitive. Enter them exactly as they are shown. For example, the flags `-m` and `-M` have different functions.

Toggles

Some of the flags described here are toggles, which are flags that reverse effect each time they are specified. For example, if you specify the same toggle flag twice on the command line, the second time will "toggle" the first, and the net effect will be the same as if neither flag were present. If you specify the toggle flag three times, it will have the same effect as specifying it once. This property lets you override permanent settings made by SETUP or by your BFLAGS variable (explained later in this chapter).

The flags in the BFLAGS variable are processed before the flags on the command line, so for non-toggling flags, the flags specified on the command line take precedence.

Editing environment flags

Tab fill toggle *-t*

Controls which characters (tabs or spaces) are inserted when the *Tab* key is pressed. Normally, if `-t` isn't specified, tabs (and remaining spaces, if necessary) are inserted. If `-t` is specified, only spaces are inserted. The number of spaces inserted is equivalent to the current tab setting.

Substituting spaces for tabs can make lines longer, because a tab character (independent of the tab setting) is a single character. Each of the multiple spaces required to replace a tab character is stored separately.

You can control whether the *Tab* key inserts tab characters or spaces in three ways:

- During SETUP, specify "Fill empty areas with" tabs or spaces
- The Use Tab Characters command
- The `-t` flag

Example:

```
b -t
```

In this example, if the tab stops are eight spaces apart, BRIEF will insert just enough spaces to get to the next tab stop, instead of inserting a literal tab character.

Backup control toggle
-b

Turns automatic backup on or off. The `-b` flag reverses the default in SETUP (or in your BFLAGS) for automatic backup. BRIEF automatically creates backup files. The `-b` flag can be used to override automatic backup for one editing session. The **Backup File Toggle** command toggles backups from inside BRIEF.

Backup files are stored in the directory specified by the BBACKUP environment variable (see "Environment variables"), which is normally defined by SETUP.

Example:

```
b -b
```

In this example, if the default in SETUP was set to not create backup files (and hence BFLAGS contains `-b`), the `-b` overrides that, and a backup file is saved.

End of file character toggle
-z

Sets end of file character. During SETUP, you can choose whether BRIEF terminates a file with an EOF (*Ctrl+Z, Ctrl+Z,* ASCII 26) character or with nothing. Terminating with *Ctrl+Z* ensures file compatibility with versions of DOS prior to 2.0. If you do not want your files terminated in this manner, specify this flag.

Example:

```
b -z
```

Cursor values
-c

Sets cursor appearance. During SETUP, you can select the shape of the cursor in each possible situation: insert mode, insert mode in virtual space, overstrike mode, and overstrike mode in virtual space. The following cursor shapes, numbered 1 to 4 from left to right, are possible:

Possible cursor shapes

To use this flag, specify the number of the cursor you want for each of the four modes in the following order, separated by commas: Insert Mode, Insert Mode/Virtual Space, Overstrike Mode, Overstrike Mode/Virtual Space. The SETUP default is 1,2,3,4.

If you specify `-c` with no digits, the effect is the same as specifying:

```
-c1,1,3,3
```

Example:

```
b -c3,4,1,2
```

This example would switch the insert and overstrike mode cursors.

Number of undoable commands per buffer
-unum

Sets the number of undoable commands. The default value is 30, but you can specify an integer from 1 to 300. The size of the number has absolutely *no* effect on BRIEF's speed or on the total amount of memory needed by BRIEF. However, if BRIEF starts to run low on memory, it will purge the undo list.

Example:

```
b -u50
```

The current buffer has priority over all others when undo information is saved. If a high `-u` value is used, you may be able to undo a much smaller number of commands in other buffers.

Swapping toggle—*DOS*
-M

Controls whether or not BRIEF steps aside before running DOS programs. When swapping is off (the default), programs that are run from within BRIEF have a greatly reduced amount of memory in which to work. This is because BRIEF can only give these programs the largest block of memory that it's not using for itself.

With swapping enabled, BRIEF makes a special effort to "step aside" and frees most of its memory for use by other programs by swapping a large portion of itself to EMS memory or to disk.

Since a lot of data is swapped out, the speed of the swapping operation depends on the speed of the media BRIEF swaps to. Swapping is fastest to EMS memory, followed closely by swapping to a RAMdisk. It takes substantially longer to swap to a hard disk (especially a slow hard disk); swapping to a floppy disk should only be considered by those with patience to spare.

Who should use swapping

Swapping should be used by those who have enough EMS memory, RAMdisk space or a fast hard disk and need to run programs from within the editor. There are no real disadvantages to swapping, except for the time it takes to swap to a slow device. In fact, since you can run larger programs inside BRIEF without having to use the `-Mnum` option (which slows BRIEF down; see below), there may be a significant speed *advantage.*

Swapping storage requirements

The storage device you're swapping to must have at least as much free space as CHKDSK reports free memory at the DOS prompt. This means that if CHKDSK reports 458,752 bytes free before BRIEF is loaded, you'll need about that much EMS memory or disk space for swapping to work.

When swapping is enabled, BRIEF first checks to see if you have enough (or any) EMS memory. If you don't, BRIEF tries to swap to the BTMP directory and, if that doesn't exist, the TMP directory (see "Environment variables"). If BRIEF can't swap to either EMS or disk, it runs the program as if swapping were disabled.

Warning: When swapped to disk, BRIEF saves crucial information in a file named `BR-xxxx.SWP` in the BTMP or TMP directory, where `xxxx` is a four-digit hexadecimal number. The information in this file is *extremely* important. If the swap file is tampered with or deleted, your editing session ***will definitely be lost***, so please be careful.

Example:

```
b -M
```

Maximum memory for file storage
-Mnum

This flag allows you to select the maximum amount of memory (in kilobytes) that BRIEF allocates for file storage. Any extra parts of files will be read in from disk as they are needed. Usually, this flag is only used if swapping is not enabled and more memory is needed to run a program inside BRIEF.

The *num* specified must be a number greater than or equal to 20, such as `-M40`.

SETUP does not let you specify a value for this parameter directly; it just sets it as necessary to run your compiler inside BRIEF.

Specifying this option does *not* affect the size of the file that you can edit. As the number you specify decreases, BRIEF has to read from the disk more often. This makes BRIEF run substantially slower. If you need to run large programs inside BRIEF, we strongly recommend that you enable swapping.

DOS

BRIEF allocates memory as follows. First, about 90K is reserved for BRIEF's program code. An additional 64K, or one-third of the remaining memory, whichever is smaller, is reserved for BRIEF's stack, heap, and static data. Half of the remaining memory, up to the amount specified by the *-Mnum* option, is reserved for macro and file storage. The rest of memory is released to DOS, and is used as needed to store changes to the file and run programs inside BRIEF.

Example:

```
b -M40
```

Maximum line length
-llength

BRIEF's maximum line length can be set during SETUP. The default is 144 characters.

BRIEF will *never* truncate a line that has not been modified, no matter how long it is. However, no *modified* line can have more than the maximum number of characters (tab characters count as one character), counting the newline character at the end of each line. If you routinely work with very short or very long lines, use this option to set a new line length from 80 to 512 characters. When you extend the length of the line, BRIEF uses more memory, and you may have to save your file much more often.

For the most satisfactory results, set the line length only to the length that you need. Borland recommends a practical limit of 256 characters.

Example:

```
b -l160
```

Sets the line length to 160 characters.

Number of remembered keystrokes
-Knum

This flag lets you select the maximum number of keystrokes remembered by the **Remember** command. The default number is 350, but you can increase this number up to 2048. This flag is *not* set by SETUP.

Example:

```
b -K500
```

Sets the maximum number of keystrokes to 500.

Hardware and software compatibility flags

Keyboard compatibility toggle
-k

If your keyboard does not seem to be working properly (for example, BRIEF hangs when you hold down an arrow key), you may need to specify the -k flag. This flag helps make BRIEF more compatible with IBM PC near-compatibles (like the IBM 3270 PC).

Use of this flag disables BRIEF's keyboard speedup logic, which is controlled by the -k*num* flag (see below).

Keystroke repeat rate
-knum

This flag sets the speed of keystroke auto-repeating. *Num* is an integer that can range from 0 to 127 and specifies the speed of auto-repeat. If *num* is 0, keys will not auto-repeat. For values greater than 0, the higher the value, the faster the auto-repeat. At some point, depending on the speed of your processor, increasing *num* will have no effect. The default value for *num* is 3.

num represents the maximum number of keystrokes that will repeat each 1/18th of a second.

Refreshing rate control toggle—*DOS*
-r

You *must* use this option with a 3270 PC, and may want to use it if you have a redesigned graphics board that doesn't have "snow," such as the COMPAQ or the IBM Enhanced Graphics Adapter. -r tells BRIEF to refresh at full speed, instead of slowing down refreshing to prevent "snow" on the screen. If you're not sure about whether to use this option, try the snow test in SETUP.

Video page compatibility toggle—*DOS*
-p

If you have one of the following installed:

- A color graphics card or a 3270 PC
- Multitasking system (such as DESQview)
- Other resident software (such as SideKick, SuperKey, or Turbo Lightning)

the cursor may not appear when you invoke BRIEF. This happens with certain pieces of hardware or software that do not support multiple video pages.

Normally, on a color monitor, BRIEF uses the second video display page (page 1) to display your text, so that your DOS screen is preserved. The -p option tells BRIEF to use the first display page (page 0) instead of the second, so your cursor should appear properly.

Faster CPU toggle—*DOS*
-f

If you are using a Color Graphics Adapter and notice snow on the screen, you're probably using a processor that runs faster than the normal 4.77 Mhz. This flag stands for "fast processor" and removes the snow caused by this situation. This flag may slow down screen refreshing if your processor runs at less than twice the speed of the PC.

Display screen width
-Cnum

BRIEF's default display width is 80 columns, but if your display has more, and, under DOS, if it's a memory-mapped display with contiguous memory, you can modify BRIEF for compatibility. Specify an integer from 80 to 255 for the number of columns you want displayed. The display must be in the proper mode before running BRIEF. SETUP does not set this flag.

BRIEF's default display width is 80 columns, but if your display has more, and, under OS/2, if the additional columns are recognized, you can modify BRIEF for compatibility. Specify an integer from 80 to 255 for the number of columns you want displayed. SETUP will set this flag, if desired.

It is not necessary to set this flag when using a device driver that automatically handles wider screens, such as `ega.drv`, `hercplus.drv`, and `wyse700.drv`.

Display screen length
-Lnum

BRIEF's default display length is 25 lines, but if your display has more (or less) than 25 lines, and, under DOS, if it's a memory-mapped display with contiguous memory, you can modify BRIEF for compatibility. Specify an integer from 10 to 127, which represents the number of lines you want displayed. The display must be in the proper mode before running BRIEF. SETUP does not set this flag.

BRIEF's default display length is 25 lines, but if your display has more (or less) than 25 lines, and, under OS/2, the additional lines are recognized, you can modify BRIEF for compatibility. Specify an integer from 10 to 127, which represents the number of lines you want displayed. SETUP will set this flag, if desired.

It is not necessary to set this flag when using a device driver that automatically handles longer screens, such as `ega.drv`, `hercplus.drv`, and `wyse700.drv` or that is modified with EGA or VGA.

Window border toggle
-Bnum

The *-Bnum* startup flag toggles whether or not BRIEF's window borders are displayed, where *num* is the background color number of the initial window when borders are off. Note that *num* is optional; if not specified, the first window color is black (0).

When borders are off, there is no visible distinction between the various windows on the screen, except that the color of each window is different. If you are using a monochrome display, this option is not especially useful when more than two windows are on the screen.

Load device driver
-Dname

Loads a device driver, which is a program that tells BRIEF how to manipulate the devices attached to your computer. `Name` is the name of a device driver file, which must be in either DOS PATH or OS/2 LIBPATH. If you are running DOS, it should preferably be in the same directory as `b.exe`.

If the driver's file extension is `.drv` under DOS or `.dll` under OS/2, it need not be specified.

DOS

Included with BRIEF are device drivers for the Enhanced Graphics Adapter (EGA) and VGA, Hercules Graphics Plus and Incolor cards, Wyse 700 (and Amdek 1280) high resolution displays, and the Tandy 2000 computer. In addition, a special driver is included that allows BRIEF to run in a window (or in the background) under DESQview and Omniview. SETUP can automatically configure BRIEF to use the EGA/VGA, Hercules, and Tandy 2000 drivers.

DOS

These drivers are all sensitive to the type of display present in the system. If, for example, you have both an EGA and a Hercules Incolor Card, you can use `-Dega` and `-Dhercplus` at the same time. The drivers will determine which display is active and adjust themselves accordingly.

DOS

If you plan to switch display modes while in BRIEF, and you are using multiple device drivers, it is important to have them set up for the same number of lines and columns. For example, it is not legal to use an EGA in 43x80 mode and a Hercules Graphics Plus card in 43x90 mode during the same BRIEF session. You can, however, use both modes in different BRIEF sessions.

OS/2

The only device driver included with BRIEF is for the 101 key extended keyboard.

OS/2

When `101key.dll` is used, BRIEF automatically changes its default key assignments to free the numeric keypad. All of BRIEF'S *Shift+Keypad* assignments are changed to *Alt+Dedicated Key* on the separate cursor pad.

The following section describes how to use these drivers.

EGA/VGA device driver—*DOS*
ega.drv

To use the EGA/VGA driver, specify the `-Dega` startup flag. The EGA/VGA driver allows BRIEF to automatically use the EGA/VGA's extended line and column capabilities.

If you are in the normal 25×80 text mode when you enter BRIEF, the EGA driver will set BRIEF to use the 43×80 mode if you're using an EGA, or 50×80 if you're using a VGA.

If the device is in a text mode other than 25×80, the driver will automatically adjust the screen dimensions accordingly.

When you leave BRIEF, your initial screen mode will be restored.

This device driver expects the EGA to be in BIOS modes 0, 2, 3, 7 or a mode greater than 0×16, and determines the number of lines and columns on the screen with standard BIOS function calls. If your device is in an extended mode and does not follow these conventions, BRIEF won't recognize this fact, and will reset to either 43×80 or 50×80 instead.

EGA/VGA cursor emulation toggle—*DOS*
-E

If the EGA/VGA driver has been loaded with `-Dega`, the `-E` flag can be used to suppress BRIEF's special EGA cursor handling. This flag should only be used if the cursor looks strange or does not appear when using the EGA driver.

This flag is automatically selected for VGA devices; specifying it on the command line will have no effect.

EGA/VGA driver toggle—*DOS*
-e

If the EGA/VGA driver has been loaded with `-Dega`, the `-e` flag can be used to disable it.

Hercules Graphics Plus (and Incolor) device driver—*DOS*
hercplus.drv

This driver takes advantage of the Hercules RamFont mode, and allows BRIEF to display screens of 43 lines by 80 or 90 characters. If the Hercules card is already in the selected mode when BRIEF is started, it will remain in the proper mode when you leave.

To take advantage of this feature, you need to have the `8x8.fnt` file in the same directory as the driver. These files are copied automatically by the SETUP program.

Hercules Graphics Plus (and Incolor) width toggle—*DOS*
-H

If the Hercules Graphics Plus driver has been loaded with `-Dhercplus`, the `-H` instructs BRIEF to display 90 columns of text rather than the usual 80 columns.

Hercules Graphics Plus (and Incolor) driver toggle—*DOS*
-h

If the Hercules Graphics Plus driver has been loaded with `-Dhercplus`, the `-h` flag can be used to disable it.

Tandy 2000 device driver—*DOS*
tandy.drv

This driver allows BRIEF to run on the Tandy 2000 computer. It compensates for the hardware and software differences between the Tandy 2000 and true IBM compatible computers. If you have a Tandy 2000, SETUP will detect this fact and install the Tandy 2000 driver for you.

Wyse 700 (and Amdek 1280) device driver—*DOS*
wyse700.drv

This driver takes advantage of the high resolution modes of the Wyse 700 and Amdek 1280, and allows BRIEF to display screens of 50 lines by 80 or 160 characters. If the Wyse 700 or Amdek 1280 is already in the selected mode when BRIEF is started, it will remain in that mode when you leave.

To take advantage of this feature, you need to load the `screen.exe` program provided with your high resolution display. See your hardware manual for more information on the screen program.

Wyse 700 (and Amdek 1280) width toggle—*DOS*
-W

If the Wyse 700/Amdek 1280 driver has been loaded with `-Dwyse700`, the `-W` flag instructs BRIEF to display 160 columns of text rather than the usual 80 columns.

Wyse 700 (and Amdek 1280) driver toggle—*DOS*
-w

If the Wyse 700/Amdek 1280 driver has been loaded with `-Dwyse700`, the `-w` flag can be used to disable it.

DESQview/Omniview device driver—*DOS*
view.drv

This driver allows BRIEF to run in a window under DESQview, and in the background when using Omniview.

Extended Keyboard device driver
101key.drv—DOS
101key.dll—OS/2

This driver allows BRIEF to recognize the additional keys available on the Extended (101 key) keyboard.

When this file is used, BRIEF automatically changes its default key assignments to free up the numeric keypad. All of BRIEF's *Shift-Keypad* assignments are changed to *Alt+Dedicated Key* on the separate cursor pad.

Flags for macros

Running macros
-mmacro

-m loads the named macro file before reading in any files, and tries to run a macro with the same name after loading in all files (and running the startup macro). If -m is used more than once on the command line or in BFLAGS, *all* of the specified macros are loaded and run. The macros in BFLAGS are run first, followed by the macros specified on the command line. Within BFLAGS and the command line, the macros are run in left-to-right order. See the *Macro Language Guide* for examples of extending BRIEF with the macro language.

Example:

```
-mMDS
```

Loads the compiled macro file `mds.cm` and executes the MDS macro, which contains the customization commands from SETUP for a user with the initials MDS.

You can pass parameters to macros executed on the command line by enclosing the macro name and parameters in quotes. For example, if you wanted to start BRIEF, load the file `test.c`, and go to the 12th line in the file, you'd use the following command line:

```
b -m"goto_line 12" test.c
```

Idle time indicator
-isecs

Specifies the amount of time (in seconds) BRIEF must be idle before executing the idle action macro(s). By default, SETUP sets this value to 120 seconds for the automatic file saving feature (see "Complete installation" in "Installing BRIEF" [Chapter 1]). Idle time macros are discussed in the *Macro Language Guide.*

Example:

```
b -i90
```

Sets the idle time indicator to 90 seconds.

Environment variables

Configuring BRIEF from a batch file

Environment variables let you specify information to BRIEF from a batch file, such as

DOS `autoexec.bat`

OS/2 `os2init.cmd, config.sys, config.os2`

By putting SET commands in one of these files, you don't need to specify your flags every time you run BRIEF. In addition, you can set options like a path for BRIEF to find BRIEF macros, or a default compiler and default file name.

SETUP can automatically modify the appropriate file, so in most cases you won't need to change the settings outside of SETUP.

To set an environment variable, use the SET command as shown in the examples below. Note that you *cannot* have any spaces on either side of the equals sign, and that the right hand side of the equals sign is case-sensitive. Also, any environment variables you set when you **Suspend BRIEF** (*Alt+Z*) are cleared when you return to BRIEF. Always **Exit** completely before setting environment variables.

BFLAGS Specifies the flags to be in effect during any BRIEF editing session. You can set BFLAGS to one or more BRIEF flags.

Example 1:

This example sets the number of undoable commands to 50 and loads the EGA/VGA device driver:

DOS `SET BFLAGS=-u50Dega`

OS/2 `SET BFLAGS=-u50D101key`

Example 2—*DOS*:

This example is a typical BFLAGS value created by SETUP, which would force BRIEF to use only the first video page, avoid slow "snow" refreshing, set the idle time indicator to 120 seconds, and run the MDS initials macro:

```
SET BFLAGS=-i120pr -mMDS
```

Example 3—*OS/2*:

This example is a typical BFLAGS value created by SETUP, which would tell BRIEF to use 43 line mode, set the idle time indicator to 120 seconds, and run the MDS initials macro:

```
SET BFLAGS=-i120L43 -mMDS
```

Backup file directory *BBACKUP* Specifies the directory that BRIEF should use for backup files. If BBACKUP is not set, backups are stored in `\brief\backup` on the current drive.

If BBACKUP is set, but the specified directory does not exist, it is created.

You can direct files to a directory relative to the current directory using a relative path specification. Relative path specifications do not start at the root directory (\).

Example 1:

```
SET BBACKUP=backup
```

Since the BBACKUP setting does not start at the root, it is taken to be relative from the current file's directory. If you were editing a file in `\mine`, your backups would appear in `\mine\backup`.

Example 2:

```
SET BBACKUP=.
```

In this example, a file called `test.c` would be backed up as a file called `test.bak` in the same directory. When the backup file is created in the same directory as the file, the backup file has the extension `.bak`.

You can specify a non-relative (absolute) directory by starting the path at the root (\).

Example 3:

```
SET BBACKUP=\reserve
```

In this example, backup files would be put in the `\reserve` directory on the same drive as the file.

You can also make BBACKUP even more specific by specifying a drive letter. If no drive letter is specified, backup files are always created on the same drive the file is on.

Example 4:

```
SET BBACKUP=c:\backup
```

With this BBACKUP setting, backup files would always be placed in the `\backup` directory of the `c:` drive.

Macro search path
BPATH

Sets the path that BRIEF searches to find macro files. By default (if no BPATH is set), BRIEF searches `\brief\macros`. This search starts with the current directory. If the macro is not found there, BRIEF searches `\brief\macros` on the default drive.

Example:

```
SET BPATH=c:\macros;a:\brief\macros
```

In this example, the `\macros` directory on the `c:` drive would be searched first when loading macros. If the macro file could not be found there, BRIEF also checks the `\brief\macros` directory on the `a:` drive. The current directory is *not* checked in this example.

Help file location
BHELP

Sets the directory that BRIEF looks in for help files. You can set BHELP to any directory, as long as that directory contains the help files. If BHELP is not set, BRIEF looks in `\brief\help` on the default drive.

Example:

If the help files were present in the `\help` directory, you'd set BHELP with the command:

```
SET BHELP=\help
```

The BHELP environment variable can contain only a single directory.

Macro package control
BPACKAGES

Turns on special packages, such as language support or word processing, whenever you're editing a file with the specified extensions. For information on using the programming support features, see Programming support (Chapter 6).

Example:

This example (which is equivalent to the defaults in SETUP) sets BPACKAGES to automatically use BRIEF's regular indenting features with any `.c`, `.m`, or `.asm` file, and to use BRIEF's "text formatting" package with any `.txt` or `.doc` file.

```
SET BPACKAGES=m,c,asm:r;txt,doc:wp
```

Compiler control
BCxxx

This set of variables is also related to BRIEF's programming support features by specifying which compiler command to use when a program in a particular language is being compiled from within the BRIEF editor. The `xxx` represents a file extension (for example, BCC is for `.c` files, BCM is for `.m` files). For information on using the programming support features, including more detailed information on BCxxx, see "Programming support" (Chapter 6).

Example 1:

You can set BCC to automatically use a different compiler than the default, in this case the Microsoft C compiler `cl` command. If you are typing the command at the command line, the format is:

```
SET BCC="cl -c %s.c"
```

If you are entering the command in a batch file, you must use two percent signs:

```
SET BCC="cl -c %%s.c"
```

Example 2:

The BRIEF Macro Compiler could be called with the –e option:

```
SET BCM="cm -e %s.m"
```

Default file
BFILE

Specifies a file to edit if no file is listed on the BRIEF command line. If BFILE is not set, BRIEF prompts for a file.

Example:

```
SET BFILE=newfile
```

Warning: The BFILE environment variable is used by the **restore** and PWB macros to specify where to save the session information. If these macros are being used, BFILE cannot be used for any other purpose.

Temporary file directory
TMP

Specifies the location that should be used for BRIEF's temporary files (such as the file used for swapping). The specified directory must already exist; BRIEF will not create it.

BRIEF checks for an environment variable named BTMP first. If none exists, BRIEF looks for an environment variable named TMP. If neither of these environment variables is set, BRIEF uses the root directory of the drive that was current when BRIEF was started.

Example 1:

```
SET TMP=c:\tmp
```

Example 2:

```
SET TMP=d:\
```

Many popular programs, including the Microsoft C compiler, use the TMP environment variable to specify the location of their temporary files. If you use the TMP environment variable instead of BTMP, TMP should point to a location that has enough memory to satisfy the requirements of all such programs.

Network users: The TMP or BTMP variables *must* point to a directory in which you have permission to create and write files.

Disable EMS—*DOS*
BEMS

If EMS memory is available, BRIEF uses it for file storage and swapping.

To prevent BRIEF from using EMS storage, set BEMS equal to zero. Nonzero values are ignored and do not disable EMS.

Example 1:

```
SET BEMS=0
```

C H A P T E R

6

Programming support

BRIEF is a programmer's editor, providing language packages with features like

- Automatic indenting
- Statement completion (template editing)
- Compiling programs from within the editor
- Automatic error location

Examples and practice sessions for C are provided in this section; packages for other languages work in a similar manner.

General information

Indenting and statement completion

Structured programming requires indenting and a powerful programmer's tool should make indenting easy. BRIEF goes even further by providing a feature that practically enters the statements for you: template editing.

In BRIEF, the set of features tailored for a language is called a language package and BRIEF supports most popular languages.

BRIEF's language packages provide the following features:

- Regular indenting (a useful tool for any language)
- Smart, syntax-sensitive indenting
- Template editing

Selecting a language package

You can attach these packages so that they are automatically used with program files that have specific file extensions. Attaching a language package can be done by

- SETUP's File Extensions menu
- Manually setting an environment variable at the prompt or in a batch file

Both methods set the same environment variable; the SETUP program sets it through menu choices.

BPACKAGES is the environment variable used by all file extension dependent packages, including simple text formatting (word processing) and independently marketed packages, like dBRIEF (by Global Technologies).

The format of BPACKAGES is

```
set bpackages=extension:environment variable,extension...
-equivalent
extension:package,package...;extension:package;...
```

extension is the file extension that will invoke a specific style. More than one extension can be specified for each style.

The special extension `default` is used to specify all extensions not specifically included in the BPACKAGES string.

environment variable is explained in "Flags and variables" (Chapter 5). Abbreviating the package name saves environment space; since environment variables are limited in length to 128 characters, using the abbreviation is recommended. Abbreviations are included in the following explanations.

equivalent extension specifies that the immediately preceding extensions should be treated the same as the *equivalent extension* by BRIEF's language-sensitive features.

All extensions preceding the *equivalent extension* back to a preceding semicolon, another *equivalent extension*, or the beginning of the BPACKAGES are affected by this command. So, if BPACKAGES were set to

```
set bpackages=asm:r;cpp,hpp,h-c:t
```

BRIEF would consider the extensions `.cpp`, `.hpp` and `.h` equivalent to the extension `.c`, and would use `.c` template editing for all three. Since `.asm` is separated from the other extensions by a semicolon, it is not affected. Note also that since `.c` isn't specified as an *extension*, no packages are assigned to it.

BRIEF will check for packages using the actual extension before checking for one with the equivalent extension. If, in the previous example, smart indenting and template support were actually available for `.cpp` files, BRIEF would use the `.cpp` support, not the `.c` support.

This feature is useful when BRIEF doesn't automatically recognize an extension, and isn't able to provide smart indenting, template, or other language-specific support. Using *extension equivalence*, you can tell BRIEF that an unsupported extension should be treated like an equivalent supported one.

SETUP will automatically handle the details of extension equivalence when you choose an extension's language type.

package is the macro package attached to the extension. Each extension can have multiple packages associated with it, as long as the packages don't conflict with each other.

For more information on environment variables (when to use them and others that are available) see "Flags and variables" (Chapter 5).

Each package discussed in this section provides a specific example of BPACKAGES to use for that style.

Regular indenting

Regular indenting is the basic support provided for all programming languages: when you press *Enter*, the cursor moves down one line and returns to the indenting level of the previous non-blank line. To move in (indent), you press *Tab*; to move out (outdent), you press *Shift+Tab*.

Setting BPACKAGES

You can use regular indenting with any program file by attaching the package to a specific file extension using the BPACKAGES environment variable. For example, if you want to use regular indenting in Pascal files, set the environment variable to

```
set bpackages=pas:regular
```

In the BPACKAGES environment variable, you can abbreviate `regular` to `r`.

If you want to use regular indenting with Pascal, BRIEF macro, and Modula-2 files, set BPACKAGES to

```
set bpackages=pas,m,mod:r
```

Smart indenting

Smart indenting is a step more advanced than regular indenting and is supported for many popular languages such as C, CBRIEF, C++, Pascal, Modula-2, Ada, FORTRAN, COBOL, and BASIC. Smart indenting indents for you by recognizing statements that require a change in the indenting level.

Although the following examples all describe the C language package, other languages packages work similarly.

For example, in C the `if` statement is a signal that the statement (or statements) that follow will be indented an additional level.

If you type

```
if (a == b)_
```

and then press *Enter*:

```
if (a == b)
   _
```

the cursor indents automatically, without requiring you to press any tabs.

If you type `++b;`:

```
if (a == b)
   ++b;_
```

the statement is complete. When you press *Enter* the next time

```
if (a == b)
   ++b;
_
```

the cursor outdents, returning to the same level as the `if` statement. Other languages work in a similar fashion. More examples for C follow in the "Practice" section below.

Setting BPACKAGES

Since the smart indenting package provided with BRIEF recognizes the syntactic features of specific languages, it will only work with that language's program files. BRIEF is smart enough to figure this out; if you specify an indenting type that is too "smart" for your current language, BRIEF will reduce its intelligence to the point where it will work. If, in the future, additional support is provided for the specified language, the level of support will automatically be upgraded.

To set up smart indenting for C files, set the BPACKAGES environment variable to

```
set bpackages=c:smart
```

In the BPACKAGES environment variable, you can abbreviate `smart` to `s`.

You can specify both smart indenting and regular indenting in the BPACKAGES environment variable, but not both for the same extension:

```
set bpackages=pas:r;c:s
```

Regular indenting would be set up for files with extensions `.pas`, and smart indenting set up for files with the extension `.c`. If smart indenting is not supported for the specified extension, BRIEF will use regular indenting instead.

Modifying the indenting style

Most aspects of smart indenting can be configured to correspond more closely to your own indenting style.

For example, you can change C smart indenting so it does not indent when you type an opening ({) and/or closing (}) brace. To do so, place two digits (each 0 or 1) separated by a space immediately after the `:s` in the BPACKAGES variable. The first digit is a 1 if you want to indent opening braces (0 if you don't), and the second digit is a 1 if you want to indent closing braces (0 if you don't). If the numbers are omitted, 1 is assumed for both.

For example, an `if` statement would normally be indented as

```
if (a == b)
   {
   a++;
   b++;
   }
```

If you `set BPACKAGES=c:s 0 0`, BRIEF would indent as follows:

```
if (a == b)
{
   a++;
   b++;
}
```

BRIEF's language support is quite flexible and can be configured to fit most any style, regardless of the language. See the section describing the language package you're interested in for more information.

Practice: C smart indenting

Smart indenting recognizes various uses of the flow-of-control statements that require special indenting. The exercises that follow will familiarize you with BRIEF's smart indenting capabilities for a C file.

1. Set the environment variable to register smart indenting for `.c` files
2. Invoke BRIEF; edit a `.c` file
3. Enter the line used in the example above, but this time add a brace after the condition:

   ```
   if (a == b) {_
   ```

   Press *Enter* and note that the cursor automatically moves down to the correct indenting level.
4. Now enter the line again. But this time, add the brace on the second line:

   ```
   if (a == b)
      {_
   ```

5. The cursor automatically moves to the correct indenting level after you press *Enter*. Anything you enter in between the braces remains indented. When you supply a closing brace, BRIEF automatically outdents and places the cursor directly under the `if` statement again.

Other flow-of-control statements, such as `do` and `while`, indent and outdent in a similar fashion.

Languages other than C

The other BRIEF language packages indent using similar rules, paying attention to the conventions and requirements of the given language. For example, the FORTRAN package recognizes flow of control statements, and will automatically insert a comment character when a blank line is typed (since blank lines are illegal in FORTRAN). The COBOL package will optionally reserve columns 1–6 for sequence numbers. And both the FORTRAN and COBOL packages automatically deal with the details of statement continuation and column sensitivity.

Template editing

Statement completion

Statement completion, or template editing, further adds to the features of smart indenting by automatically completing frequently used statements. Template editing finishes the names of some statements and adds the appropriate braces, parentheses, semicolons and other punctuation.

Setting BPACKAGES

For example, to set up template indenting for Modula-2 files, you would set BPACKAGES to

```
set bpackages=mod:template
```

In the BPACKAGES environment variable, you can abbreviate `template` to `t`.

Minimum abbreviation length

You can adjust the sensitivity of BRIEF's keyword recognition with a parameter specified after the `t` in the BPACKAGES environment variable.

Normally, BRIEF defaults to a one-character minimum abbreviation length. This means BRIEF will expand a template as soon as you've typed one or more letters that match the template and pressed *Spacebar*. If you find that BRIEF is expanding templates at inopportune times, increase the minimum abbreviation length to make BRIEF less sensitive.

If you set the minimum abbreviation length to 0, templates are only expanded when the *Tab* key is pressed to explicitly expand them (never when *Spacebar* is pressed).

Additional details about template editing control and parameter placement are contained in the section that deals with the language package you're interested in.

Using templates

To use a template, type one or more letters of the statement and press *Spacebar*. The statement name is completed and supplied with parentheses or braces, if appropriate. For example, if you were editing a C program and typed

```
w <Spacebar>
```

Template editing would expand this to

```
while (_)
```

The cursor is placed at the last parenthesis, where you would ordinarily type the condition of the while loop.

Using the Quote command

Alt+Q Spacebar prevents expansion. When you want to insert a character that is usually expanded, such as `w` or `d`, enter the letter and then type *Alt+Q* for the **Quote** command (see "Command overview" [Chapter 3]) before pressing *Spacebar*.

For example, to enter `w` instead of `while ()`, type

```
w <Alt+Q> <Spacebar>
```

Alt+Q { enters an unmatched open brace. C template editing automatically inserts a matching close brace when you enter `{`. More information about the C language package can be found in the "C, CBRIEF, C++" section of "Third-party packages" (Chapter 7).

Practice: C template editing

Try expanding the C language `do` statement using template editing:

1. Set the BPACKAGES environment variable to use template editing for `.c` files.
2. Invoke BRIEF; edit a `.c` file.
3. Enter the following:

   ```
   d <Spacebar>
   ```

4. Your buffer should look like this:

   ```
   do
      {
      _
      }
   while ();
   ```

5. The cursor is placed between the braces and you can immediately start typing the body of the `do` statement.

Language packages provided with BRIEF

BRIEF has extensive language package support for

Supported languages

- Ada
- BASIC
- C, CBRIEF, C++
- COBOL
- FORTRAN
- Modula-2
- Pascal

Each package supports smart, syntax-sensitive indenting and template editing, and each package is controlled through the standard BPACKAGES interface (described earlier in "Selecting a language package").

Normally, SETUP will handle the details of setting BPACKAGES. If, however, you find that the default behavior is not to your liking, the characteristics of each package can usually be changed. This is done by specifying one or more parameters after the package name in the BPACKAGES string. Smart indenting parameters are specified first, followed by template editing parameters.

The following section describes each language package's features and options. We invite you to experiment with the different packages and options.

Ada

BRIEF's Ada language package fully supports standard Ada indenting conventions, and automatically supports the `.ada` extension. Other extensions can easily be supported by selecting the Ada language type when configuring the extension in SETUP, or by manually using extension equivalence.

The Ada language package accepts the following parameters:

Smart indenting

Smart indenting for Ada is not configurable.

Template editing

Parameter number	Function
0	Specifies the minimum abbreviation length for template expansion. If 0, templates will only expand when *Tab* is pressed. Default: 1.
1	Specifies the case that should be used for template expansion. If 0, templates are expanded to all lower case. If 1, templates are expanded to all upper case. If 2, templates have a leading upper case letter, with the remainder in lower case. Default: 0.

Templates expanded

The following templates are listed alphabetically. When the initial letters of two templates conflict, the template with priority is listed first.

Note that only the first portion of the template is listed. When the template expands, additional information may also be inserted, and the cursor will move appropriately.

Also note that some templates have short forms that allow easier access to the complete template. When this is the case, the short form is listed first, followed by the full length minimum abbreviation.

Template/Keyword	Minimum abbreviation
accept	a
abort	ab
begin	b
case	c
else	e
elsif	ei
	elsi
entry	en
exception	ex
exit	exi
exit when	ew or exit w
function	f

Template/Keyword	Minimum abbreviation
for	fo
if	i
loop	l
or	o
procedure	p
package	pa
package body is	pb or package b
package is new	pn or package i
return	r
record	rec
subtype	s
select	se
type	t
task	ta
task body is	tb or task b
terminate	te
use	u
with	w
when	wh
while	whi
when others	wo or when o

BASIC

BRIEF's BASIC language package fully supports structured BASIC indenting conventions and automatically supports the `.bas` extension. Other extensions can easily be supported by selecting the BASIC language type when configuring the extension in SETUP or by manually using extension equivalence (see above).

BRIEF's BASIC support does not include automatic line number generation, as line numbers are being phased out.

The BASIC language package accepts the following parameters:

Smart indenting

Parameter number	Function
0	Controls whether or not the body of a procedure should be indented relative to its definition. Set this parameter to 1 if the body should be indented, and 0 if not. Default: 1.
1	Controls whether or not the END of a block should be indented. Set this parameter to 1 if the END should be indented, and 0 if it shouldn't. Default: 0.
2	Controls the indentation of the outermost BEGIN/END of a procedure and its declarations. Set this parameter to 1 if they should be indented, and 0 if not. Default: 0.

Template editing

Note that if template editing is configured, the smart indenting parameters must also be specified. The parameter number given reflects this fact.

Parameter number	Function
3	Specifies the minimum abbreviation length for template expansion. If 0, templates will only expand when *Tab* is pressed. Default: 1.
4	Specifies the case that should be used for template expansion. If 0, templates are expanded to all lower case. If 1, templates are expanded to all upper case. If 2, templates have a leading upper case letter, with the remainder in lower case. Default: 1.

Templates expanded

The following templates are listed alphabetically. When the initial letters of two templates conflict, the template with priority is listed first.

Note that only the first portion of the template is listed. When the template expands, additional information may also be inserted, and the cursor will move appropriately.

Also note that some templates have short forms that allow easier access to the complete template. When this is the case, the short form is listed first, followed by the full length minimum abbreviation.

Template/Keyword	Minimum abbreviation
CASE	C
DEF FN	D
ELSE	E
ELSEIF	EI or ELSEI
EXIT	EX
FOR	F
FUNCTION	FU
IF	I
ON GOSUB	O
ON ERROR GOTO	ONE
RETURN	R
SELECT CASE	S
SUB	SU
WHILE	W

C, CBRIEF, C++

BRIEF's C language package fully supports standard C indenting conventions, and automatically supports the `.c` extension. Other extensions and syntactically similar languages, such as CBRIEF and C++, can easily be supported by selecting the C language type when configuring the extension in SETUP, or by manually using extension equivalence (see above).

Previous versions of BRIEF also included C support, and this support has not changed. The order of the template indenting parameters has been altered, however. If you're upgrading from a previous version of BRIEF, and had configured smart indenting or template editing, please ensure that the parameters you have specified are in the correct location.

The C language package accepts the following parameters:

Smart indenting

When C smart indenting is turned on, open and close braces automatically reposition themselves based on the current indenting style (described below). If you want to insert a brace without having it reposition, use *Ctrl+Brace* or *Alt+Q Brace* instead.

Parameter number	Function
0	Controls the placement of an opening brace. If 0, the opening brace is not indented with the code it encloses. If 1, the brace is indented along with the code. Default: 1.
1	Controls the placement of a closing brace. If 0, the closing brace is not indented with the code it encloses. If 1, the brace is indented along with the code. Default: 1.
2	Controls the placement of the set of braces enclosing a procedure. If 0, and the first two parameters are 1, the first brace is outdented. If 1, the first braces are treated just like any other opening or closing brace. Default: 0.

Template editing

Note that if template editing is configured, the smart indenting parameters must also be specified. The parameter number given reflects this fact.

Parameter number	Function
3	Specifies the minimum abbreviation length for template expansion. If 0, templates will only expand when *Tab* is pressed. Default: 1.

Templates expanded

The following templates are listed alphabetically. When the initial letters of two templates conflict, the template with priority is listed first.

Note that only the first portion of the template is listed. When the template expands, additional information may also be inserted, and the cursor will move appropriately.

Also note that some templates have short forms that allow easier access to the complete template. When this is the case, the short form is listed first, followed by the full length minimum abbreviation.

Template/Keyword	Minimum abbreviation
break	b
case	c
continue	co
do	d
default	de
else	e
else if	ei or else i
for	f
if	i
return	r
switch	s
while	w

COBOL

BRIEF supports the keywords and indentation conventions of both the ANSI 1985 and ANSI 1974 Cobol standards. Special support includes:

- Optional reservation of columns 1–6 for sequence numbers
- Automatic placement of division, section, and paragraph names
- Automatic placement of FD, RD, and level 01 data descriptors
- Data descriptors with higher level numbers than that of the preceding line are automatically indented to the right; those with lower numbers are automatically indented to the left
- Automatic conversion of tabs to spaces, due to the column sensitivity of the language
- Automatic statement wrap past column 72
- Automatic comment wrap past column 80

The `.cob` extension is automatically supported; other extensions can easily be added by selecting the Cobol language type when configuring the extension in SETUP, or by manually using extension equivalence (see above).

The Cobol language package accepts the following parameters:

Smart indenting

Parameter number	Function
0	Controls whether or not columns 1–6 should be used for sequence numbers. Set this parameter to 1 if columns 1–6 should be reserved, 0 if not. Default: 0.
1	Controls the cursor position used for comment continuation. Specify 0 if the comment should be indented to the same level as the last code line, 1 if it should be indented to the same level as the last comment line, and 2 if it should be indented to a standard comment continuation column (column 10 if columns 1–6 are reserved for sequence numbers, and column 4 if not). Default: 0.
2	Controls the type of COBOL supported. Set this parameter to 0 for ANSI 1974, or 1 for ANSI 1985. Default: 1.

Template editing

Note that if template editing is configured, the smart indenting parameters must also be specified. The parameter numbers given reflect this fact.

Parameter number	Function
3	Specifies the minimum abbreviation length for template expansion. If 0, templates will only expand when *Tab* is pressed. Default: 1.
4	Specifies the case that should be used for template expansion. If 0, templates are expanded to all lower case. If 1, templates are expanded to all upper case. If 2, templates have a leading upper case letter, with the remainder in lower case. Default: 1.

Templates expanded

The following templates are listed alphabetically. When the initial letters of two templates conflict, the template with priority is listed first.

Note that only the first portion of the template is listed. When the template expands, additional information may also be inserted, and the cursor will move appropriately.

Also note that some templates have short forms that allow easier access to the complete template. When this is the case, the short form is listed first, followed by the full length minimum abbreviation.

Template/Keyword	Minimum abbreviation
ADD	A
ADD TO GIVING	AG
AT END	AE or AT
CALL	C
CLOSE	CL
COMPUTE	CO
DISPLAY	D
DATA	DA
DELETE	DE
DIVIDE	DIV
DIVIDE INTO REMAINDER	DG
DELETE RECORD	DR
DELETE FILE	DF
ELSE	E
ENVIRONMENT	EN
EVALUATE	EV
EXIT	EX
EXEC CICS	EC or EXEC C
GIVING	G
IF	I
IDENTIFICATION	ID
MOVE	M
MULTIPLY	MU
MULTIPLY BY GIVING	MG

Template/Keyword	Minimum abbreviation
OPEN	O
PERFORM	P
PROCEDURE	PR
READ	R
REWRITE	REW
RETURN	RET
SET	S
SEARCH	SEA
SELECT	SEL
STRING	ST
SUBTRACT	SU
SUBTRACT FROM GIVING	SG
THRU	T
UNTIL	U
UNSTRING	UNS
VARYING	V
WHEN	W
WRITE	WR

FORTRAN

BRIEF's FORTRAN language package supports the keywords and indentation conventions of the FORTRAN 77 standard. Special support includes

- Automatic insertion of the comment character when a blank line is typed (since blank lines are illegal in FORTRAN)
- Automatic conversion of a label in the text to a standard FORTRAN label
- Automatic conversion of tabs to spaces, due to the column sensitivity of the language
- Automatic statement wrap past column 72

The `.for` extension is automatically supported; other extensions can easily be added by selecting the FORTRAN language type when configuring the extension in SETUP, or by manually using extension equivalence (see above).

The FORTRAN language package accepts the following parameters:

Smart indenting

Parameter number	Function
0	Controls the cursor position used for comment continuation. Specify 0 if the comment should remain at the current indentation level, 1 if it should be indented to the same level as the last comment line, and 2 if it should be indented to the standard comment continuation column (7). Default: 0.
1	Sets the number of levels continuationlines should be indented relative to the start of the statement. Default: 2.

Template editing

Note that if template editing is configured, the smart indenting parameters must also be specified. The parameter number given reflects this fact.

Parameter number	Function
2	Specifies the minimum abbreviation length for template expansion. If 0, templates will only expand when *Tab* is pressed. Default: 1.
3	Specifies the case that should be used for template expansion. If 0, templates are expanded to all lower case. If 1, templates are expanded to all upper case. If 2, templates have a leading upper case letter, with the remainder in lower case. Default: 1.

Templates expanded

The following templates are listed alphabetically. When the initial letters of two templates conflict, the template with priority is listed first.

Note that only the first portion of the template is listed. When the template expands, additional information may also be inserted, and the cursor will move appropriately.

Also note that some templates have short forms that allow easier access to the complete template. When this is the case, the short form is listed first, followed by the full length minimum abbreviation.

Template/Keyword	Minimum abbreviation
BLOCK DATA	B
COMMON	C
DO	D
DATA	DA
ELSE	E
ELSEIF	EI or ELSEI
FORMAT	F
FUNCTION	FU
IF	I
PROGRAM	P
READ	R
RETURN	RET
SUBROUTINE	S
WRITE	W

Modula-2

BRIEF's Modula-2 language package fully supports standard Modula-2 indenting conventions, and automatically supports the `.mod` and `.def` extensions. Other extensions can easily be supported by selecting the Modula-2 language type when configuring the extension in SETUP, or by manually using extension equivalence (see above).

The Modula-2 language package accepts the following parameters:

Smart indenting

Parameter number	Function
0	Controls whether or not the body of a procedure should be indented relative to its declaration. Set this parameter to 1 if the body should be indented, and 0 if not. Default: 1.
1	Controls whether or not the END or UNTIL of a block (but not a procedure, module, or RECORD) should be indented. Set this parameter to 1 if the END or UNTIL should be indented, and 0 if it shouldn't. Default: 0.

Parameter number	Function
2	Controls the indentation of the outermost BEGIN/END of a procedure and its declarations. Set this parameter to 1 if they should be indented, and 0 if not. Default: 0.
3	Specifies the minimum abbreviation length for template expansion. If 0, templates will only expand when *Tab* is pressed. Default: 1.

Template editing

Note that if template editing is configured, the smart indenting parameters must also be specified. The parameter numbers given reflect this fact.

Templates expanded

The following templates are listed alphabetically. When the initial letters of two templates conflict, the template with priority is listed first.

Note that only the first portion of the template is listed. When the template expands, additional information may also be inserted, and the cursor will move appropriately.

Also note that some templates have short forms that allow easier access to the complete template. When this is the case, the short form is listed first, followed by the full length minimum abbreviation.

Template/Keyword	Minimum abbreviation
BEGIN	B
CASE	C
CONST	CO
DEFINITION	D
ELSE	E
ELSIF	EI or ELSI
EXPORT	EX
FOR	F
FROM	FR
IF	I
IMPLEMENTATION	IM

Template/Keyword	Minimum abbreviation
IMPORT	IMPO
LOOP	L
MODULE	M
PROCEDURE	P
REPEAT	R
RETURN	RET
TYPE	T
UNTIL	U
VAR	V
WHILE	W
WITH	WI

Pascal

BRIEF's Pascal language package fully supports standard Pascal indenting conventions, and automatically supports the `.pas` extension. Other extensions can easily be supported by selecting the Pascal language type when configuring the extension in SETUP, or by manually using extension equivalence (see above).

The Pascal language package accepts the following parameters:

Smart indenting

Parameter number	Function
0	Controls whether or not BEGIN/END blocks are indented (except for the first BEGIN/END block: see below). Set this parameter to 1 if they should be indented, and 0 if not. Default: 1.
1	Controls whether or not the contents of a BEGIN/END block are indented. Set this parameter to 1 if the contents should be indented, and 0 if not. Default: 0.
2	Controls the indentation of the outermost BEGIN/END of a procedure and its declarations. Set this parameter to 1 if they should be indented, and 0 if not. Default: 1.

Parameter number	Function
3	Specifies the minimum abbreviation length for template expansion. If 0, templates will only expand when *Tab* is pressed. Default value: 1.
4	Specifies the case that should be used for template expansion. If 0, templates are expanded to all lower case. If 1, templates are expanded to all upper case. If 2, templates have a leading upper case letter, with the remainder in lower case. Default: 1.

Template editing

Note that if template editing is configured, the smart indenting parameters must also be specified. The parameter numbers given reflect this fact.

Templates expanded

The following templates are listed alphabetically. When the initial letters of two templates conflict, the template with priority is listed first.

Note that only the first portion of the template is listed. When the template expands, additional information may also be inserted, and the cursor will move appropriately.

Template/Keyword	Minimum abbreviation
BEGIN	B
CASE	C
CONST	CO
ELSE	E
FOR	F
FUNCTION	FU
IF	I
LABEL	L
PROCEDURE	P
PROGRAM	PROG
REPEAT	R

Template/Keyword	Minimum abbreviation
TYPE	T
UNTIL	U
VAR	V
WHILE	W
WITH	WI

Compiling from within the editor

Compiling from within BRIEF means that you can write and compile programs without ever leaving the editor.

DOS

If swapping is not enabled, you'll need at least 320K of RAM (and possibly more, depending on your compiler). If it can, SETUP automatically adjusts the `-M` command line flag so you have sufficient memory to compile inside BRIEF.

Supported compilers

The SETUP program allows you to configure BRIEF for virtually every popular compiler. The following compilers are directly supported by SETUP:

- BRIEF Macro Compiler
- CBRIEF
- C Compilers: Microsoft, Lattice, Microsoft Quick C, Turbo C, Computer Innovations C86+, High C, High C 386, Manx Aztec, Zortech, Datalight, Watcom C
- Assemblers: Microsoft Macro Assembler v4.0+, OPTASM, Phar Lap 386 asm, Borland Turbo Assembler
- Pascal Compilers: Oregon Pascal-2, Microsoft, Microsoft Quick Pascal, MetaWare Professional Pascal, MetaWare Professional Pascal 386,Turbo Pascal v4.0+
- FORTRAN Compilers: Microsoft, Ryan-McFarland, Lahey (16 and 32 bit)
- COBOL Compilers: Realia, Ryan-McFarland, Microsoft, Micro Focus, mbp Visual COBOL
- dBase Compilers: Nantucket Clipper, Quicksilver, FoxBASE Plus, dBFast

- Modula-2 Compilers: Logitech Modula-2JPI TopSpeed Modula-2
- C++ Compilers: Guidelines, Advantage, Zortech
- Prolog Compilers: Arity
- Ada Compilers: Janus Ada, Meridian Ada, Alsys Ada
- BASIC Compilers: Microsoft QuickBASIC, Microsoft BASIC Compiler

Specifying the compiler command

The compiler command can be specified (and run) by

- Using SETUP, choose the **File Extensions** item from the main menu. If the file extension for your compiler is on the next menu, choose it. Otherwise, select **Add Extension** and specify your extension. Then, select the appropriate compiler from the dialog box that appears.

  This is the preferred method of setting up BRIEF for a given compiler. If you use this method, most of the following information can be ignored.

- Set the `BCxxx` environment variable. You can set the environment variable directly to the compile command, or to the name of a macro you write to support your compiler. SETUP automatically creates environment variables for each language you intend to compile.

Setting the BCxxx environment variable

`BCxxx` is the environment variable related to compiler support. The `xxx` represents the file extension of the program file. You can set this environment variable during SETUP, directly in a batch file, or from the command line.

For example, `BCC` is the environment variable for `.c` (C language) files. `BCPAS` is the environment variable for `.pas` (Pascal) files, and so on.

The format of the `BCxxx` environment variable is:

```
set BCxxx=option
```

where:

BCxxx is the `BC` prefix for the environment variable plus the `xxx`, which stands for the file extension.

option is either an unquoted macro name (for compilers you write support macros for) or the exact compiler command of an compiler that you want to use, surrounded by double quotes.

For example:

- When you set `BCxxx` to a quoted string from the command line, specify:

```
set BCASM="!tasm %s;tlink %s"
```

- When you set `BCxxx` to a quoted string from a batch file, specify (for example):

```
set BCASM="!tasm %%s;tlink %%s"
```

- When you set `BCxxx` to a macro name, specify:

```
set BCASM=macro_name
```

Notice the extra `%` signs in the second case; since `%` has a special meaning in batch files (like DOS `autoexec.bat` or OS/2 command files, like `os2init.cmd`), you must use `%%` when you want `%` as the result.

The `%s` in the compiler command gets expanded to the name of the file being compiled (without the extension) when the file is actually compiled. See the section "Adding a compiler option" later in this chapter for more information.

If your compiler requires more than one pass (as the one in the example does), separate the passes with a semicolon. Use `\;` to enter a real semicolon.

If your compile command is a batch file rather than a `.com` or `.exe` file, you must include the `.bat` extension after the command name.

Setting the compiler command in a macro

One way to set the compiler command is by creating a macro in the BRIEF macro language. Using a compile macro (rather than setting `BCxxx` directly to the compiler command) lets you abbreviate the command supplied on the command line to an environment variable.

To execute a compiler macro you have written

- Set `BCxxx` environment variable to the macro name:

```
set BCC=macro_name
```

- Execute the macro like a command without a key assignment: press *F10*, type the macro name, and press *Enter*.

Automatic error location

After a program has been compiled, BRIEF automatically locates and displays any errors that were found. The cursor moves to the first line with an error. There, you can correct the error and press *Ctrl+N* to move to the next line containing an error. To see a list of all the errors in the current file, press *Ctrl+P*.

Practice

If your computer has at least 320K RAM (or you have swapping turned on under DOS), practice using these features with the BRIEF macro compiler. Since this compiler is provided with BRIEF, this exercise will work for every user with enough memory available.

At the prompt, enter

```
b test.cb
```

Type this short BRIEF macro. You need not understand macro syntax to practice compiling, but you can easily look up the statements in the *Macro Language Guide* later:

```
void hello()
{
  message ("Hello, world!");
}
```

Now, press *Alt+F10*.

You should receive the message `Macro compiled and loaded.` When BRIEF compiles a macro, it automatically loads this newly compiled version. Therefore, you can test the **hello** macro immediately after you compile it. Press *F10* and type `hello` (press *Enter*). `Hello, world!` appears on the status line.

Now, try compiling a macro with errors, so you can use the automatic error location feature. Remove the closing quote in the second line:

```
void hello()
{
  message ("Hello, world!);
}
```

This time, when you press *Alt+F10* to compile the macro, the cursor moves to the line beginning with `message` and the status line contains the information about that line: `Unterminated string.`

Compilers that don't tell BRIEF about errors

Some compilers don't tell BRIEF that the compilation they've attempted to perform hasn't succeeded. Rather than using the convention of returning an *error code* to the calling program (as most compilers do), they return nothing. BRIEF thinks the compilation succeeded, and displays the message `Compilation successful.`

To get around this problem, BRIEF must be told that it can't rely on the compiler to tell it that warnings or errors occurred. Instead, it must check the output of the compiler for warning or error messages. If it finds any, it considers the compilation to have failed.

The two methods described below tell BRIEF to consider both warning-message-only compilations and those that produce errors to have failed. This can be quite useful if you want to ensure that your code produces absolutely no compiler diagnostics.

Modifying the BCxxx environment variable

The first possible solution is modifying the `BCxxx` environment variable to tell BRIEF that this *particular* compiler doesn't return an error code. You do this by making the first non-blank character in the compilation string an exclamation point. The exclamation point can be either inside or outside the quotation marks.

Example 1:

```
set bcc=!"cl -c %s.c"
```

Example 2:

```
set bcc="!cl -W3 -c %s.c"
```

SETUP uses this method to control this option on a compiler-by-compiler basis.

The Warnings only compile toggle command

Alternatively, you can use the **Warnings only compile toggle** command to tell BRIEF to examine the output from *all* compilers for error messages. The setting of this toggle is remembered by the **restore** macro.

Example:

```
<F10> warnings_only
```

Adding a compiler option

If you want to simply add a compiler option to one of the supported compilers, the easiest way is to use SETUP to specify the command(s) you use to compile.

If you don't want to use SETUP, create a new BCxxx variable value for the compiler command you specify, using the quoted string form of the variable.

Example:

Suppose you want to compile your Microsoft C programs using the -Gs option. If you were using the built-in support, your BCC variable was probably set as follows in your autoexec.bat file:

```
set BCC="cl -c %s.c"
```

Now, change the form of the BCC variable to reflect the addition of the -Gs option, by changing the line in your autoexec.bat file to:

```
set BCC="cl -Gs -c %s.c"
```

This would call the compiler with the following line for the file test.c:

```
cl -Gs -c test.c
```

Background compilation—*OS/2*

BRIEF fully supports OS/2 background compilation features. When a compilation is done in the background, you can continue editing while the compilation takes place. When the compiler finishes, BRIEF notifies you of the outcome by beeping and displaying a message at the bottom of the screen.

You can turn on this feature in one of two ways. First, a specific compiler command or group of commands can be made to run in the background by following each environment variable compiler command with an ampersand. The ampersand can appear either inside or outside the quotation marks. For example, to only run the Microsoft C Compiler in the background, set BCC as follows:

Example 1:

```
set bcc="cl -c %s.c"&
```

Example 2:

```
set bcc=!"cl -W3 -c %s.c&"
```

If you want to run all compiles in the background, use the **Background compilation toggle** command. The setting of this toggle is remembered by the **restore** macro.

Example:

```
<F10> bgd_compilation
```

Compilers can only be run in the background if they are specified in the BCxxx environment variable as a quoted string. Macro-based compiler commands, such as cm, will still run in the foreground.

C H A P T E R

7

Third-party packages

This chapter describes third-party packages that BRIEF supports.

It includes

- PVCS (version control)
- TLIB (version control)
- Microsoft Quick Help (quick access to help files)
- Microsoft Programmer's Workbench Support (in place of **restore**)

PVCS

Macro packages don't have to restrict themselves to language-sensitive features. A useful example that does something other than indentation or word processing is a macro package designed to integrate BRIEF with the Polytron Version Control System called PVCS. It is found in the file `pvcs.cb`.

The PVCS macro package automatically checks workfiles in and out of their corresponding logfiles by checking for the presence or attributes of a file when you attempt to edit it. If the file exists and is not read-only, no additional processing is done, and the file is handled normally. If the file doesn't exist or is marked as being read-only, BRIEF tries to **get** the file using the appropriate PVCS command.

If the **get** fails because the logfile is not found, BRIEF assumes it's not a PVCS file and continues normally. No additional processing is done. Other errors, such as locking conflicts, present a pop-up window with the error text displayed by PVCS.

If the **get** succeeds, BRIEF waits until your editing session is complete. At that point, the workfile and most recent revision in the logfile are automatically compared with **vdiff**. If they are the same, the workfile is deleted since no changes have been made. If different, BRIEF asks you if you want to check in the workfile:

Prompt:

```
Check in file name [yngq]?
```

The appropriate responses are:

- `y`

  Calls **put** to check in the workfile and prompts you for a check in comment.

- `n`

  Does not check in the workfile, which is left on disk for subsequent modification. Note that BRIEF will not **put** a workfile that it did not **get** during the current editing session.

- `g`

  The same as `y`, but also checks in any other modules checked out during the editing session without prompting.

- `q`

  The same as answering `n` to all remaining prompts. It doesn't check in any more workfiles.

You must have an installed copy of PVCS and an operating system version containing the `ATTRIB` command to use this macro package. In addition, if you're not using lock checking and plan to use the NODELETEWORK configuration parameter, you must manually mark your files read-only before BRIEF will check them out for you. BRIEF will automatically mark the files read-only again once it has checked them back in.

BRIEF will not check in a workfile that it has not checked out during a given session. If, when prompted, you choose not to check in a workfile, you will have to **put** it manually.

Turning on PVCS

The PVCS package is turned on using BPACKAGES, as are all of BRIEF's macro packages. If you want automatic PVCS processing, add the string `pvcs` to your BPACKAGES under the appropriate extensions.

Example 1:

Let's assume you keep `.c` and `.asm` files in logfiles, and would like BRIEF to check them in and out for you. You'd set your BPACKAGES as follows:

```
set bpackages=c,asm:pvcs
```

Example 2:

If you wanted template editing for `.c` files, regular indenting for `.asm` files, and automatic PVCS handling for both, you would set BPACKAGES like this:

```
set bpackages=c:t,pvcs;asm:r,pvcs
```

Note that this could also be specified as:

```
set bpackages=c:t;asm:r;c,asm:pvcs
```

Additional PVCS parameters

Normally, BRIEF's PVCS package can't tell whether or not you're using file locking, nor does it know if you're using the PVCS DELETEWORK configuration option. Because of this, BRIEF sometimes needs to take additional steps to properly check out a file.

For example, unless you tell BRIEF otherwise, it assumes you're not using locking. If, after BRIEF uses a normal **get** command to check out a workfile, it finds that the workfile is read-only, BRIEF will then try to check the file out with a lock. If BRIEF had known that lock checking was enabled, it could have skipped the first step.

As another example, assume that you're not using the DELETEWORK configuration parameter, and you're trying to edit a new `.c` file. Since BRIEF doesn't know that your workfiles would normally be left on disk read-only, it will try to check out the given file, even though it can't possibly exist.

The following parameters allow you to tell the PVCS package about your individual configuration:

Parameter number	Function
0	Controls whether or not you are using the DELETEWORK configuration parameter. Set to 0 if you're not using DELETEWORK, and 1 if you are. Default: 1.
1	Controls whether or not you're using the CHECKLOCK configuration parameter. Set to 0 if you're not using file locking, and 1 if you are. Default: 0.

The TLIB macro package

TLIB is a version control package by Burton Software Systems used to manage changes to program source code. Fully supported by BRIEF, the files `tlib.cb` and `tlib.cm` are included under `\brief\macros`.

TLIB includes the macros necessary for BRIEF to access the package and is fully documented. `tlib.cb` is included in `\brief\macros`.

If you want automatic TLIB processing, add the string `tlib` to your BPACKAGES under the appropriate extensions.

Example 1:

Let's assume you keep `.c` and `.asm` files in logfiles, and would like BRIEF to check them in and out for you. You'd set your BPACKAGES as follows:

```
set bpackages=c,asm:tlib
```

Example 2:

If you wanted template editing for `.c` files, regular indenting for `.asm` files, and automatic TLIB handling for both, you would set BPACKAGES like this:

```
set bpackages=c:t,tlib;asm:r,tlib
```

Note that this could also be specified as

```
set bpackages=c:t;asm:r;c,asm:tlib
```

The following parameter allows you to tell the TLIB package about your individual configuration:

Parameter number	Function
0	Controls whether or not you are using the DELETESRC configuration parameter. Set to 0 if you're not using DELETESRC and 1 if you are. Default: 1.

Microsoft Quick Help

The Microsoft Quick Help (QH) macro is a simple interface to the Microsoft Advisor database through the Quick Help program, which provides access to any help file. It is designed for the developer who prefers using command-line utilities and doesn't have access to the Microsoft Advisor through PWB.

The macro can be used as a command (*F10* **qh** `[word]`) or it can be assigned to a keystroke. If you use the command, the macro uses `[word]` as a parameter to the QH program.

If you only use *F10* **qh**, Quick Help will look for the word under or nearest to the cursor in the current buffer.

If you assign the macro to a key (we suggest *Ctrl+H*), the macro attempts to locate a word under or near to the cursor when you press the keystroke.

All methods look in the database for the word, which can be a library function call, data structure tag name, or key word. It then displays whatever help is available for the word.

You can paste information from the help files to your open text file. Quick Help is most often used to load examples of syntax and assignments. By using **paste** in qh, you copy the selected text to the BRIEF scrap buffer. When you return to BRIEF, press *Ins* to paste from the buffer into the file.

Quick Help can append to the paste file, so you can load numerous blocks into the QH paste buffer. Once back in BRIEF, open a new file and paste the buffer. You can then maneuver in and out of the file or load it into a new window as reference.

To use Quick Help, add an `autoload` statement to your initials macro:

Old syntax

```
(autoload "qh" "qh")
(assign_to_key "<Ctrl-h>" "qh")
```

New syntax

If setup reads new syntax or if the **initials** macro has been redone in new syntax and you don't use SETUP, use the following syntax:

```
autoload ("qh", "qh");
assign_to_key ("<Ctrl-H>", "qh");
```

If part of your screen goes blank when you load Quick Help, you need to add a `-p` flag to BFLAGS. This flag controls whether BRIEF uses first video page of the system buffer. If the flag is set, BRIEF will use only one page of display memory (it will not preserve your DOS screen).

Microsoft Programmer's Workbench Support

Microsoft Programmer's Workbench Support (PWB) is a fully integrated, mouse- and window-based development environment (bundled with Microsoft C Version 6.00).

The macro `pwb.cb`, in `\brief\macros\`, enables BRIEF to update `current.sts`, the state file maintained by PWB. BRIEF updates the shared information section of the file and adds and maintains a BRIEF-specific section.

Installing the BRIEF/PWB package

Modify your `BFLAGS=` variable by replacing `-mrestore` with `-mpwb` (which must be entered in lowercase).

BFILE variables

Your `BFILE=` variable must be set properly for PWB to work. First, answer the following questions:

- Do you have an `INIT` variable?
- Is it pointing to a single path (`INIT=c:\bin`) or multiple paths (`INIT=c:\bin;d:\c\init`)?

No INIT variable

If you don't have an `INIT` variable, then `BFILE=` should be set as

```
set BFILE=current.sts
```

This will have BRIEF and PWB look for `current.sts` in the current directory.

Single path INIT= variable

If you have an `INIT=` variable pointing to a single path, then `BFILE=` should be set as

```
set BFILE=%INIT%\current.sts
```

where `%INIT%` is replaced with the setting of the `INIT=` variable.

Multiple paths INIT= variable

If you have an `INIT=` variable pointing to multiple paths, then `BFILE=` should be set as

```
set BFILE=%INIT%\current.sts
```

where `%INIT%` is replaced with the specific path of the `INIT=` variable that contains the file `current.sts`.

BRIEF and PWB

Supported state file sections

The following sections are used and/or maintained by BRIEF:

`[shared-]`	Information shared globally between applications
`[edit-]`	Information specific to and owned by all cooperating editors
`[brief]`	Information specific to BRIEF
`[brief-history]`	BRIEF's command prompt history
`[brief-scrap]`	BRIEF's scrap buffer

(shared-)

The following items found in the `[shared-]` section are used and/or maintained by BRIEF:

`version=`	Defines the version of the state file. BRIEF only supports version 2.
`mark=`	Positional "points of interest" (debugger breakpoints or editor bookmarks). BRIEF will use and maintain `mark=edit` items for bookmarks.
`pmark=`	The primary mark. This is the last location edited by the editor or viewed by the debugger. BRIEF will use and maintain the `pmark=` item.

(edit-)

The following items found in the `[edit-]` section are used and/or maintained by BRIEF:

`srch=` Last search string. BRIEF will use this item only if the `fSrchRe=` item is set to zero. BRIEF will not maintain this item for compatibility reasons.

`fSrchRe=` Nonzero if the last search used regular expressions, else zero. BRIEF will not maintain this item for compatibility reasons.

`rpl=` Last replacement string of a search and replace. BRIEF will use this item only if the `fRplRe=` item is set to zero. BRIEF will not maintain this item for compatibility reasons.

`src=` Last object of a search and replace. BRIEF will use this item only if the `fRplRe=` item is set to zero. BRIEF will not maintain this item for compatibility reasons.

`fRplRe=` Nonzero if last search and replace used regular expressions, else zero.

`file=` May be repeated more than one time and represents the file history for the most recently active "current" window, most recently accessed files first. BRIEF will maintain a separate list of files currently being edited and will use only those files from this section that can also be found in BRIEF's section. BRIEF will maintain this item in a compatible manner.

(brief) The following items are maintained in the `[brief]` section:

`screen=` `lines cols`

The number of lines and columns BRIEF used last time. This affects window restoration. If the current number of lines and/or columns differ, then the window state will not be restored.

`toggles=` `_dir _reg_exp _block_search _t_dir _check_warnings _background`

The state of various global BRIEF settings.

`srch=` `string`

The last search string.

`src=` `string`

The last translate-from string.

`rpl=` `string`

The last translate-to string.

`file=` `name top_line top_col line col [lx by rx ty c=color]`

May be repeated. Elements enclosed in brackets are optional and will be present only for files that were visible in a window. If a file is visible in more than one window, it will be repeated for each one.

BRIEF history The **savehist** macro stores BRIEF's command-prompt history information between two section markers, `[brief-history]` and `[end-brief-history]`.

Each line in this section has a semi-colon, the comment character for the state file, prepended to avoid conflicts with other sections.

BRIEF-scrap The **scrapper** macro stores BRIEF's scrap information between two section markers, `[brief-scrap]` and `[end-brief-scrap]`.

Each line in this section has a semicolon, the comment character for the state file, prepended to avoid conflicts with other sections.

C H A P T E R

8

Command reference

The BRIEF Command reference provides descriptions and details about the BRIEF commands. The command descriptions explain the command's prompts, changes in the command in overstrike versus insert mode, and other information.

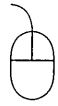
A sample command reference

The reference section lists, in alphabetical order, each of the commands available in BRIEF. The name of the command is followed by the default keystroke and a description of the actions that BRIEF takes when the command is executed.

Each command is presented in the following format:

English name *Keystroke*

command_name Description



Mouse description.

English name is the descriptive name of the command. Most BRIEF commands are always accessed by one or two keystrokes, not by the actual command name. The English name describes what the command does in one or in a few words.

Keystroke is the key assignment preset in BRIEF. If you have changed a key assignment, the listed keystroke will not apply. All key assignments can be reassigned.

command_name is the actual name that BRIEF recognizes from the command line (accessed by pressing *F10* **Execute command**). Note that you cannot use spaces in a command name; instead, underscores are used. Most commands have one command name, but a few, such as **Backspace** and **Top of buffer**, show two. The difference between the two command names is useful only when reassigning keystrokes, and is explained on a command-by-command basis.

Description is the paragraph that explains the use of the command: the prompts, appropriate responses, and examples.

Mouse icon and description is used if there is a corresponding mouse action that can replace the keystroke. The mouse icon will show if the left button, the right button, or both buttons (a logical third) should be clicked for the command.

Commands without key assignments

Commands that are not accessed with a keystroke have *F10* followed by the macro name in the *Keystroke* section.

You can assign any of these commands to keystrokes with the **Create key assignment** command.

BRIEF's command line is case sensitive, which means the commands you type at the *F10* prompt must be entered in lowercase to be properly recognized.

Many of BRIEF's commands are written in the BRIEF macro language. A macro called from the keyboard looks exactly like a built-in command. Macros, however, can be examined and altered by interested users. The compiled macro files are installed during SETUP in the directory that you specify. It is not necessary to know the macro language to use BRIEF, but it exists for those interested in customizing or modifying the editor. For information on the syntax and structure of the BRIEF macro language, see the *Macro Language Guide*.

Specifying command parameters

Prompting

Some commands need additional information to run. For example, the **Tab stops** command needs to know the tab settings that you want to use. Normally, when you invoke the **Tab stops** command by pressing *F10* and typing **tabs**, BRIEF then prompts you for the tab settings.

Command-line parameters

If you know beforehand what information the command needs, then you can specify that information immediately after pressing *F10* and typing the command name. For example, to set the tabs to every 3 spaces, you could enter

```
<F10> tabs 4 7 <Enter>
```

BRIEF recognizes the extra information, so it won't prompt you.

Ordinarily, parameters are delimited by spaces. There are, however, a few special cases:

- To include a space or spaces in a parameter, just surround the parameter with double quotes (").
- To put a double quote character in an parameter, precede it with a backslash (the override character).
- To have a string of digits interpreted as a character string, rather than an integer, put double quotes around it.

The information you enter after a prompt can be edited with *Backspace* before you press *Enter* to execute the command.

Back tab *Shift+Tab*

back_tab
slide_out

Moves the cursor to the previous tab stop without erasing tabs or characters.

See "Cursor movement" in Chapter 3 for more information.

Background compilation toggle—*OS/2* *F10* **bgd_compilation**

bgd_compilation

Toggles whether or not all compilations should be performed in the background.

See "Special commands" in Chapter 3 for more information.

Backspace

Backspace

self_insert backspace

Backspaces and erases the character preceding the cursor.

See "Editing text" in Chapter 3 for more information.

Backup file toggle

Ctrl+W

set_backup

Turns automatic backup on or off from inside BRIEF.

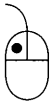
See "Saving and exiting" in Chapter 3 for more information.

Beginning of line

Home

beginning_of_line

Places the cursor at column 1 of the current line.



To use the mouse to move the cursor, double-click *Mouse Button 1* on the left arrow on the horizontal scroll bar.

See "Cursor movement" in Chapter 3 for more information.

Block search toggle

F10 **block_search**

block_search

Toggles whether or not **Search forward**, **Search back**, and **Search again** are restricted to blocks.

See "Search and translate" in Chapter 3 for more information.

Border toggle

Alt+F1

Borders

Toggles whether or not window borders are displayed.

See "Windows" in Chapter 3 for more information.

Buffer list

Alt+B

buf_list — Displays the buffer list.

See "Buffers" in Chapter 3 for more information.

Case sensitivity toggle

Ctrl+F5

search_case — Toggles upper and lower case sensitivity.

See "Search and translate" in Chapter 3 for more information.

Center line horizontally

F10 **center**

center — Centers the text on a line between the first column and the right margin.

See "Editing text" in Chapter 3 for more information.

Center line in window

Ctrl+C

center_line — Moves the current line, if possible, to the center (middle line) of the current window. This only affects the display.

See "Windows" in Chapter 3 for more information.

Change directory

F10 **cd**

cd — Changes the current working directory.

See "Special commands" in Chapter 3 for more information.

Change output file

Alt+O

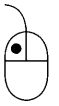
output_file — Changes the output file name for the current buffer. You cannot enter an existing file name.

See "Saving and exiting" in Chapter 3 for more information.

Change window

F1

change_window Initiates a switch from one window to another.



To position the cursor in a different window using the mouse, move the cursor to the desired window and click *Mouse Button 1*. This activates the new window, but does not position the text cursor.

See "Windows" in Chapter 3 for more information.

Color

F10 **color**

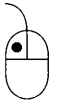
color Resets the colors used for the background, foreground, titles, and messages.

See "Color" in Chapter 3 for more information.

Column mark

Alt+C

mark 2 Starts marking a rectangular block.



To use the mouse to mark a column, press *Ctrl* and *Mouse Button 1* while moving the mouse. The selected text will be highlighted.

To extend the text selection, press *Shift* and *Mouse Button 1*. The highlighted area will move to the mouse cursor position. To continue marking text, drag the mouse or reposition the cursor and extend again.

See "Blocks and marks" in Chapter 3 for more information.

Compile buffer

Alt+F10

compile_it Compiles the file in the current buffer (and loads it if it's a BRIEF macro file).

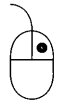
See "Buffers" in Chapter 3 for more information.

Copy to scrap

Keypad Plus

copy

Copies the block of marked characters (selected by pressing *Alt+M, Alt+C, Alt+A,* or *Alt+L* and highlighting the block with arrow keys or commands) to the scrap, replacing the contents of the scrap buffer and unmarking the block.



To copy the block with the mouse, click *Mouse Button 2* while in the window.

See "Scrap" in Chapter 3 for more information.

Create key assignment

F10 **assign_to_key**

assign_to_key

Adds a temporary key assignment to the current keyboard.

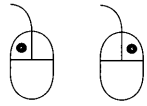
See "Special commands" in Chapter 3 for more information.

Create window

F3

create_edge

Splits the current window in half either horizontally or vertically, providing two views of the current buffer.



To create a new window with the mouse, double-click *Mouse Button 1* or *Mouse Button 2* at the left edge (for a horizontal window) or the top edge (for a vertical window). The prompts will not appear.

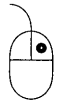
See "Windows" in Chapter 3 for more information.

Cut to scrap

Keypad Minus

cut

Copies the block of marked characters to the scrap, then deletes it, replacing the previous contents of the scrap and unmarking the block.



To delete the block with the mouse, hold *Ctrl* while clicking *Mouse Button 2* in the text area.

See "Scrap" in Chapter 3 for more information.

Delete

Del

delete_char Deletes the character at the cursor or, if a block is marked, deletes (and unmarks) the marked block.

See "Editing text" in Chapter 3 for more information.

Delete current buffer

Ctrl+Minus

delete_curr_buffer Deletes the current buffer and makes the next buffer in the buffer list the current buffer.

See "Buffers" in Chapter 3 for more information.

Delete file

F10 **del**

del Deletes a file from disk.

Warning: This command cannot be undone.

See "Special commands" in Chapter 3 for more information.

Delete line

Alt+D

delete_line Deletes the entire current line, regardless of the column position of the cursor.

See "Editing text" in Chapter 3 for more information.

Delete macro file

Shift+F9

delete_macro Deletes the specified compiled macro file from memory.

See "Macros, playback and remember" in Chapter 3 for more information.

Delete next word

Alt+Backspace

delete_next_word

Deletes from the cursor position to the start of the next word.

See "Editing text" in Chapter 3 for more information.

Delete previous word

Ctrl+Backspace

delete_previous_word

Deletes from the cursor position to the beginning of the previous word.

See "Editing text" in Chapter 3 for more information.

Delete to beginning of line

Ctrl+K

delete_to_bol

Deletes all characters before the cursor to the beginning of the line. If the cursor is beyond the end of the line, the entire line is deleted, including the newline character.

See "Editing text" in Chapter 3 for more information.

Delete to end of line

Alt+K

delete_to_eol

Deletes all characters from the current position to the end of the line.

See "Editing text" in Chapter 3 for more information.

Delete window

F4

delete_edge

Allows you to delete a window by deleting the window's edge.

To delete a window with the mouse, hold *Shift* and double-click *Mouse Button 1* on the window's close button.

See "Windows" in Chapter 3 for more information.

Display file name

Alt+F

display_file_name — Displays the name of the file associated with the current buffer on the status line.

See "Buffers" in Chapter 3 for more information.

Display version ID

Alt+V

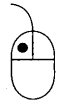
version — Displays BRIEF's version number and copyright notice on the status line.

See "Special commands" in Chapter 3 for more information.

Down

↓

down — Moves the cursor down one line, retaining the column position.



To scroll down using the mouse, place the mouse cursor on the down scroll arrow, and click or hold *Mouse Button 1*.

See "Cursor movement" in Chapter 3 for more information.

Drop bookmark

Alt+1 through *Alt+0*

drop_bookmark — Drops a numbered bookmark at the current position.

See "Blocks and marks" in Chapter 3 for more information.

Edit file

Alt+E

edit_file — Displays the specified file in the current window.

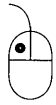
See "Buffers" in Chapter 3 for more information.

End of buffer

End End End, Ctrl+PgDn

end_of_buffer

Moves the cursor to the last character in the buffer, which is always a newline character.



To move to the bottom of the buffer with the mouse, double-click *Mouse Button 1* on the down arrow below the vertical scroll bar.

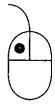
See "Cursor movement" in Chapter 3 for more information.

End of line

End

end_of_line

Places the cursor at the last valid character of the current line.



To move the cursor to the end of the line with the mouse, double-click *Mouse Button 1* on the right arrow on the horizontal scroll bar.

See "Cursor movement" in Chapter 3 for more information.

End of window

End End, Ctrl+End

end_of_window

Ctrl+End moves the cursor to the last line of the current window, retaining the column position. *End End* moves the cursor to the last character in the last line in the current window.

See "Cursor movement" in Chapter 3 for more information.

Enter

Enter

self_insert

Depending on the mode being used (insert or overstrike), either inserts a newline character at the current position, placing all following characters onto a newly created next line, or moves the cursor to the first column of the next line.

Programming packages, such as template editing and indenting, change the macro assignment for *Enter*, which alters the way the key works. See "Programming support" (Chapter 6) for more information.

See "Editing text" in Chapter 3 for more information.

Escape

Esc

Lets you cancel a command from any prompt.

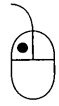
See "Help and undo" in Chapter 3 for more information.

Execute command

F10

execute_macro

Executes the specified command. This command is used to execute any command without a key assignment, such as the **Color** command.



To activate the command line with the mouse, move the cursor to the status message area and click *Mouse Button 1*.

You cannot execute macro commands that begin with an underscore, such as **_home**.

See "Special commands" in Chapter 3 for more information.

Exit

Alt+X

exit

Exits from BRIEF to DOS or OS/2.

See "Saving and exiting" in Chapter 3 for more information.

Go to line

Alt+G

goto_line

Moves the cursor to the specified line number.

See "Cursor movement" in Chapter 3 for more information.

Go to routine

Ctrl+G

routines

Displays a window that lists the routines present in the current file (if any).

See "Special commands" in Chapter 3 for more information.

Halt

Ctrl+Break

Terminates the following commands:

- **Search forward**;
- **Search backward**;
- **Translate**;
- **Playback**;
- **Execute command**;

See "Help and undo" in Chapter 3 for more information.

Help

Alt+H

help Either displays a general help menu or, if a command prompt is in the message window, displays a pop-up window of information pertaining to the command.

See "Help and undo" in Chapter 3 for more information.

Incremental search

F10 **i_search**

i_search Searches for the specified search pattern incrementally, that is, as you type it.

See "Search and translate" in Chapter 3 for more information.

Indent block

Tab (when indenting is on)

slide_in When indenting is on (see "Programming support" [Chapter 6]) and a block is marked, the *Tab* key indents all the lines in the block to the next tab stop.

See "Blocks and marks" in Chapter 3 for more information.

Insert mode toggle

Alt+I

insert_mode

Switches between insert mode and overstrike mode. *Backspace, Enter,* and *Tab* behave differently in insert mode than in overstrike mode.

See "Editing text" in Chapter 3 for more information.

Jump to bookmark

Alt+J

goto_bookmark

Moves the cursor to the specified bookmark number.

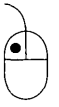
See "Blocks and marks" in Chapter 3 for more information.

Left

←

left

Moves the cursor one column to the left, remaining on the same line. When the cursor is moved into virtual space, it changes shape.



To scroll left using the mouse, place the mouse cursor on the left scroll arrow, and click or hold *Mouse Button 1.*

See "Cursor movement" in Chapter 3 for more information.

Left side of window

Shift+Home, *Alt+Home* (with 101-key keyboard)

left_side

Moves the cursor to the left side of the window.

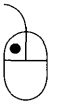
See "Cursor movement" in Chapter 3 for more information.

Line mark

Alt+L

mark 3

Starts marking a line at a time.



To use the mouse to mark a line, hold *Alt* and *Mouse Button 1* and move the mouse to the desired position.

See "Blocks and marks" in Chapter 3 for more information.

Line to bottom of window

Ctrl+B

to_bottom Scrolls the buffer, moving the current line, if possible, to the bottom of the window.

See "Windows" in Chapter 3 for more information.

Line to top of window

Ctrl+T

to_top Scrolls the buffer, moving the current line to the top of the current window.

See "Windows" in Chapter 3 for more information.

Load keystroke macro

Alt+F7

load_keystroke_macro Loads a keystroke macro into memory, if the specified file can be found on the disk.

See "Macros, playback and remember" in Chapter 3 for more information.

Load macro file

F9

load_macro Loads a compiled macro file into memory, if the specified file can be found on the disk.

See "Macros, playback and remember" in Chapter 3 for more information.

Lowercase block

F10 **tolower**

tolower Converts the characters in a marked block or the current line to lowercase.

See "Blocks and marks" in Chapter 3 for more information.

Margin

F10 **margin**

margin Resets the right margin for wordwrap, centering, and paragraph reformatting. The preset margin is at the seventieth character.

See "Editing text" in Chapter 3 for more information.

Mark

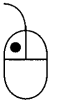
Alt+M

mark Marks a block in a buffer with no marked blocks.

When a block of text is marked, several BRIEF commands can act on the entire block:

- **Cut to scrap**
- **Copy to scrap**
- **Delete**
- **Indent block** (in files with programming support)
- **Lower case block**
- **Outdent block** (in files with programming support)
- **Print**
- **Search forward, Search backward,** and **Search again** (optionally; see the **Block search toggle** command)
- **Translate forward, Translate back,** and **Translate again**
- **Uppercase block**
- **Write**

When the **Cut to scrap, Copy to scrap, Delete, Print,** or **Write** commands are executed on a block, the block becomes unmarked.



To mark text with a mouse, hold *Alt, Ctrl,* and *Mouse Button 1* and move to the desired position. To extend the selection, position the cursor, hold *Shift* and click *Mouse Button 1.*

See "Blocks and marks" in Chapter 3 for more information.

Next buffer

Alt+N

edit_next_buffer

Moves the next buffer in the buffer list, if one exists, into the current window, making it the current buffer. The last remembered position becomes the current position.

See "Buffers" in Chapter 3 for more information.

Next character

F10 **next_char**

next_char

Moves the cursor to the next character in the buffer (if not at the end of the buffer), treating tabs as single characters and wrapping around line boundaries.

See "Cursor movement" in Chapter 3 for more information.

Next error

Ctrl+N

next_error

Locates the next error in the current file, if an error exists.

See "Special commands" in Chapter 3 for more information.

Next word

Ctrl+→

next_word

Moves the cursor to the first character of the next word.

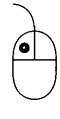
See "Cursor movement" in Chapter 3 for more information.

Noninclusive mark

Alt+A

mark 4

Equivalent to **Mark**, except that the marked area does not include the character at the end of the block.



To do a non-inclusive mark with the mouse, hold *Mouse Button 1* while moving the cursor to the desired position. To extend the selection, hold *Shift* and *Mouse Button 1*.

See "Blocks and marks" in Chapter 3 for more information.

Open line

Ctrl+Enter

open_line

Inserts a blank line after the current line and places the cursor on the first column of this new line. If the cursor is in the middle of an existing line, the line is not split.

See "Editing text" in Chapter 3 for more information.

Outdent block

Shift+Tab (when indenting is on)

slide_out

When indenting is on (see "Programming support" [Chapter 6]) and a block is marked, the *Tab* key outdents all the lines in the block to the next tab stop.

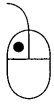
See "Blocks and marks" in Chapter 3 for more information.

Page down

PgDn

page_down

Moves the cursor down one page of text, where a page equals the length of the current window.



To use the mouse to move the cursor, place the mouse cursor on the vertical scroll bar below the box and click *Mouse Button 1*.

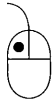
See "Cursor movement" in Chapter 3 for more information.

Page up

PgUp

page_up

Moves the cursor up one page of text, where a page equals the length of the current window.



To use the mouse to move the cursor, place the mouse cursor on the vertical scroll bar above the box and click *Mouse Button 1*.

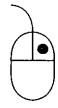
See "Cursor movement" in Chapter 3 for more information.

Paste from scrap

Ins

paste

Inserts (pastes) the current scrap buffer into the current buffer immediately before the current position, taking the type of the copied or cut block into account.



To paste text using the mouse, position the mouse cursor at the desired position in the text and double-click *Mouse Button 2.*

See "Scrap" in Chapter 3 for more information.

Pause recording toggle

Shift+F7

Tells BRIEF to temporarily stop recording the current keystroke sequence.

See "Macros, playback and remember" in Chapter 3 for more information.

Pause on error

F10 **pause_on_error**

pause_on_error

Tells BRIEF to pause when displaying run-time error messages. Otherwise, the messages flash by at a rapid rate.

See "Special commands" in Chapter 3 for more information.

Playback

F8

playback

Plays back the last keystroke sequence recorded with the **Remember** command.

See "Macros, playback and remember" in Chapter 3 for more information.

Pop-up error window

Ctrl+P

next_error 1

Displays a window of error messages and allows you to examine any message or go to the line where an error occurred. This command should be used after the current file has been compiled.

See "Windows" in Chapter 3 for more information.

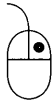
Pop-up menu

F10 **popup_menu**

popup_menu

Displays a pop-up menu. If called using the mouse, the pop-up menu is displayed centered under the mouse cursor. Otherwise, it is displayed in the middle of the screen.

The menu is in the file `\brief\help\popup.mnu`, and can be modified to add additional features.



To display a pop-up menu, place the mouse cursor where you want the menu to appear and press *Shift* and click *Mouse Button 2.*

See "Programming the mouse" in the *Macro Language Guide* for more information.

Previous buffer

Alt+Minus

edit_prev_buffer

Displays the previous buffer in the buffer list in the current window.

See "Buffers" in Chapter 3 for more information.

Previous character

F10 **prev_char**

prev_char

Moves the cursor to the previous character in the buffer (if not at the top of the buffer), treating tabs as single characters and wrapping around line boundaries.

See "Cursor movement" in Chapter 3 for more information.

Previous word

Ctrl+←

previous_word

Moves the cursor to the first character of the previous word.

See "Cursor movement" in Chapter 3 for more information.

Print block

Alt+P

print

Prints the marked block on the standard printer device.

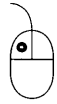
See "Blocks and marks" in Chapter 3 for more information.

Quick window switch

Shift+Arrow, Alt+Dedicated Arrow (with 101-key keyboard)

change_window

Quickly changes windows when you choose the arrow key that points to the window you want.



To position the cursor in a different window using the mouse, move the cursor to the desired window and click *Mouse Button 1*. This activates the new window, but does not position the text cursor.

See "Windows" in Chapter 3 for more information.

Quote

Alt+Q

quote

Causes the next keystroke to be interpreted literally, that is, not as a command.

See "Editing text" in Chapter 3 for more information.

Read file into buffer

Alt+R

read_file

Reads a copy of the specified file into the current buffer, inserting it immediately before the current position.

See "Buffers" in Chapter 3 for more information.

Redo *Ctrl+U*

redo Reverses the effect of commands that have been undone. New edits to the buffer cause the undo information for commands that were not redone to be purged.

Reformat paragraph *F10* **reform**

reform Reformats a paragraph, adjusting it to the current right margin.

See "Editing text" in Chapter 3 for more information.

Regular expression toggle *Ctrl+F6*

toggle_re Toggles whether or not regular expressions are recognized in patterns.

See "Blocks and marks" in Chapter 3 for more information.

Remember *F7*

remember Causes BRIEF to remember a sequence of keystrokes.

See "Macros, playback and remember" in Chapter 3 for more information.

Repeat *Ctrl+R*

repeat Repeats a command a specified number of times.

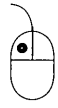
See "Special commands" in Chapter 3 for more information.

Resize window

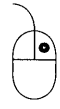
F2

move_edge

Changes the dimension of a window by moving the window's edge.



To resize a window from the top or left edge, position the mouse cursor on the top or left window edge to be moved. Hold *Mouse Button 1* to select the edge, then move the mouse until the window is the desired size.



To resize a window from any edge, position the mouse cursor on any window edge to be moved. Hold *Mouse Button 2* to select the edge, then move the mouse until the window is the desired size.

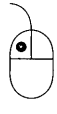
See "Windows" in Chapter 3 for more information.

Right

→

right

Moves the cursor one column to the right, remaining on the same line. If the cursor is moved into virtual space, the cursor changes shape.



To move the cursor with the mouse, position the mouse cursor on the right arrow next to the horizontal scroll bar and click or hold *Mouse Button 1*.

See "Cursor movement" in Chapter 3 for more information.

Right side of window

Shift+End, Alt+End (with 101-key keyboard)

right_side

Moves the cursor to the right side of the window, regardless of the length of the line.

See "Cursor movement" in Chapter 3 for more information.

Save keystroke macro

Alt+F8

save_keystroke_macro

Save the current keystroke macro in the specified file. If no extension is specified, `.km` is assumed.

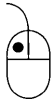
See "Macros, playback and remember" in Chapter 3 for more information.

Scroll buffer down in window

Ctrl+D

screen_down

Moves the buffer, if possible, down one line in the window, keeping the cursor on the same text line.



To scroll through the buffer using the mouse, place the mouse cursor on the vertical scroll box, press *Mouse Button 1*, and move the cursor to reposition the box.

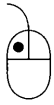
See "Cursor movement" in Chapter 3 for more information.

Scroll buffer up in window

Ctrl+E

screen_up

Moves the buffer, if possible, up one line in the window, keeping the cursor on the same text line.



To scroll through the buffer using the mouse, place the mouse cursor on the vertical scroll box, press *Mouse Button 1*, and move the cursor to reposition the box.

See "Cursor movement" in Chapter 3 for more information.

Search again

Shift+F5

search_again

Searches either forward or backward for the last given pattern, depending on the direction of the previous search.

See "Search and translate" in Chapter 3 for more information.

Search backward

Alt+F5

search_back

Searches backward from the current position to the beginning of the current buffer for the given pattern.

See "Search and translate" in Chapter 3 for more information.

Search forward

F5, Alt+S

search_fwd

Searches forward from the current position to the end of the current buffer for the given pattern.

See "Search and translate" in Chapter 3 for more information.

Suspend BRIEF

Alt+Z

dos

Exits temporarily to the operating system.

DOS

Warning: This command creates a "shell process" that runs DOS. A new copy of `command.com` is created by BRIEF. Although this shell has a reduced amount of memory, most normal commands can be executed. Please remember that BRIEF cannot perform checks on your actions when you are in this shell. If you delete the files you are editing, or if a program you execute overwrites memory that is not allocated to it by DOS (and used by BRIEF), your editing session may be lost. This command is extremely useful, but please be careful.

See "Saving and exiting" in Chapter 3 for more information.

Swap cursor and mark

F10 **swap_anchor**

swap_anchor

Exchanges the current cursor position with the mark.

See "Blocks and marks" in Chapter 3 for more information.

Tab

Tab

self_insert
slide_in

Depending on the mode being used (insert or overstrike), either inserts a tab into the buffer at the current cursor position or moves the cursor to the next tab stop.

See "Cursor movement" in Chapter 3 for more information.

Tab stops

F10 **tabs**

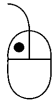
tabs — Sets the tab stops for the current buffer.

See "Editing text" in Chapter 3 for more information.

Top of buffer

Home Home Home, Ctrl+PgUp

top_of_buffer — Moves the cursor to the first character of the buffer.



To move to the top of the buffer using the mouse, place the mouse cursor on the up arrow above the vertical scroll bar and double-click *Mouse Button 1*.

See "Cursor movement" in Chapter 3 for more information.

Top of window

Home Home, Ctrl+Home

top_of_window — *Ctrl+Home* moves the cursor to the top line of the current window, retaining the column position. *Home Home* moves the cursor to the top line and the first column of the current window.

See "Cursor movement" in Chapter 3 for more information.

Translate again

Shift+F6

translate_again — Searches again for the specified pattern in the direction of the previous **Translate** command, replacing it with the given string.

See "Search and translate" in Chapter 3 for more information.

Translate backward

Alt+F6

translate_back — Searches for the specified pattern from the current position to the beginning of the buffer, replacing it with the given string.

See "Search and translate" in Chapter 3 for more information.

Translate forward

F6, Alt+T

translate — Searches for the specified pattern from the current position to the end of the buffer, replacing it with the given string.

See "Search and translate" in Chapter 3 for more information.

Undo

*Alt+U, Keypad **

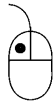
undo — Reverses the effect of the last n commands (or as many as your memory can hold). Any command that writes changes to disk (such as **Write**) cannot be reversed.

See "Help and undo" in Chapter 3 for more information.

Up

↑

up — Moves the cursor up one line, staying in the same column. When the cursor is moved into virtual space, it changes shape.



To scroll up using the mouse, place the mouse cursor on the up scroll arrow, and click or hold *Mouse Button 1*.

See "Cursor movement" in Chapter 3 for more information.

Uppercase block

F10 **toupper**

toupper — Converts the characters in a marked block to uppercase.

See "Blocks and marks" in Chapter 3 for more information.

Use tab characters

F10 **use_tab_char**

use_tab_char — Determines whether spaces or tabs are inserted when the *Tab* key is pressed to add filler space.

See "Special characters" in Chapter 3 for more information.

Warnings only compile toggle

F10 **warnings_only**

warnings_only

Forces the **Compile buffer** command to check the output from the compiler for messages. If any warning or error messages are found, the compile is considered to have failed.

See "Special commands" in Chapter 3 for more information.

Write

Alt+W

write_buffer

Writes the current buffer to disk or, if a block of text is marked, prompts for a specific file name. BRIEF does not support writing column blocks.

See "Saving and exiting" in Chapter 3 for more information.

Write all and exit

Ctrl+X

write_and_exit

Writes all modified buffers, if any, and exits BRIEF without prompting.

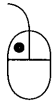
See "Saving and exiting" in Chapter 3 for more information.

Zoom/unzoom window toggle

Alt+F2, Ctrl+Z

zoom_window

If there is more than one window on the screen, **Zoom window toggle** will enlarge the current window to a full screen window, and save the previous window configuration.



To enlarge the current window using the mouse, position the mouse cursor in the top right corner of the window on the zoom button and click *Mouse Button 1*. You can also position the mouse cursor on the title bar and double-click *Mouse Button 1*.

See "Windows" in Chapter 3 for more information.

C H A P T E R

9

BRIEF error messages

There are a number of errors that could occur during your editing session. BRIEF provides a method of recovering from all but the most serious errors. Following are a list of the errors that you could encounter and the normal recovery method.

Any of the following message descriptions that has a section in angle brackets (<>) indicates that the text within the brackets will vary depending upon the nature of the error.

<n> buffers have not been saved. Exit (ynw)?

BRIEF is exiting to DOS, but some of your buffers still have changes that haven't been saved to disk. If you want to exit anyway, discarding the changes, type `y`. If you want to return to BRIEF, type `n`. If you want to write all modified buffers and then exit, type `w`.

<macro name> already defined.

A macro file was loaded containing a macro name that already exists in another macro file. The name overrides the old one, so that macro calls to the macro are directed to the new function.

<macro name> already loaded or autoloaded.

You tried to autoload a macro that was already loaded into memory, or "autoloaded" with a previous call to autoload. You can only autoload macros that aren't currently in memory, and that haven't already been autoloaded.

<name> undefined.

BRIEF encountered a reference to an undefined macro name or global variable. One likely cause is that the macro file where the macro or variable is declared has not been loaded. Another problem to watch out for is that the names of macros and variables are case-sensitive; the macro **List** is not the same as the macro **list**. If an undefined macro name was found, the message is displayed and macro execution continues after the incorrect macro call. If an undefined variable caused the problem, an error is returned to the function that referred to the variable.

<macro name>: invalid parameters.

A macro was called with incorrect parameters. Possible causes of this error are: the wrong number of parameters were specified, the wrong type of parameters (integer for string, or vice versa) were found, or an incorrect value for a parameter was given.

Access denied: RO, bad drive, file or path.

You tried to access a file that was read-only, located on a non-existent or write-protected disk drive or in a non-existent directory, or had an invalid file name. BRIEF cannot complete the function started because it cannot access the file. Remove the write-protect tab or use the **Change Output File** command to specify a new file name/directory, then retry the command. See the DOS documentation for information on legal drives, file names, and path names.

Bad characters in name.

You specified invalid characters in a file name. The following characters are illegal in DOS file names: , <> [ ] + \ " = | * ? ; and any ASCII character with a value less than or equal to 32 (a space).

Buffer not modified.

You attempted to write a buffer from the buffer list, but the buffer was not modified. Only a modified buffer may be written.

Can't compile: no BC<extension> environment variable.

No BC variable exists for the file name extension of the current buffer. The BCxxx environment variable must be set so BRIEF knows how to call your compiler (see "Programming support," Chapter 6).

Can't create menu.

BRIEF can't locate a menu file that it needs to display a menu, or the menu file that BRIEF has is invalid. Make sure that your BHELP environment variable points to the correct directory, and that the files in that directory are legitimate menu files.

Can't delete: no other buffers.

BRIEF must always have at least one buffer at any given time. If you want to delete a buffer, and there are no other buffers in memory, create a new buffer (*Alt+E* by default), and then delete the old one.

Can't play back while remembering.

You tried to play back a keystroke recording while you were making a recording. Recursive keystroke recordings (keystroke recordings that invoke themselves) are not allowed.

Can't play back while paused.
Can't remember while paused.

You tried to play back or record a keystroke recording while you were paused. Playing back or recording in the middle of a keystroke recording is not allowed.

Can't undo in mid-command.

This message appears when you try to undo a command from within a macro. Undo does not work inside of macros; it only works as a command in itself. See the description of **undo** in the *Macro Language Guide* for more information.

Changes only effective for this editing session.

You used the keys command to modify your default key assignments, but the keys macro can't find `keyboard.h`, the file it needs to make those changes permanent. This file must be on your BPATH for the keys macro to complete the operation.

Compile failed; run "cm startup". Press a key.

The keys command was unable to compile the startup macro. This usually happens because there isn't enough memory to run the compiler from within BRIEF. To rectify the situation, exit BRIEF and perform the compilation operation yourself.

Current buffer is not a .<extension> file.

You tried to run a language or file-type specific function on the wrong kind of file.

delete_macro: macro isn't loaded.

delete_macro was called with a file that isn't currently loaded, so there's nothing to delete.

Disk error. Abort (return to BRIEF), retry or ignore?

Any of a number of errors happened. The most common are

- BRIEF tried to write to a write-protected disk.
- One of the diskette drive doors was inadvertently left open.

- The diskette in the drive was not formatted.
- A bad sector was encountered on a disk.

You have three options:

- Returning to BRIEF (aborting the operation, not available with versions of DOS prior to 3.1)
- Retrying the operation in the hope that it will work the second time (usually not, unless you correct the situation)
- Ignoring the error, which will cause unpredictable results

If you can easily correct the situation, retrying is usually the best of these options.

Disk error. Enter free drive, or Esc to abort:

BRIEF tried writing to the TMP disk drive, but failed. This usually happens because the TMP directory is full or is a network directory you don't have write permission for. You must enter a drive letter for a drive where BRIEF can write temporary information in the root directory. The only other option is to abort your editing session.

Disk full, write file to another disk.

There is not enough room for BRIEF to store the file you want to save in the desired directory. The easiest way of recovering is to use the **Change Output File** command to direct the file to a disk that has more room on it, or to switch disks. You could also use the **Suspend BRIEF** command to exit to DOS, delete a few files, and then return. **Never delete a file that you are editing!**

DOS 2.0 or later required

BRIEF does not run under DOS versions prior to 2.0.

Edge does not have just two adjoining windows.

When moving or deleting a window edge, the edge must be shared by two and only two windows. If there are more or less than two windows on the edge, you will receive this error message. See the various window commands in "Command reference" (Chapter 8) for more information.

Editing changes will be lost! Abort?

This message appears when you press *Ctrl+Break* while running a macro. If you indicate that you want to abort your session, and you still have unsaved buffers, this message appears as a reminder. If you respond affirmatively, all unsaved editing changes will be lost; otherwise, the macro will resume execution.

File has not been modified -- not written.

You tried to write a buffer, but there haven't been any changes made to the buffer since the last time that it was written. If you really want to write the buffer, for some reason, make a small change, undo it, and then write the buffer.

File is not a macro.

A file was loaded as a compiled macro that wasn't actually a compiled macro file. Make sure that the file you specified for **load_macro** is a compiled `.cm` file from the *current* version of the macro compiler.

Illegal character within ().

An extra `[` or ~ was found in a set or range. If you want to use those characters literally, put a backslash (`\`) in front of them.

Illegal search across newline.

It is illegal for the <, >, `$` or `%` regular expression characters to appear in the middle of a pattern. To search across line boundaries, use `\n`.

Illegal use of |.

Either the regular expression | (or) operator was applied to a `\c` or | expression, or the expression on either side was missing.

Illegal use of \c within ().

`\c` is not allowed in a bracketed set, since a set can only match a single character. Put the `\c` outside the set, immediately adjacent to the `[` or `]` (depending on the effect needed).

Incorrect parameter found: <string / integer> expected.

A function retrieved a parameter, but it was the wrong type (integer or string, depending on the actual message you get). This message will usually be followed by an error message from the function that was retrieving the parameter, so you will know which function is having the type mismatch.

Insert <filename> disk in <drive>: and press any key...

BRIEF needs the file named `<filename>` and cannot find it. This happens if, at some point during the editing session, you changed disks. To continue, find the disk with `<filename>` on it and insert it in the specified `<drive>`. Then press any key to continue.

This error may also indicate a temporary disk drive hardware failure. If this is the case, waiting a few seconds and pressing any key will usually solve the problem.

Insert new disk in <drive>: and press any key...

BRIEF needs to write a temporary file, but the disk in the specified drive is full. To continue, insert a formatted disk with space available on it in the requested drive and press any key to continue.

Integer divide by 0
Internal error: bad access <number>, size <number>
Internal error: unable to load macro.
Internal error: undefined operation.

You should never receive any of these messages. If any of them occur, please call Borland technical support and explain the circumstances as completely as you can.

Invalid BBACKUP setting.

The BBACKUP environment variable has been set incorrectly. This variable lets you specify the directory where your backup files are created. See the "Environment variables" section in "Flags and variables" (Chapter 5) for information on BBACKUP.

Invalid BFILE specified.

The BFILE environment variable has been set incorrectly. This variable lets you set the name of the file to edit if no file name is specified when BRIEF is started. The variable must be a valid file name, optionally including a full path and/or drive letter. See the DOS documentation for more information on valid file names.

Invalid bookmark number.

The number you entered at the bookmark prompt is not valid. Valid bookmark numbers are integers from 1 to 10.

Invalid BPACKAGES environment variable.

The BPACKAGES environment variable has been set incorrectly. This variable lets you set the type of indenting allowed for each file type. See the "Environment variables" section in "Flags and variables" (Chapter 5) for more information on setting BPACKAGES.

Invalid BPATH setting.

The BPATH environment variable has been set incorrectly. This variable lets you specify where BRIEF can find your compiled macro files. It has the same format as a DOS path string. See the "Environment variables" section in "Flags and variables" (Chapter 5) for information on BPATH.

Invalid character after '\'.

A search pattern or replacement string ended with a \. If you want a literal backslash at the end of the string, use a double backslash (\\).

Invalid closure.

The @ or + regular expression characters appeared at the beginning of a pattern or group, or immediately after an @, +, <, %, >, $, \c, or | character. Closure is only relevant to matched characters, not to the regular expression operators themselves.

Invalid color: must be 0-15.

The only colors permissible on IBM compatible graphics adapters range from 0 to 15. If you specify a color number out of this range, this message appears.

Invalid color: same as background.

To prevent you from accidentally making text invisible, BRIEF won't let you set any of the foreground text colors equal to the background. If you try to do so, this message appears.

Invalid drive specified.

The drive letter you specified does not exist. To check for the presence of a drive, exit to DOS and look for files on that drive.

Invalid global reference.

A macro referred to a global variable that doesn't exist. Each global variable must be declared in one file, and that file must be loaded before macros in any other files can use it. If an invalid global reference occurs, an error code will be returned to the function that was retrieving the parameter.

Invalid number specified.

You entered an invalid number in response to a prompt asking for a numeric response (for example, `Go to line:`). Try the command again. The function that was prompting for the number receives an error code.

Invalid output file name.

The file name you gave was invalid because the file name or path or drive letter were invalid, or the output file name is the name (or the output file name) of another edited buffer.

Invalid path specification.

The path you specified is incorrect because the directories on the path do not exist, or because the directory names are invalid. See the DOS documentation for information on legal path names.

Invalid range.

A regular expression range was invalid because the ASCII value of the left side of the range was greater than the value of the right side, or the left or right side was omitted.

Macro file already loaded.

This message appears when you try to load a macro file that has already been loaded. If you want to re-load the macro file (perhaps because you have recompiled the source without exiting BRIEF), first delete the macro file with **delete_macro**, then reload it.

Maximum keystroke length reached.

You have recorded the maximum number of keystrokes allowed in a keystroke recording, and the recording has automatically been terminated. To increase the maximum number of keystrokes allowed, use the `-K` option (see "Flags and variables" [Chapter 5]).

Maximum line length reached.

BRIEF will not allow you to put more than the current *line length* (set with the `-l` option) (see "Flags and variables" [Chapter 5]) characters on a single line of text. Undoing the last command will usually recover any text that may have been "lost."

Mismatched <braces or brackets>.

The brackets or braces in a regular expression pattern did not pair up properly.

New file (unable to open <file name>).

This message appears when you create a buffer and the file in the buffer does not exist. BRIEF creates an empty buffer and when the buffer is written, the file is created automatically. This message is simply a status message, not an indication of an error.

No file specified.

A null (empty) string was passed as the file name for one of the calls that uses file names (for example, **create_buffer**, **edit_file**, **output_file**). Make sure that you always specify a file name in macro function calls if one is required, or omit the parameter entirely so BRIEF prompts for the value.

No help files found in <BHELP directory>.

BRIEF could not locate its help files. These files should be in the `\brief\help` directory on the default drive, or in the directory specified with the BHELP environment variable. Make sure the files are in the right place, and that BHELP is set properly (if at all).

No marked block.

You tried to perform an operation that works on marked areas, but you don't have any area marked. For example, you might have executed **Print Block** with no block marked.

No memory: OK to write <file name> (yn)?

If BRIEF cannot obtain enough memory to store the changes you are making to the text, it has to write one or more of the files that you are editing to disk. Once this procedure is completed, enough memory is usually freed to allow you to continue editing. If the `-a` flag was specified on startup, this procedure takes place automatically.

Note that if this write fails, BRIEF will display an error message explaining what went wrong, followed by three periods. To continue, press any key. You will then be asked to change the output name of the file. This lets you direct the file to another disk if that disk is full, or change its name if it is incorrect. If you decide that you do not want to write this particular file, press *Esc*.

No other buffers.

You tried to use the **Next Buffer** or **Previous Buffer** command, but there weren't any other buffers in the buffer list.

No previous search pattern.

You tried to search again for the previous search pattern, but there was no previous pattern to search for. **Search Again** only works if you've done a normal search earlier in your editing session.

No previous translate pattern.

You tried to repeat the previous translation, but you hadn't done one. **Translate Again** only works if you've done a translation earlier in your editing session.

No scrap to insert.

You can't insert an empty scrap. Be sure to cut or copy to the scrap before trying to insert it.

No substitution allowed in pattern.

You can't use `\<number>` in the search pattern to refer to an already defined group. `\<number>` can only be used in the replacement text.

No such group in pattern.

The replacement string referred to a group that did not exist in the search pattern.

No valid files were specified.

All of the file names you specified to edit were invalid. Any component of the file name could have been invalid: the drive letter, the path, or the file name itself. See the DOS documentation for information on legal file, path, and drive names.

No window to delete!

You tried to delete an overlapping window, but none existed. **delete_window** only works if overlapping windows have been previously created with **create_window**: one **delete_window** call should go with each **create_window** call.

Not enough space for arguments
Not enough space for environment

You should never receive any of these messages. If any of them occur, please call Borland technical support and explain the circumstances as completely as you can.

Nothing to play back.

You tried to play back a non-existent keystroke recording. To play back a recording, you must make the recording first.

Null characters in file fixed.

The only character that can't be in a BRIEF buffer is the null character (ASCII value 0). When BRIEF reads a section of a file, it checks for null characters in that section. If any are found, they are replaced with spaces (ASCII value 32). Null characters are rarely found in text files. When they are, it is usually because a program (telecommunications, or another editor) used nulls to pad a file to a multiple of 128 characters, as was required in some early versions of DOS and CP/M.

Null pattern or group.

One of the search functions was called with a null string or a string that contains a null (empty) group. Make sure that a valid pattern is specified for all search function calls.

Null pointer assignment

You should never receive this message. If it occurs, please call Borland technical support and explain the circumstances under which it occurred.

Out of memory -- recovering.

If BRIEF cannot obtain enough memory to store the changes you are making to the text, it has to write one or more of the files that you are editing to disk. Once this procedure is completed, enough memory is usually freed to allow you to continue editing.

Note that if this write fails, BRIEF will display an error message explaining what went wrong, followed by three periods. To continue, press any key. You will then be asked to change the output name of the file. This lets you direct the file to another disk if that disk is full, or change its name if it is incorrect. If you decide that you do not want to write this particular file, press *Esc*.

Out of storage; undo information was lost.

BRIEF has to keep a record of the changes you have been making in order to be able to undo them. If a command makes more changes than your machine has memory to hold, this message is displayed. There is no way of recovering from this error, but your text is not affected in any way.

Out-of-date macro. See documentation.

One or more of the compiled macro files on your disk have not been recompiled with the most recent version of `cm.exe`. All macros created with earlier versions of BRIEF must be recompiled to work with the new version. If you are using the "standard" BRIEF configuration, instead of recompiling the macros, run SETUP to install the new macros on your disk. If you've modified the macros, see the `read.me` file on disk for information on possible minor incompatibilities between the current version and previous versions, modify your macros if necessary, and recompile them.

Printer error. Abort (return to BRIEF), retry or ignore?

There is some sort of problem with your printer: usually, this error is caused by a printer being either off or off-line. You have three options:

- Returning to BRIEF (aborting the operation, not available with versions of DOS prior to 3.1)
- Retrying the operation after fixing the problem
- Ignoring the error, which will cause unpredictable results

If you can easily correct the situation, retrying is usually the best of these options.

Printer is not ready or out of paper.

You tried printing a marked area, but your printer is turned off, off line, or out of paper. Fix the problem and try again.

read: Invalid number of characters to read.

The **read** macro function was called with an invalid number of characters to read. The number of characters to read must fall between one and the maximum line length.

register_macro: specified macro not defined.

You called the **register_macro** function to register a macro, but the macro you tried to register was not loaded or autoloaded. Make sure the macro is loaded or autoloaded before calling **register_macro**.

Replacement string too long.

The replacement text, after substitution of all groups, exceeded 32,767 characters. Simplify the pattern and try again.

restore_position: no saved position.

This error message appears if you call **restore_position** without a previous matching call to **save_position**. If **save_position** is not called first, **restore_position** has no position to restore.

Save files before exiting (Esc to continue)?

If you press *Ctrl+Break* while a macro is running, this message appears. The possible responses are:

- `y` will save your files and exit to DOS
- `n` will exit to DOS without saving your files
- *Esc* will resume execution of the interrupted macro

search_string: invalid pattern.

If you specify an invalid pattern (one with syntax errors in it), **search_string** will display this message. Try running the pattern with the normal search functions to find out the exact nature of the problem.

Stack overflow

You should never receive this message. If it occurs, please call Borland technical support and explain the circumstances under which it occurred.

String length overflow -- result truncated.

An operation was performed that created a string longer than the maximum string length (144 or the maximum line length, whichever is greater). The most likely operations where this would occur are + and +=. Any excess characters greater than the maximum string length have been discarded. If this situation becomes a regular problem, you can increase the maximum line length. See "Flags and variables" (Chapter 5) or "Complete installation" for more information.

Tabs must be 2-144, in ascending order.

All tab stops must fall in the range 2-144, regardless of the maximum line length. Also, they must be specified from lowest to highest (ascending order). All tab stops after column 144 are set to the same width as the distance between the last two tab stops before column 144.

That bookmark doesn't exist.

You tried to jump to a bookmark you haven't dropped. Press *Tab* at the **Jump to Bookmark** prompt to see a menu of the valid bookmark numbers.

Too many groups.

More than ten sets of braces (`{}`) were found in the search pattern.

Too many open files (no handles left).

You've run out of DOS file handles, most likely because you're running multiple programs at the same time (with DESQview, DoubleDOS, Omniview, BRIEF inside dBASE, or other programs inside of BRIEF). To correct the situation, increase the number of FILES specified in your `config.sys` file. If you don't have a line `FILES=xx` in your `config.sys` file, insert one, with `xx` equal to 15 or more, and reboot.

Unable to autoload <file name>.

This message appears when you try to run a macro that has been autoloaded and the file name where the macro is located cannot be loaded. The most likely sources of error are that the initial **autoload** specified an incorrect file or that your BPATH is set incorrectly.

Unable to load macro file.

BRIEF was unable to load the macro file specified. Make sure the file exists, and that you specified the right file in the **load_macro** call, and try again.

Unexpected DOS error <error number>: notify BRIEF authors.

You should never receive this message. If it occurs, please call Borland technical support and explain the circumstances under which it occurred. The error number reported in the message is a DOS error code; if you want to find out exactly what the problem was, see the *DOS Technical Reference*.

Unrecoverable error, hit a key to abort:

BRIEF has encountered a severe error condition and is unable to recover. All file information should have been saved when this message appears, but BRIEF must exit to DOS. This message should never appear in normal operation; if you do see it, you probably have a software or hardware compatibility problem, or there is a bug in a macro that uses advanced macro functions.

Unrecoverable error. Save all files (yn)?

BRIEF has encountered a severe error situation, requiring it to exit to DOS. Before exiting, though, it can save all of your modified files, so that none of your work is lost. If you answer `y`, all modified buffers will be saved, and you will exit to DOS. If you answer `n`, you will exit to DOS without saving any files.

Window would be too small.

Each tiled window must have room for at least one line of text and the title of the window. You've tried to move an edge such that one of the windows sharing the edge would be too small if the move were completed. Try again, but don't move the edge as far.

You can't delete a viewed buffer.

You tried to delete a buffer from the buffer list that is being viewed by a window. Only buffers that are not viewed by any windows can be deleted in the buffer list.

You can't delete an edited file.

You attempted to delete a file with the **del** command that is being edited. This is not allowed, as it would cause BRIEF to lose information.

You can't edit a system buffer.

System buffers are reserved for use by macros for internal purposes and cannot be edited directly. You have tried to edit a buffer that was created as a system buffer. See the *Macro Language Guide* for more information on system buffers.

You can't invoke that macro from the keyboard.

Certain macros are intentionally "locked out" of the keyboard by placing an underscore before their names. This step is taken when they are not designed to be invoked from the keyboard (for example, internal subroutines in a macro file). It is a protective measure, so that you won't accidentally crash BRIEF by calling a macro in a way that it's not expecting to be called. If you are writing your own macro, and you want to be able to invoke it from the keyboard, don't start the name with an underscore.

You can't overwrite an edited file.

You tried to write a marked area to a file that is currently being edited. Since BRIEF may need to access that file at some point in the future, it won't let you overwrite it. Choose another file name for writing the marked area.

You can't read a file into itself.

The **Read File Into Buffer** command was told to read the current file into itself. This operation is illegal. If you want to insert a copy of the file, mark the entire file, copy it to the scrap, and insert it at the desired point.

You can't split an overlapping window.

Overlapping window manipulations are limited to creation and deletion. Resizing (moving edges) can only be done to tiled windows.

You must set BPACKAGES to turn on WP.

You attempted to turn on word wrap with the **wp** command. This is not allowed. Word wrap must be turned on through the BPACKAGES interface, which can be set with SETUP or, if you prefer, manually. See "Using special features" (Chapter 4) for more information.

C H A P T E R

10

Questions and answers

This chapter contains answers to some of the more common questions that are asked about BRIEF.

As a minimum, which files do I need to run BRIEF?

The most important file is b.exe, the executable file for BRIEF. You'll also need the compiled macros included with BRIEF. Keep the .cb files, which contain the macro source code, somewhere, but neither they nor SETUP need to be kept on your work disk. After you become familiar with BRIEF, you can erase the files in the \brief\help subdirectory.

Why won't SETUP respond to my commands?

You probably need to press the *Num Lock* key.

Why does BRIEF seem slow at times?

If some commands seem to run slowly, it's probably because of virtual memory. If you're using the `-Mnum` flag (see "Flags and variables" [Chapter 5]), you can usually speed up BRIEF by specifying a larger argument or by eliminating this argument altogether and using swapping instead. See the description of the `-M` flag for more details.

Can I move the help, macro, and backup files somewhere else?

Yes. The easiest way to do this is to use SETUP, which gives you complete control over the location of all these files. Alternatively, all of BRIEF's options can be controlled manually by environment variables. The DOS `SET` command is used to change these variables (generally in your `autoexec.bat` file).

- BPATH, the macro search path, defaults to `;\brief\macros`. Setting BPATH to `D:\` will tell BRIEF to look for your macro files in the root directory of the `D` disk.
- BHELP, the help directory, defaults to `\brief\help`. Setting BHELP to `C:\help` informs BRIEF that help files are in the `\help` directory of the `C` disk.
- BBACKUP, the backup directory, defaults to `\brief\backup`. Setting BBACKUP to `C:\backup` tells BRIEF to put backups in `\backup` on the `C:` disk. ("`.`" will create `.bak` files in the current directory).

See "Flags and variables" (Chapter 5) for additional information.

DOS tells me that I have no more space in the environment. What should I do?

SETUP can increase the size of the DOS environment under most versions of MS-DOS. Run it, choose a larger environment size, and then reboot: you should now have enough space. If your environment is still too small, you need to purchase DOS 3.2 or above, which allows an environment up to 32,768 bytes.

Why do I get snow on my screen?

Run SETUP and make sure that BRIEF is properly configured for your hardware.

Why does BRIEF use so much memory?

If it seems to you that BRIEF is using an unusually large amount of memory, make sure you're not nested two levels deep: you may have created a DOS shell process with *Alt+Z* and invoked BRIEF within that. If this is the case, the level indicator in the lower right corner of the window border will show a number from 2 to 9 (see "Installing BRIEF" [Chapter 1]).

If that's not the problem and you are running under DOS, remember that after subtracting a constant amount of memory for program, stack, and constant data (about 160K), BRIEF tries to allocate up to half the remaining available memory for buffer storage. You can

eliminate most of this "resident" memory by turning on swapping; this allows BRIEF to step aside when other programs are being run. You can also reduce this amount by using the -M*num* flag, although this may slow down normal BRIEF operations. Note that the M must be in upper case.

How do I reconfigure my keyboard, turn on the automatic indenting, compile within BRIEF, or do word processing?

All of these procedures are documented in the "Using special features" (Chapter 4) section. If you have specific questions about how the features work, examine the source code for the particular macro you are using. A little time spent learning to read macros will pay off in the long run.

How do I change the compile macro to work with my compiler?

You have to set the BCxxx environment variable (which you can do with SETUP) and/or modify the compilation macros. Instructions for this procedure are in "Programming support" (Chapter 6). Beware of the common stumbling blocks. First, if you don't have enough memory to run your compiler inside BRIEF (generally, at least 320K with -M30), it won't work. Also, your compiler must be in your DOS PATH, and the COMSPEC environment variable must contain the name of a valid copy of command.com (if you have no COMSPEC variable, command.com must be in the root directory).

Which programs can I run from within BRIEF?

You should be able to run almost any program that is fully debugged and doesn't use memory that is not already allocated to it by the operating system. System commands like **DIR** and **COPY** work fine. So do most compilers. It is unwise to compile a program within BRIEF and then run that program in the shell (unless you save all your files first): one little bug might destroy your editing session. Also, you cannot delete any files you might be editing.

Finally, it is very important not to run a program that remains resident in the BRIEF shell. Programs like this will take up memory that BRIEF needs to resume working and could cause BRIEF to crash.

Why does my file have right arrow characters and garbage at the end?

These characters appear in files created under early versions of DOS, and in some files downloaded from other systems. These files end with an EOF character (ASCII 26) and are padded until their length is a multiple of 128. The EOF character displays as a right arrow, so what you see as a line of arrows is a result of this padding. What

appears to be garbage after the first EOF character is actually part of the file, but most programs (like compilers and DOS commands) ignore everything after the first EOF. You can delete these characters if you don't like seeing them, but they shouldn't cause any problems for BRIEF or other programs. BRIEF does put one EOF character at the end of files for compatibility with commands that require it, unless the -z flag is specified.

Why do several common functions not work, and why are my tabs only one character wide?

BRIEF can't find its macros. This situation should only occur if you didn't run SETUP or if SETUP terminated improperly. Either copy the .cm files into \brief\macros on the default drive, or use BPATH to specify where the macros actually are. You should also make sure that the macro directory exists, and, if you've recently received an update, that the macros have been compiled with the most recent macro compiler.

Will BRIEF run with other programs in memory?

It should. With SideKick, and any other program that doesn't support multiple pages of memory, use the -p flag. ProKey version 3.0 also works fine; with ProKey 4.0, you must specify ProKey's /z- flag.

Why does search not work as expected?

You may have included some characters in your search string that are special regular expression symbols. See the section titled "Using search and translate" in "Command overview" (Chapter 3) for further information. Another possibility is that you are using the @ sign incorrectly: [0-9]@ will match zero or more consecutive integers, which is quite different from matching one or more. To match one or more, use [0-9]+.

Why do I get the message "<macro name> undefined" when I try to execute a command using *F10*?

There are two possible problems: the command could really be undefined or you may not be entering the name correctly. In the first case, make sure that the file where the macro is defined is loaded before you try to invoke the command. In the second case, you must enter command-line commands in lower case, without leading blanks.

How do I translate an integer ASCII value to a character and vice versa in a macro?

To go from a character to an integer, use **(atoi)**. To go back, use the **(sprintf)** function. Read the documentation in the *Macro Language Guide* and note that the format string is not restricted to integer and string types, but may be any C-type control string. This means you can use `%c` to format a character. The call

```
sprintf (output_string, "%c", 32);
```

will put a space (ASCII 32) into **output_string**.

Why won't BRIEF do what I want after I've rewritten the macros?

You probably forgot to recompile the macro.

How can I put a backspace (ASCII 8), carriage return (ASCII 13), line feed (ASCII 10), null (ASCII 0), or graphics character (ASCII 128+) in my file?

Linefeed can be inserted with *Ctrl+j*, as can carriage return with *Ctrl+m*, but single carriage returns are always followed by linefeeds, whether you type one in or not. Graphics characters can be entered by using the *Alt* key in conjunction with the numeric keypad; just type the ASCII code. Backspace can be inserted with the **Quote** command (see "Command overview" [Chapter 3] or "Command reference" [Chapter 8]). Nulls cannot be entered at all.

Is there a way to get better error messages from the macro compiler?

Yes. The compiler's `-p` flag will show you the line where the error occurred, as well as the approximate location of the problem.

Can I simulate arrays in the macro language?

You can simulate an array by using a buffer. For an example, see the **buf_list** macro in `\brief\macros\buffers.cb`.

A P P E N D I X

A

Help for new users

If you are a new BRIEF user, this section has been designed to help you become familiar with the screen, keyboard, and mouse. It also contains overviews of some basic BRIEF functions.

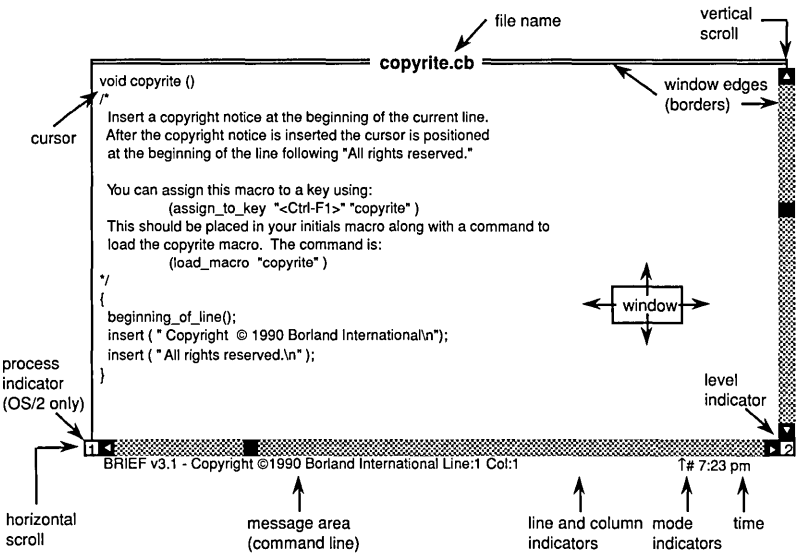
This chapter includes

- The BRIEF screen
- The keyboard
- The mouse
- Command overview
- Buffer overview
- Window overview
- Scrap overview
- Current position

The BRIEF screen

The main features of the BRIEF screen are the window and the message area.

The BRIEF screen



Window

The window is the area bounded by single and double lines, or borders, that displays text entered into the file. The top border of the window contains the file name you specified on the command line. If the file is a new file, this space is empty and the window is blank.

Message area

The message area, below the window, is used to display messages and to prompt for information.

The message area is sometimes referred to as the command line.

If you enter an existing file name, the copyright notice for BRIEF appears. If you enter a new file name, the following message is displayed:

```
New file (unable to open file name.ext)
```

Line and column indicators

The next two items identify the line (row) and column where the cursor is located. When you invoke BRIEF, the cursor is located at line 1, column 1.

Level indicator

The level indicator is a single digit that appears in the lower right corner of the window border if you have more than one copy of BRIEF resident in memory at once. The number (which ranges from 2 to 9) indicates the number of copies of BRIEF currently in memory. More than one copy can be in memory at once through the use of the **Suspend BRIEF** command or DOS multi-tasking systems (see "Using special features" (Chapter 4) and "Command reference" (Chapter 8) for more information). If only one copy of BRIEF is running, no number appears.

OS/2 process indicator

The process indicator is a single digit that appears in the lower left corner of the window border if you are running other programs inside BRIEF in the background. The number (which ranges from 1 to 9) indicates the number of background processes currently running under BRIEF's control. See "Programming support" (Chapter 6) for more information. If only one version of BRIEF is running, no number appears.

Time

The last item in the message area is the time, which is displayed in hours and minutes, with a blinking colon as a separator. Twenty-four hour time is used if appropriate for your country.

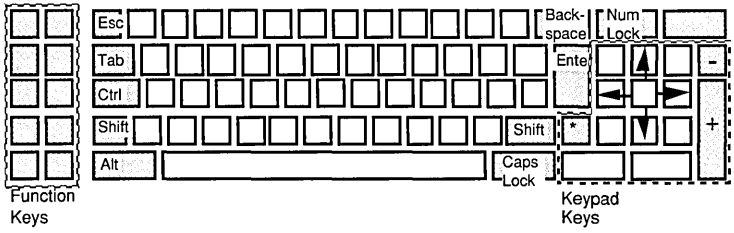
The keyboard

Look at the keyboard diagrams on the next page and locate some of the special keys.

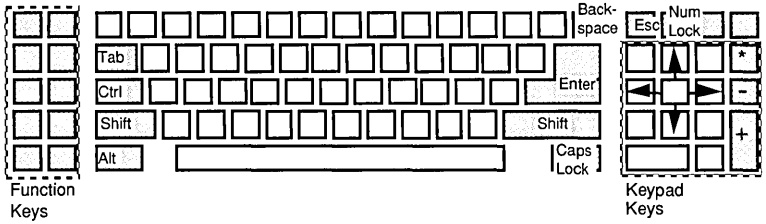
The three keyboard layouts described here are

DOS	`PC keyboard layout`
DOS–OS/2	`AT keyboard layout`
DOS–OS/2	`Enhanced keyboard layout`

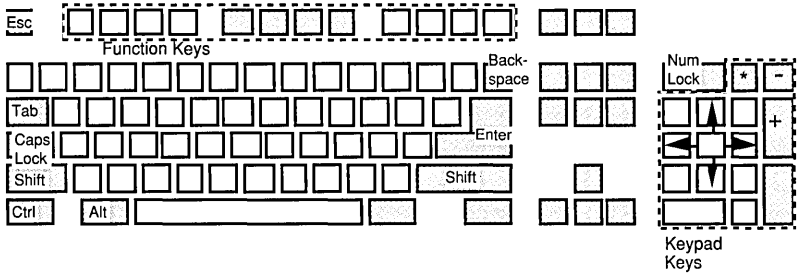
PC Keyboard Layout



AT Keyboard Layout



Enhanced Keyboard Layout



Shift and caps lock

On most IBM PCs and compatibles, the *Shift* keys are located at either end of the row of keys above the space bar and marked with a thick up arrow. In some cases, they are also marked with the word `Shift`.

Press a *Shift* key. While it is pressed down, note that the message area displays an up arrow in the mode indicator area to the left of the time. The arrow indicates that the keyboard is in a shifted state, so you can enter capital letters. When you release the *Shift* key, the arrow disappears.

Another key that affects the shifted state is the *Caps Lock* key. It is located below the *Shift* key on the right side of the keyboard. On the Enhanced IBM keyboard, it's located to the left of the *A* key.

Press the *Caps Lock* key and note that the up arrow stays in the mode indicator area even after you release the key. *Caps Lock* puts the keyboard in a shifted state until it is pressed again. When *Caps Lock* is on, the function of the *Shift* key is reversed. That is, when *Caps Lock* is on and the *Shift* key is pressed, the up arrow goes away and a lower case letter or character will be entered.

Num lock

Next, locate the *Num Lock* key. On most keyboards it's on the top row of the keypad, above the 7 or 8.

Press *Num Lock*. The mode indicator area displays a pound symbol (#). This indicates that *Num Lock* is on and the numeric keypad can be used to enter numbers. The arrow keys can only be used to move the cursor if the *Shift* key is pressed.

Locating other keys

Now you should locate other keys frequently used in BRIEF. The following list describes the location of keys on the IBM PC keyboard; if your keyboard differs, note where the equivalent keys are placed.

- Function keys

  On PC/AT keyboards, the function keys are located in two vertical rows at the left of the keyboard. The keys are labeled *F1* through *F10*. On Enhanced keyboards, they're located along the top and labeled *F1* through *F12*.

- *Esc*

  On the PC/Enhanced keyboards, the Escape key (marked *Esc*) is located near the top left corner of the keyboard. On the AT, it's located on the numeric keypad.

- *Ctrl*

  On the PC/AT, the Control key (marked *Ctrl*) is located to the left of the *A* key. On the Enhanced keyboard, it's below the left *Shift* key.

- *Tab*

  The *Tab* key, located in the row of keys to the left of the *Q* key, is usually marked with two horizontal arrows facing in opposite directions.

- *Backspace*

  The *Backspace* key, located in the upper right corner of the main key section, is identified with an arrow pointing to the left.

- *Enter*

  The *Enter* key, the large key on the right side of the main key area, is marked with a bent arrow pointing to the left.

- *Alt*

  The *Alt* key is located on the bottom row to the left of the space bar.

- Keypad keys

  The numeric keypad keys are to the right of the keyboard. Note that if you have separate cursor keys, you must still use the keypad cursor keys for shifted commands (for example, *Shift+Left*, *Shift+Home*).

- Gray keypad keys

  The gray keypad keys (–, +, and *) surround the keypad and are darker in color.

- Dedicated keys

  The dedicated keys are only present on the enhanced keyboard. These keys duplicate most of the keys on the numeric keypad.

Many of the commands assigned to these keys are discussed in the tutorial (Chapter 2). *Shift+Right* or *Ctrl+T* indicates that two keys should be pressed simultaneously.

The mouse

If your system includes a mouse, you can use it in place of commands or some keystrokes.

Mouse cursor

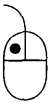
The mouse has its own cursor displayed on the screen, regardless of which window is active. Moving the mouse moves its cursor and quickly changes the position of the text cursor.

A mouse can have either two or three buttons. A two-button mouse (left and right buttons) can imitate a three-button mouse. The buttons are typically labeled left to right as 1, 3 (if present), 2 (this set-up is predominantly used by right-handers).

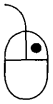
Mouse button 1 is user-definable in SETUP. See "Complete installation" in Chapter 1 for more information.

Mouse icons

Any commands in BRIEF that use the mouse display one of the mouse icons:



The left button (1) is the one used most often.



The right button (2) is often used for the opposite action to *Mouse Button 1*. For example, if *Mouse Button 1* scrolls the buffer up, *Mouse Button 2* scrolls the buffer down.

If you have a two-button mouse, using both buttons at the same time is equivalent to using *Mouse Button 3*. The documentation uses this icon when *Mouse Button 3* is to be used.

Mouse button actions

The actions associated with the mouse are

- **click**

  To click a mouse button, position the cursor where desired and press, then release, the button.

- **drag**

  To drag the cursor, press the mouse button, position the cursor where desired by moving the mouse, and release.

- **double click**

  To double-click a button, press and release it quickly twice.

- **with other keys**

  Some instructions include the *Shift*, *Alt*, or *Ctrl* keys. Hold the appropriate key, position the cursor, press the appropriate button, drag the mouse, and release both the button and the key.

To change the position of the text cursor, move the mouse cursor to the position desired and click *Mouse Button 1*. If you are using multiple windows, you can activate a new one by moving the mouse cursor to the window and clicking *Mouse Button 1*. Note that this action does not reposition the text cursor. You must reposition the mouse cursor and click again to do that.

See "Command overview" (Chapter 3) for mouse instructions in commands.

Command overview

Commands are actions that can be done from the keyboard. They are used to write files, delete characters, search for patterns, etc.

The more frequently used commands have assigned keystrokes that are usually mnemonic, such as *Alt+G* for **Go to Line**.

Commands without assigned keystrokes are executed by pressing *F10* and entering the full command name (such as **margin** or **color**). The command line is case-sensitive and all commands must be entered in lower case to be recognized. Key sequences can be reconfigured to execute any command and the set of keystrokes mapped to commands is called a *keymap*. You can also use the mouse in place of *F10* by clicking *Mouse Button 1* in the command area. For information on reconfiguring the keyboard, see "Using special features" (Chapter 4).

Escaping from a command

Execution of any command that requires a response to a prompt, such as **Go to Line**, can be stopped by pressing *Esc*. This keystroke does not work for commands that provide immediate execution, such as **Insert Mode Toggle**.

Command response history

Up to ten responses are saved for each prompt displayed, as long as they are more than one character long. Use ↑ and ↓ to cycle through the list of remembered responses. *Esc*, which cancels the command, is not stored.

Last response

You can also recall the last response entered at any prompt by typing *Alt+L*. This is useful when you find you entered the correct response to the wrong prompt. Press *Esc* to cancel the first command, issue the new command, and press *Alt+L* to recall the last response.

File name completion and wildcards

Certain commands (**Edit File**, **Read File**, **Load Macro**, and **Delete Macro**) can take advantage of the file name completion feature. Enter the first one or two letters of the file name and press *Tab*. If only one file in the directory matches, it is displayed. If more than one file is found, a menu is displayed. Select the desired file. If you want to choose more than one file, use *Spacebar* to select the files.

When the correct file(s) have been selected, press *Enter*.

Standard wildcard characters (such as those used with `DIR`) are allowed in partial file name specification. All matching files are presented.

If no matches are found, the system beeps.

Buffer overview

A buffer is an area of memory that holds text, which can be added to, deleted from, or changed without affecting the original file (until the buffer is saved). Buffers are restricted in width to the current line length (default: 144), but they can be as long as desired, if they fit on the disk.

See the section on buffers in "Command overview" (Chapter 3) for more information.

Window overview

A window views a portion of text from a buffer. If a buffer contains more text than can be seen in one window, the text must be scrolled so the text can be viewed.

BRIEF uses tiled, not overlapping, multiple windows for editing. These windows appear side-by-side or over-and-under on the screen.

Windows can view different buffers or they can look at different areas of the same buffer. Windows can be created, deleted, and resized through keystrokes or with the mouse.

See the windows and mouse sections in "Command overview" (Chapter 3) for more information.

Scrap overview

A scrap buffer is a special buffer that is only used for moving or copying blocks of text. It differs from a regular buffer in that:

- Blocks of text can only be copied into or pasted from it
- It is automatically created whenever you cut or copy text
- Text inside the scrap cannot be edited (except through macros)

The three commands that work on the Scrap Buffer are **Copy**, **Cut**, and **Paste**. You can use keystrokes or the mouse.

See the scrap and mouse sections in "Command overview" (Chapter 3) for more information.

Current position

The current position is the position in the current buffer where the cursor is located. The current buffer is always viewed by the active window. Most commands use the current position as a reference point.

I N D E X

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

BRIEF®
FOR DOS AND OS/2®

BORLAND

Corporate Headquarters: 1800 Green Hills Road, P.O. Box 660001, Scotts Valley, CA 95067-0001, (408) 438-8400. Offices in: Australia, Belgium, Canada, Denmark, France, Germany, Hong Kong, Italy, Japan, Korea, Malaysia, Netherlands, New Zealand, Singapore, Spain, Sweden, Taiwan and United Kingdom ▪ Part #15MN-BRF02-31 ▪ BOR 4165