TE6

D 7205

F editor Preliminary version

INMOS Limited

72 TDS 277 00

March 1991

INMOS document number: 72 TDS 277 00

Contents

1 F editor overview

1.1 The F editor

1.1.1 Introduction

The F editor is a general purpose text editor which may be used for editing documents or program source code in a variety of languages. It is a direct descendant of the editor for occam source which was a principal component of the INMOS Transputer Development System. F uses normal ASCII text files as supported in the host filing system. Several files may be edited at the same time and there are facilities for transferring blocks of text between these files. Host operating system commands may be called without leaving the editor.

The editor is based on the concept of 'folding'. The folding operations allow the text to be given a hierarchical structure ('fold structure') which reflects the structure of the program or document under development.

This document starts by explaining folding. As with many systems, the best way to start learning about F is to start using it. A tutorial file is provided and is discussed at the end of this section.

1.1.2 Typographical note

Throughout this manual the convention of referring to function keys by name will be followed; for example: $\boxed{\text{CURSOR UP}}$ or $\boxed{\text{LOCATE LINE}}$. In fact these logical names may correspond to a combination of physical keypresses at the terminal. The actual keys associated with these function key names are given in the keyboard layout diagrams in appendix A. The mapping is determined by the keyboard section of an `ITERM` file, see appendix B.

1.1.3 Folding

Just as a sheet of paper may be folded so that portions of the sheet are hidden from view, the folding editor provides the ability to hide blocks of lines in a document. A fold contains a block of lines which may be displayed in two ways: open, in which case the lines of the fold are displayed between two marker lines (called creases), or closed, in which case the lines are replaced by a single marker line called a fold line.

To create a fold the user inserts creases around the text to be folded; the fold is closed automatically when the second crease is made. Any text may be placed on the fold line to indicate what the fold contains; this text is called the 'fold header'.

A fold may be removed so that its contents are once again placed in sequence with the surrounding lines.

Folds may contain text lines and also fold lines. Folds can be nested to a maximum depth of 50.

An example of how folds are displayed by the editor follows. The fold line is marked with three dots (`...`). A top crease is marked with the symbol `{{{`. A bottom crease is marked with `}}}`. There are two folds in the example program below: one marked **`Declarations`**, and one marked **`initialise`**. In the second example the fold **`initialise`** has been opened.

Example: occam program with closed folds

```
... Declarations
SEQ
  ... initialise
  WHILE going
    process(ch, going)
```

Example: occam program with open fold

```
... Declarations
SEQ
  {{{  initialise
  going := TRUE
  input ? ch
  }}}
  WHILE going
    process(ch, going)
```

A fold has an indentation associated with it; the fold and crease line markers begin at this indentation level. No text may be inserted within the fold to the left of this indentation. In occam the indentation of a line is significant; the folding features of the editor make it relatively easy to change the indentation of part of an occam program.

The editor facilitates the systematic use of indentation in other block structured programming languages, such as Ada, C or Pascal. Folding, in conjunction with the ability to nest folds, provides a way of organising a large document or program as a hierarchy. The editor has functions to 'enter' a fold, which opens the fold and moves down into it, and also to 'exit' the fold, which closes the fold and returns to the level from which the fold was entered. For example, entering the fold marked **`Declarations`** in the example above would make the following lines the only visible lines on the screen.

Example: entering a fold

```
{{{  Declarations
INT ch:
BOOL going:
PROC process()
  ... body of process
:
}}}
```

Here the line marked **`body of process`** is a fold nested inside the fold **`Declarations`**.

Any document can be folded in such a way that most of the folds are shorter than the length of the screen. Fold operations then become the principal method of traversing a document, with screen scrolling operations used only for small local movement.

Because a closed fold is represented by a single line on the screen, some editor line operations may act on fold lines as well as text lines. When such an operation is applied to a fold line it also applies to the fold contents. For example, deleting a fold line deletes all its contents as well. This means that operations to transform the fold structure, (such as moving, copying, and deleting folds) appear identical to the line operations which are familiar to any user of a screen-oriented editor.

When a file is written to disk each top and bottom crease line is converted into a comment line, using a representation which depends on the language in which the text is written. The curly bracket sequences `{{{` and `}}}` are included within the text of these comment lines so that their nature is immediately obvious if the file is printed.

Files with fold creases represented in this way can be created by Tools or Utility functions in the INMOS Transputer Development System (TDS), which is the precursor of the F editor, but uses a different representation of text files. TDS files can also be flattened into host text files on a PC by using the occam toolset program **`iflat`**.

If desired, space may be saved in text files on disk by using TAB characters to represent groups of 8 consecutive spaces at the start of text lines. This saving is particularly valuable in highly indented occam source programs. All INMOS occam compilers interpret leading TAB characters in text lines in source files as sequences of 8 spaces.

1.1.4 The philosophy of F

The F editor retains the concept of folding from its predecessor, the TDS editor. Users of the TDS will notice two principal changes: firstly there is no longer a concept of nested filed folds, although similar effects may be achieved using the power of F to edit several files together and secondly the files are written to disk as ordinary host text files which are accessible to all other tools and do not require any special naming conventions.

Folding is a mechanism which allows as much detail of a file to be concealed or revealed as desired. Users of folding editors find that they are able to focus attention on critical areas of development in a way not possible in a conventional 'flat' editor. An interesting side-effect of this is that folding users gradually rely less and less on paper listings of programs. This section of the introduction offers an insight into the motives which governed the development of F and hence some suggestions for using F in the most efficient manner.

The editor is designed at its core level to be simple to use with a minimum number of magic keys. As users become more familiar with the editor, additional features can be used until they have a working set which suits their personal style.

All terminal based INMOS software tools use an ITERM file containing keyboard mapping information to map functions onto keystrokes so that the tools may be written in a terminal and system independent manner. F retains and slightly extends this concept. F is expected to be used in three different work environments; firstly, a user who has personalised the interface to the editor, secondly, a user logging into a system-wide facility which uses a standardised interface to the editor, and thirdly, a user who uses a system-wide facility but wishes to have some degree of personalisation.

To accommodate these three modes F assumes default names of information files which can be superseded at a number of levels. The primary information file is the ITERM file which controls the key mappings. The ITERM file can be used to reference an F startup file which controls how F interprets folds in the file being read. The startup file could be system-wide but a user may wish to extend the fold marker set chosen and so can override the reference to the startup file by defining an environment variable to point at a file of his own choosing. The startup file references the name of a command history file.

The command history file is opened at the start of an editing session. By default it contains a list of executable commands - commands which operate on files rather than editing commands which operate inside a file. If the command history file is kept as a global reference from a standard startup file it would not be edited by an individual user. The command file can, however, be a reference to a local file in which a user can retain a list of commands issued on a regular basis in that environment. For example, a list of all the source files in a particular directory along with a backup command and a call to a make file could be retained in a command history file.

The editor is intended to be consistent so that all commands behave in the same manner in all environments. An example of this consistency is shown in the handling of the command file and the help file which being ordinary text files can both be navigated using the editing facilities of F. Because the command history file is no different from any other file, all commands which can be obeyed from the command history file can also be obeyed from within a normal text file being edited. As a consequence opening additional files and interacting with the host system can be done with a minimum of keystrokes and special states.

Modern editors often have windowing capabilities built in and allow different windows to

show different views of the same file. F assumes that a windowing environment will always provide more sophisticated facilities than could be incorporated into the editor and thus provides only the mechanism to switch between files. On a folding editor, multiple positions in a single file are undesirable because the view of the file depends on the state of lines within the file (for example one view could close a fold containing the current position of another view).

INMOS welcomes suggestions for future versions of this editor. A particular area that could be expanded in a variety of ways is the mechanisms for searching and replacing, including provision for more versatile pattern matching, multiple file searches, global replacement, etc.

1.1.5 Calling the editor from host operating system

The editor is called from the host operating system as an executable program or a command file. The latter mechanism is used when it is necessary to set and restore components of the operating environment required by co-existing software. The command line should include as parameters a list of one or more names of files to be edited. Ensure that this program or command file will be called whenever `f` is typed.

F assumes the existence of a host environment variable **`FTERM`** containing the name of an appropriate ITERM file, see appendix B. If the ITERM file cannot be found, or is not in the correct form, the editor will fail to start up. A minimal ITERM file **`ASCII.ITM`** and corresponding keyboard help file **`ASCII.HLP`** are provided in case the editor is being installed in an unfamiliar operating environment. This requires a terminal whose keyboard can generate ASCII control codes and ANSI cursor move codes only. When using **`ASCII.ITM`**, function keys are emulated by escape sequences using ESC, Ctrl-A and Ctrl-W as prefix codes. It should then be possible to create a better ITERM file using the F editor, especially with the help of the DISPLAY KEY key.

An F ITERM file in turn identifies an F startup file. If the same ITERM file is to be used from multiple directories it may be desirable to ensure that full directory paths are used for the name of the startup file and further files which are mentioned therein. If an environment variable **`FSTART`** is defined, the file named by this variable is used as startup file in preference to any defined in the ITERM file.

The startup file identifies the command history file. It also defines the conventions used in different languages for converting crease lines into comments. As a file is read by F the language convention will be determined from the first line containing 3 curly braces. If no such line is found the convention will default to the first in the list.

A set of up to 10 keystroke macros may be defined in macro folds in the startup file and are associated at startup time with the ten tool keys. These enable users to create compound operations of their own design for commonly used operations, such as commenting out chunks of program, etc. A few such macros are shipped in the default startup file as examples.

1.1.6 Keyboard layout

To display a map of the keyboard layout, press the HELP function key. A message identifying the help key is copied from the ITERM file or the startup file to the top of the screen at the start of an editing session. A keyboard map, which is also a folded text file, will appear; you can return to the normal editor display by pressing the FINISH key (usually mapped onto CTRL-X). Keyboard layouts for common hosts are also shown in appendix A.

The information in the keyboard map display is read from a file whose name may appear in the startup file, or by default is derived from the name of the ITERM file by replacing the suffix (usually `.itm`) by `.hlp`. If the user changes the keyboard mapping then an appropriately modified keyboard map file should be created. If no keyboard map appears, check that a suitable file will be located by these rules.

A special function key DISPLAY KEY is provided to faciliate the setting up of an ITERM file for an unfamiliar keyboard type or host environment.

1.1.7 Repainting the screen

The function key REFRESH repaints the entire screen. This may be useful to check that the editor is driving the screen correctly, or if the terminal is accidentally switched off and on again. If the editor is running in a windowed environment this key is used to resize the display if the size of the window is changed.

1.1.8 Ending the session

If a single file is being edited, pressing FINISH saves the file and returns to the operating system. If folds have been entered, it is not necessary to exit them before FINISH can be used.

If more than one file is being edited (see below) FINISH saves the current file and the next in the ring becomes current. Repeated use of FINISH will save all files being edited. There are also named commands which can terminate an editing session in a variety of ways, see 1.3.14.

1.1.9 Tutorial file

If you have not yet successfully installed the software you may now need to refer to appendix C or to the delivery manual of the accompanying toolset.

The tutorial file included with the system provides an introduction for those starting to use

the system. The file is called `FTUTOR.OCC`. This file contains a detailed practical example on using F and anyone new to this system is strongly advised to work through it.

To use the tutorial move to the appropriate directory, check that the environment variable `FTERM` is set up, make a backup copy of `ftutor.occ` and then type:

```
    f ftutor.occ
```

to start the system.

It may be helpful to have a printed form of the appropriate keyboard layout available. Keyboard layouts appear in appendix A.

The contents of the file will then give you detailed instructions on how to proceed.

The tutorial file does not assume any knowledge about F so it can be worked through before reading the rest of the chapter.

The rest of this chapter describes the editor interface in some detail, and then describes the facilities for calling other tools from within the editing environment.

1.2 The editor interface

This section defines some terms which are used to describe the behaviour of the editor keys. Figure 1.1 shows a graphical representation of these terms.

1.2.1 Editor's view of a document

At any time during the session, the editor has a view of the document, consisting of a sequence of text lines, closed folds and open folds. This is called the current view.

The current view of the document at any time is principally determined by the fold operations which have been carried out. At the start of the session the current view contains a sequence of lines at the top level of the first file mentioned on the command line when F was called from the host operating system. Other files mentioned on the command line will be accessible by means of NEXT FILE or PREVIOUS FILE. These files, and any others subsequently added, are conceptually held in a ring of files.

Whenever ENTER FOLD is pressed on a fold line, the current view is stacked up, and the contents of the fold become the current view. After editing the contents of the fold it is possible to return to the previous view by using EXIT FOLD.

1.2.2 The screen display

The screen is divided into two parts. The top line of the screen is used to display messages. The rest of the screen displays a 'window' into the current view (that is, it displays as many lines of the current view as will fit on to the screen).

The editor provides functions to move the screen window up and down the current view, thus providing a scrolling facility. These functions do not change the editor's view of the document, merely what is visible in the screen window.

The cursor is used to point to a position in the screen window; functions are provided to move the cursor around the screen. The cursor cannot be moved below the end of the current view.

The current column is the column which the cursor is on. The current line is the line which the cursor is on. The current enclosing fold is the fold which contains the current line, or, if the current line is a crease line, the fold formed by that crease and its partner. The current file is the file from which the text in the current view was read, and to which, by default, it will be saved at the conclusion of the edit.

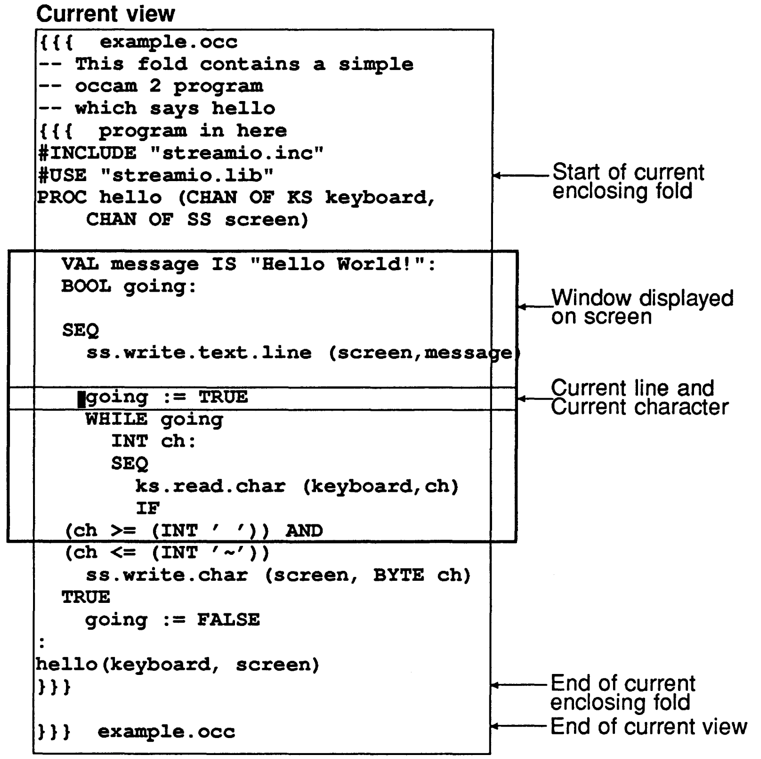
1.2.3 Line types

Five general types of line may be displayed; they are text lines, top creases, bottom creases, fold lines and fold creation marks.

Fold lines and crease lines start with a marker symbol. The different types of marker symbols are:

Fold line	...
Top crease	{{{
Bottom crease	}}}
Fold creation mark	!!!

All marker symbols consist of the textual symbol above, plus two following spaces to give the symbol a width of five characters. The marker symbols of the outermost creases of a file also include the name of the file. The marker is protected from change by the editor. Text may be written onto a crease line beyond the marker symbol. The text on a top crease is also visible when the fold is closed. The text on a bottom crease is only visible when the fold is open (and is lost if the file is reimported into the TDS).


Current view
Start of current enclosing fold
Window displayed on screen
Current line and Current character
End of current enclosing fold
End of current view

```
{{{  example.occ
-- This fold contains a simple
-- occam 2 program
-- which says hello
{{{  program in here
#INCLUDE "streamio.inc"
#USE "streamio.lib"
PROC hello (CHAN OF KS keyboard,
    CHAN OF SS screen)

  VAL message IS "Hello World!":
  BOOL going:

  SEQ
    ss.write.text.line (screen,message)

   ▌going := TRUE
    WHILE going
      INT ch:
      SEQ
        ks.read.char (keyboard,ch)
        IF
  (ch >= (INT ' ')) AND
  (ch <= (INT '~'))
    ss.write.char (screen, BYTE ch)
  TRUE
    going := FALSE
:
hello(keyboard, screen)
}}}

}}}  example.occ
```

Figure 1.1 Editor's view of a document

1.3 Editor functions

This section introduces and describes the functions provided by the editor. A detailed listing of the keys used by the editor is available in chapter 2. The mapping of key names to keys on the keyboard is given in appendix A.

1.3.1 Overview of editor functions

The editor accepts and acts on sequences of keystrokes from the user. If any of the sequences are not recognised the terminal bell rings and/or a message appears in the message line. The table below provides an overview of the available editor functions, which are described in detail in the following sections. On-line versions of this table enhanced with the names usually found on the keytops are included in most versions of the keyboard help files shipped with the software.

Moving the cursor	CURSOR UP	CURSOR DOWN	WORD LEFT	TOP OF FOLD
	CURSOR LEFT	CURSOR RIGHT	WORD RIGHT	BOTTOM OF FOLD
	START OF LINE	END OF LINE		
Case changing	TO UPPER	TO LOWER		
Scrolling the screen	LINE UP	LINE DOWN		
	PAGE UP	PAGE DOWN		
Fold browsing	ENTER FOLD	EXIT FOLD	BROWSE	
	OPEN FOLD	CLOSE FOLD		
Inserting and	Character keys	RETURN	DELETE WORD LEFT	
deleting characters	DELETE	DELETE RIGHT	DELETE WORD RIGHT	
			DELETE TO END OF LINE	
Fold creation and removal	CREATE FOLD	REMOVE FOLD		
Deleting lines	DELETE LINE	RESTORE LINE		
Moving and	MOVE LINE	COPY LINE	COPY PICK	PICK LINE
copying lines			PUT	
Search and replace	NEW SEARCH	NEW REPLACE	LOCATE LINE	
	SEARCH	REPLACE		
Defining and using	DEFINE MACRO	CALL MACRO	SAVE MACRO	GET MACRO
a keystroke macro	TOOL0 ..	TOOL9		
File changing	LIST FILES	NEXT FILE	PREVIOUS FILE	SAVE
Escapes	FINISH	COMMAND	OBEY	COMMENT STYLE
	DISPLAY KEY	TOGGLE TABS	REFRESH	

1.3.2 Editor states

At certain times when using the editor, only a limited subset of the editor functions may be available. For example, a fold is created by two presses of a key called CREATE FOLD; one to mark the top of the fold and one to mark the bottom of the fold. Between these two presses normal editing operations are not allowed; the only keys which the editor will accept are those needed to move the cursor up and down and the help key. All other keys cause a warning message to be displayed. When the editor is only accepting a restricted subset of keys, this is known as an editor 'state'. It is indicated by a message on the top line of the screen which persists until the operation requiring the special state has been completed. The default editor state is `Editing`. This state has two additional flags defining whether or not the file has been changed since being read from disk, and whether or not leading spaces are to be compressed into TABs on output. In the state message an asterisk `*` after `Editing` shows that the file has been changed and a `!` that space compression will be done. The `!` flag is initialised according as TABs are found in the file when it is read and is toggled by the TOGGLE TABS key.

In the rest of this chapter, where a function results in a change of editor state, this is indicated in the appropriate section.

1.3.3 Moving the cursor

The normal cursor positioning functions are used to move the cursor around the screen window. The cursor may be moved into any part of the screen, except the message line. In addition there are functions to move the cursor to the start or the end of the current line, and one word to the right or left on the line.

The cursor keys when used at the top and bottom of the screen cause the screen to scroll. Separate screen scrolling functions can be used to scroll the screen up and down the current view; these are described in the next section.

CURSOR UP moves the cursor up one line.

CURSOR DOWN moves the cursor down one line.

CURSOR LEFT moves the cursor left one column.

CURSOR RIGHT moves the cursor right one column.

END OF LINE places the cursor after the last significant character on the current line (which is normally the last non-blank character).

START OF LINE places the cursor on the first significant character of the current line (normally the first non-blank character).

Two keys, WORD LEFT and WORD RIGHT, are provided to move the cursor one word at a time. A word consists of a sequence of alphanumeric characters or adjacent non-alphanumeric characters. More precise definitions of the word move operations are given in chapter 2 under the definitions of the relevant keys.

WORD LEFT moves the cursor one word to the left of the current cursor position.

WORD RIGHT moves the cursor one word to the right of the current cursor position.

TOP OF FOLD moves the cursor to the top crease line of the current enclosing fold. If the top crease line is not within the screen window the screen will be scrolled.

BOTTOM OF FOLD moves the cursor to the bottom crease line of the current enclosing fold. If the bottom crease line is not within the screen window the screen will be scrolled.

1.3.4 Changing case

There are two keys for changing the case of alphabetic characters. These both change the case if a character is currently a letter of the opposite case, and then move the cursor one place to the right. Other characters are not changed.

TO LOWER changes a character from upper to lower case.

TO UPPER changes a character from lower to upper case.

1.3.5 Scrolling and panning the screen

Scrolling can occur as a side effect of some other operations or can be explicitly requested. Panning always occurs as a side effect of cursor moves off the sides of the screen.

The functions introduced below scroll the screen up and down the current view by a line or a page at a time. A page is the number of lines in the screen window.

LINE UP moves the screen one line up the current view, if there are lines in the current view above the screen, otherwise behaves as cursor up.

LINE DOWN moves the screen one line down the current view, if there are lines in the current view below the screen, otherwise behaves as cursor down.

PAGE UP moves the screen one page up the current view, or to the top of the current view, whichever is the nearest.

PAGE DOWN moves the screen one page down the current view, or to the bottom of the current view, whichever is the nearest.

There are no keys which explicitly control panning, but if actions are taken which drive the cursor off the sides of the screen the whole screen will be repainted with the cursor position on the screen and an indication of how many columns are missing at the left in the message line at the top of the screen. There is an absolute maximum line length of 511 characters which can never be exceeded and it is recommended that the need to pan is minimised by keeping lines within the capacity of the screen.

If a line extends beyond the rightmost writable position in a line, which may be different from the rightmost visible position, a `Long line` message will be displayed.

1.3.6 Fold browsing operations

Opening and closing folds

This section describes the keys which are used, along with the cursor positioning keys, to move around a document. There are two pairs of fold browsing operations, one pair being ENTER FOLD and EXIT FOLD, and the other pair being OPEN FOLD and CLOSE FOLD.

The folding features of the editor give a document a hierarchical structure. The keys ENTER FOLD and EXIT FOLD are used to move around the hierarchy. When ENTER FOLD is pressed on a fold the screen is cleared and the contents of the fold become the current view. The previous view is saved, and can be returned to by using EXIT FOLD

ENTER FOLD is appropriate when the fold contains a reasonably self-contained piece of text. However, it may often be more desirable to view a piece of text in its surroundings; for example the body of a `WHILE` loop may be folded up, and it may be best viewed with the `WHILE` condition displayed above it. OPEN FOLD and CLOSE FOLD are provided for this purpose.

OPEN FOLD inserts the contents of a fold between the surrounding lines, bracketted with top and bottom creases. CLOSE FOLD may be used to close an opened fold, and replace the displayed contents with a single fold line.

ENTER FOLD is useful where a quick return up to a particular position is required; doing an ENTER FOLD at that position will allow, at some future time, an EXIT FOLD to cause a return back up to that position.

If there are open folds within a fold which is exited, then these folds will remain open if the exited fold is reentered during the current session.

Browsing state

Sometimes when viewing an existing document it is useful to set the editor into a state so that you can not accidentally change the document. The key BROWSE can be used to get into and out of this state. While in this state a message is displayed on the message line, and all editor functions which could change the document are disallowed.

1.3.7 Inserting and deleting characters

In general, characters may be inserted or deleted at the cursor position, but there are some exceptions, as follows:

1. Text may not be be inserted when the cursor is on a fold or crease marker, when the cursor is on a fold creation mark line, or when the cursor is to the left of the leftmost column of an open fold or there are already 511 characters in the line (including indentation spaces).
2. The indentation of a closed fold may be changed by inserting or deleting spaces to the left of the fold marker symbol. No other text may be inserted there.

Insertion

A character or space can be inserted in the current column position and the cursor, the character at the cursor and all subsequent characters on the line are moved right by one place. If a character is inserted beyond the end of visible text on a line, spaces are implicitly inserted before it.

$\boxed{\text{RETURN}}$ is used to split lines and insert blank lines. It has no effect if used within the fold marker on a fold line.

Deletion

$\boxed{\text{DELETE}}$ is used to delete the character to the left of the cursor. This causes the character at the cursor and the rest of the line to the right to be moved one place to the left. If $\boxed{\text{DELETE}}$ is used at the extreme left of a line it concatenates the line with the preceding line, if that line is not too long. It has no effect if used at the extreme left of a fold or crease line or within a fold marker.

$\boxed{\text{DELETE RIGHT}}$ deletes the character at the cursor. All the characters to the right of the cursor are moved left by one place. The cursor remains in the same position.

Character deletion has no effect when the cursor is on part of a marker symbol, or is to the left of the leftmost column of an open fold.

Spaces may be deleted to the left of a closed fold to change the indentation of the fold.

$\boxed{\text{DELETE TO END OF LINE}}$ deletes all text from the character at the cursor, to the last significant character on the line, inclusive. The cursor remains in the same position.

Deletion can take place a word at a time. A word can be considered to be a sequence of alphanumeric characters or adjacent non-alphanumeric characters, as for cursor movement.

$\boxed{\text{DELETE WORD LEFT}}$ deletes the word to the left of the cursor.

$\boxed{\text{DELETE WORD RIGHT}}$ deletes the word to the right of the cursor.

1.3.8 Fold creation and removal

Fold creation is achieved by marking the top and bottom of the sequence of lines required to form the contents of a fold. Two presses of $\boxed{\text{CREATE FOLD}}$ are needed to do this. Firstly the cursor should be placed on the top line and $\boxed{\text{CREATE FOLD}}$ pressed. A fold creation mark will appear. The cursor should then be moved to the line below the bottom line to be folded, and $\boxed{\text{CREATE FOLD}}$ pressed again. Alternatively the bottom crease may be marked first and the cursor moved up to the position for the top crease. The effect is identical whichever order the creases are marked.

After $\boxed{\text{CREATE FOLD}}$ has been pressed once, the editor changes its state and all editing functions other than vertical moves or searches are suspended until this key has been pressed again to complete the process of fold creation.

In a valid fold, lines between the top and bottom lines must be indented at least as far as the indentation of the fold to be created. When a fold is created, its indentation is determined by the position of the cursor within the line or, if there are non-blank characters to the left of this position, the leftmost non-blank character within the new fold, whichever is the less.

Once a fold has been created, it is good practice to add a comment by inserting text after the fold marker. This text is known as the fold header. Text may be moved from the fold header to the following line by using $\boxed{\text{RETURN}}$ or in the opposite direction by using $\boxed{\text{DELETE}}$ at the start of the line.

A created fold has an indentation associated with it, given by the indentation of the fold marker when it is closed, and the indentation of the creases when it is open. It is not possible to insert text to the left of this indentation.

An empty fold can be created above the current line by pressing $\boxed{\text{CREATE FOLD}}$ twice in succession.

A fold can be removed by placing the cursor on a fold line and pressing $\boxed{\text{REMOVE FOLD}}$. The fold contents are inserted between the lines above and below the fold.

If $\boxed{\text{CREATE FOLD}}$ is pressed accidentally the fold must be completed by pressing $\boxed{\text{CREATE FOLD}}$ again. The resulting empty fold may then be removed.

1.3.9 Deleting lines

DELETE LINE deletes the current line from the document. If this is a fold line, the fold and all its contents are deleted. Since this makes DELETE LINE a very powerful operation, it should be used with care.

There is a function RESTORE LINE to undo a deletion, restoring the last deleted line at the current position in the document. A stack of deleted lines is maintained by the editor and so these lines may be restored in the reverse order of deletion.

1.3.10 Moving and copying lines

Often when using an editor it is necessary to make structural changes to the text, moving lines and blocks of lines around. In F the representation of folds as lines on the screen means that substantial structural changes can be made to a document in the same manner as the reorganisation of lines. An individual line can be picked up, or a block of lines can be folded and then picked up.

The functions COPY LINE and MOVE LINE are used to copy and move sections of the document from one place to another. A text line or fold can be duplicated with the COPY LINE function, or moved to another position in the document using MOVE LINE.

COPY LINE duplicates the current line, inserting the copy in the text.

Two presses of MOVE LINE are needed to move a line from one part of the document to another; one to pick up the line, and one to put it down. If a sequence of lines is to be moved, the lines should be folded up first. A buffer (the 'move buffer') is used to store the line between the two operations. There is no need to go and put the line down immediately; the buffer will be retained until the next press of MOVE LINE, which may be after switching to another file.

Using the above keys, it is difficult to collect a number of different parts of a document before putting them down together. Here PICK LINE and COPY PICK are more appropriate. These make use of a different buffer (the 'pick buffer') that is cumulative. This enables the user to gather together, in the buffer, various pieces of text that can be put down in one place. PUT is used to put down the text in the buffer, which is emptied at the same time.

PICK LINE is used to pick up a line, which may be a fold line, so that it may be moved to another place in the current document, or in another document. It removes the current line from the document and appends it to the end of the pick buffer.

COPY PICK is used to copy a line, which may be a fold line, so that it may be moved to another place. It makes a copy of the current line and appends it to the end of the pick buffer. As the document is not changed it may be used in `Browsing` state.

PUT puts down the contents of the pick buffer at the current position in the document. It inserts, above the current line, the sequence of lines placed in the pick buffer using PICK LINE and COPY PICK. The pick buffer is cleared. If there are no lines in the pick buffer PUT has no effect on the document.

1.3.11 Search and replace

NEW SEARCH and NEW REPLACE allow the user to define the current search and replace strings, respectively. These strings should be entered on the message line in response to the relevant prompts. Each string is terminated by RETURN. When a new search string is defined, a search is then performed for the next matching occurrence of the search string in the current view. When a new replace string is defined and the cursor is on a match of the search string, this is replaced by the replace string.

When the strings are defined it is possible to locate the next instance of a copy of the search string in a forward scan from the cursor position by pressing SEARCH. If it is required to replace this string by the replace string this may be done by pressing REPLACE with the cursor on the matched string.

The key LOCATE LINE may be used to locate a line by its number. If an empty string is inserted then the number of the current line will be displayed. If a string containing decimal digits is entered these will be turned into an integer and the line with this number will be located.

1.3.12 Defining and using keystroke macros

The key DEFINE MACRO can be used to define a sequence of keys (which are commonly going to be used together during a session) and assign the sequence to a single keystroke. Two presses of DEFINE MACRO are needed to define a key sequence; the required keys (which may not include DEFINE MACRO or CALL MACRO) should be pressed between the two presses of DEFINE MACRO. The sequence may contain up to 255 keys. Any previously defined macro is forgotten. The defined macro sequence may be invoked using the CALL MACRO key. The currently defined macro may be saved in a new fold above the current line by pressing SAVE MACRO. A saved macro may be recovered from a fold by pressing GET MACRO. A keystroke macro which is of permanent value may be stored in the startup file or elsewhere for future use. In the startup file such a macro may be given a special name which will associate it with one of the tool keys TOOL0 .. TOOL9.

1.3.13 Editing multiple files

Any number of files may be edited concurrently. Files being edited are organised as a ring. New files may be added to the ring or removed from it by command line commands, see below.

Note especially the ability to obey a **#include** line as a command, and so enable easy navigation of nested file structures. If **#include**, (case insensitive) is not allowed by the compiler or processor for the current language then **#include** may be inserted into the document invisibly by making it the heading on an empty fold.

To switch to the next file in the ring press NEXT FILE. To switch to the previous file in the ring press PREVIOUS FILE. A file is not written to disk by either of these operations, which merely perform a switch. The contents of delete, move and pick buffers are preserved across a switch of files.

If a command is issued which terminates the edit of the current file, and there are other files in the ring, then the next file in the ring becomes the current file.

A table of currently open files may be obtained by pressing LIST FILES. This takes the form of a sequence of **goto** commands identifying the files. By pressing OBEY on one of these lines the indicated file will immediately become current.

At any time the current file may be saved to disk by pressing SAVE.

1.3.14 Command line commands

Command line commands may be issued from the special purpose command window, which can act as a command history. Alternatively text lines written into the editing window can be obeyed as commands.

Press COMMAND to enter the command window. The editor is then in the implicit state **Commanding**. A single host command may then be issued, or one of a group of predefined internal F commands may be issued. To issue a host command whose name matches an internal F command precede its name by the command name **system**. To spawn another instance of the operating system use the appropriate host command (e.g. **command** in DOS). The names of F commands are not case sensitive. A command is obeyed by pressing OBEY with the cursor anywhere on the line. The line may be a text line or a fold or crease line.

The internal F commands are:

1 **insert** *filename*
Inserts a copy of the text in the named file above the current line. If issued from the command window the insertion will be in the file from which the command window was entered.

2 **save** *filename*
Saves the current file on disk, using the name given. If issued from the command window the file saved will be the file from which the command window was entered. If no name is given, writes it back to the file it was read from. See also SAVE.

3 **saveall**
Saves all currently open files.

4 **f** *filename*
Adds the named file to the ring of current files and makes it current. If it is already in the ring, the user is given the option of switching into the current copy or taking a fresh copy from the file and displaying it in the **Browsing** state.

5 **#include** *filename*
As **f**. The filename may optionally be enclosed between single or double quotes.

6 **get** *filename*
This command quits the edit of the current file without saving its current state and starts to edit the named file. As the edit is lost it is always safe to write this command onto a fresh line within the editing window and obey it from there.

7 **goto** *filename*
The named file, which must correspond to a file in the ring of current files becomes the current file.

8 **exit** *filename*
Saves the current file on disk, using the name given, and then removes this file from the ring of current files. The next file becomes current. If the file name is omitted the file is saved where it came from (equivalent to FINISH)

9 **exitall**
Performs **exit** on all currently open files and terminates the editor.

10 **quit**
Abandons the editing of the current file without saving it. As the edit is lost it is always safe to write this command onto a fresh line within the editing window and obey it from there. The next file becomes current.

11 **quitall**
Performs **quit** on all currently open files, thereby terminating the edit session without saving any edits made since previous saves.

12 **system** *host command*
Sends the host command to the host operating system. The word **system** may be omitted if the host command does not clash with an F command name. The behaviour of this command depends on the behavior of the **system** call in the library of the C system used to compile the F editor.

When an attempt is made to edit the same file twice, the editor will test for identity by looking for exact equality of file name. If identity is detected the user will be given the option to take a fresh copy of the most recently saved version or to go to the currently open copy. Extreme care is required if two copies of the same file are being edited, as the final state of the file on disk will depend on which version is saved last.

A command is issued by using $\boxed{\text{OBEY}}$ on any line of text in the command window or in an editing window. Commands issued from the command window are remembered in that window for the duration of the current session. After a command is obeyed in the command window a blank line is inserted above it and that blank line becomes the current line. A command history thus grows upwards in the command window. These commands may be remembered for future editing sessions by explicitly editing them into the file whose name is given in the `command` line of the startup file. This file is read in by F, when it starts up, as the initial state of the command window.

When issuing a command which may act on a file currently being edited it is important to issue a **`save`** or **`saveall`** to ensure that it acts on an up to date version. The key $\boxed{\text{SAVE}}$ may be used to save the current file in its default location.

After a host command has been executed the user is invited to press any key to continue, after allowing for screen output to be read.

If the F editor is running on a transputer board attached to the host, care must be taken to avoid issuing host commands which themselves use the transputer board. Such a command will reset the transputer and so the editing session may not then be resumed.

If the F editor is running on the host itself then arbitrary host commands including compilation and program building commands may be called from within the editor. If the host has sufficient memory this makes it possible for all program development activities to be performed from within the F editor. It is a matter of user preference whether or not this mode of working is adopted.

1.3.15 Language dependencies

The F editor is suitable for editing text files in a variety of languages, with appropriate commenting conventions. Text files are represented as ordinary host text files with crease lines converted into comment lines in one of the (extensible) set of supported languages. Crease lines are made up from a comment start string, 3 curly braces of the appropriate kind, arbitrary comment text and a comment terminator string. Each set of comment conventions is associated with a language name in the startup file whose name is given in the f section of the ITERM file.

In the startup file a line starting with the keyword **`foldmark`** should have a sequence of three strings in double quotes. These are respectively the language name, the comment start string and the comment terminator string. The comment start and terminator are restricted to a maximum of three characters each.

Examples:

```
foldmark     "Occam"  "--"  ""

foldmark     "C"  "/*"  "*/"

foldmark     "Pascal"  "(*"  "*)"

foldmark     "Latex"  "%%"  ""

foldmark     "C++"  "//"  ""
```

When using languages which allow nested comments, care may need to be taken to avoid terminating comments within the text of fold comments by accident.

When F starts up, the set of languages defined in the startup file becomes known, and when each edit starts a search is made for crease lines including 3 curly braces and matching one of the known language comment conventions. The language is determined by the contents of the file, and defaults to the language corresponding to the first **`foldmark`** line. The current language is shown in the status message. If it is required to edit a file without using the fold information the appropriate foldmark line should be removed from the startup file.

The key $\boxed{\text{COMMENT STYLE}}$ allows the style of crease comments that will be used when the current file is written to disk to be determined by offering the known languages in sequence, and finally allowing the user to define a new one. At each stage the process may be aborted by pressing $\boxed{\text{FINISH}}$ or the currently offered language accepted by pressing $\boxed{\text{OBEY}}$. New language styles defined in this way are available for any subsequent presses of $\boxed{\text{COMMENT STYLE}}$ in the current edit, but must be explicitly edited into the startup file if they are to be used at file input time in subsequent editing sessions.

Whenever a file is saved, its creases are written in the appropriate style.

Multiple files in different languages may be edited concurrently. The language of each current file is shown when the key $\boxed{\text{LIST FILES}}$ is pressed.

2 Editing keys - alphabetical reference

This chapter presents full definitions of the editing keys in alphabetical order of their names for reference. The names are related to actual keyboard codes in the ITERM file, and are shown for the principal keyboard types in the keyboard maps in appendix A.

BOTTOM OF FOLD

Places the cursor on the line displaying the bottom crease marker of the current enclosing fold, or, if already on such a crease, the fold enclosing the current one.

BROWSE

Used to set the editor into `Browsing` state, in which no changes may be made to the document. BROWSE is also used to end browsing state and return to `Editing` state. It switches the set of allowable key functions between the full set and a reduced set which does not allow any form of data input or any other operations which could change the content of the current file.

CALL MACRO

Invokes a sequence of keys defined using the DEFINE MACRO key. If no macro sequence has been defined, the key has no effect.

CLOSE FOLD

Closes the current enclosing fold. The opened/closed status of any folds inside this fold is remembered during the current edit. The closed fold line is placed on the line of the screen where the top crease was, unless the top crease was off the top of the screen, in which case the closed fold line appears at the top of the screen. The cursor is positioned on the closed fold line, at the same column position as it was before CLOSE FOLD was pressed. CLOSE FOLD has no effect if the current enclosing fold was opened with an ENTER FOLD operation, but a message is given to remind the user that EXIT FOLD should be used to get out of the current fold.

COMMAND

Moves to the command window derived at start up from the `command` file referenced from the startup file. By appropriate edits to the startup file the user has the option of using a single command file identified by an absolute path name from any directory, or alternatively a local copy. Any command may be edited into this window and obeyed. The user is thus able to maintain a history of commands in whatever way is most convenient.

The user has the choice of entering one of a set of predefined F internal commands, see 1.3.14, or a line of text that will be passed as a command to the host operating system. Press OBEY to obey the current line as a command. If obeyed in the command window, a blank line will be inserted above the command line and this will be the current line on next entry to the command window. After obeying a command in the command window, the current file (whose identity may have changed as a result of the command) will be displayed.

See 1.3.14 for further discussion of the command system, including a list of the F internal commands.

COMMENT STYLE

The user is given the option to change the style of comments that will be used to represent crease lines when the current file is next written to disk. The table of languages defined by `foldmark` lines in the startup file is displayed in the message line one line at a time. For each language the user is given the option to change the style to that language, to abandon the language change, or to move on to the next language, by pressing OBEY, FINISH, or any other key respectively.

After offering all known languages F then allows the user to define a new language. To the question `Language:` reply with an arbitrary language name terminated by RETURN. To the next two questions reply with the string to be used before the curly braces, and the possibly empty string which trerminates all comment lines, respectively.

Typical language conventions showing how these are combined are:

Language	top crease		bottom crease	
occam	`--{{{`		`--}}}`	
C	`/*{{{`	`*/`	`/*}}}`	`*/`
Pascal	`(*{{{`	`*)`	`(*}}}`	`*)`
LaTeX	`%%{{{`		`%%}}}`	
C++	`//{{{`		`//}}}`	

COPY LINE

Copies the current line and inserts the copy below the current line. If the line is a closed fold then all the text lines and nested folds in the fold are copied. COPY LINE has no effect if the current line is a top or bottom crease. The cursor is placed on the copy.

COPY PICK

Copies a line, which may be a closed fold line, so that it may be moved to another place in the document. It makes a copy of the current line and appends it to the end of the pick buffer. The cursor is then moved down one line.

As COPY PICK has no effect on the document, it may be used in `Browsing` state.

CREATE FOLD

The first use of CREATE FOLD inserts a fold creation mark above the current line, at the current column. The second use of CREATE FOLD inserts a new crease mark above the new current line, which may be above or below the fold creation mark. It then creates a fold containing the lines between these two marks. The fold is closed and the cursor is placed at the end of the fold line marker, where fold header text may be inserted.

Between the two presses of CREATE FOLD the editor is in `Creating` state and all editor functions except up and down cursor movement, scrolling, locating and searching are disallowed.

The indentation of the new fold is determined by the leftmost non-blank character in the fold or the leftmost column of the fold creation mark whichever is closest to the left margin. The lines to be enclosed within the new fold will all be sufficiently indented to fit into a fold at this indentation.

CURSOR DOWN

Moves the cursor down one line. On the bottom line of the screen it scrolls the screen one line down the current view, if there are lines in the current view below the screen, and the cursor remains in the same position on the screen.

CURSOR LEFT

Moves the cursor left one column, except in the leftmost column on the screen where it may cause the view to pan.

CURSOR RIGHT

Moves the cursor right one column, except in the rightmost column on the screen where it may cause the view to pan.

CURSOR UP

Moves the cursor up one line. On the top line of the screen it scrolls the screen one line up the current view, if there are lines in the current view above the screen, and the cursor remains in the same position on the screen.

DEFINE MACRO

Used to define a sequence of keys (which are going to be used together repeatedly) and assign the sequence to a single keystroke. Two presses of DEFINE MACRO are needed to define a key sequence; the required keys (which may not include DEFINE MACRO or CALL MACRO) should be pressed between the two presses of DEFINE MACRO. N.B. the keys are obeyed when defining the macro. The sequence may contain up to 255 keys. Any previously defined macro is forgotten. The defined macro sequence may be invoked using the CALL MACRO key.

DELETE

Deletes the character to the left of the cursor. The cursor, the character at the cursor and all subsequent characters on the line are moved left by one place.

If the cursor is in the leftmost column of the current enclosing fold, and the total length of the current line and the line above does not exceed the maximum allowed, DELETE concatenates the current line with the line above.

DELETE in the leftmost column has no effect if the current line is a fold line, top crease or bottom crease, or is a line following a fold line or bottom crease.

Spaces may be deleted before a closed fold marker symbol to change the indentation of the fold.

DELETE LINE

Removes the current line from the document, and places it at the end of the delete buffer. All the lines below the current line in the view are moved up by one line. DELETE LINE has no effect if the current line is a top crease or bottom crease. RESTORE LINE may be used to reverse the effect of this key.

DELETE RIGHT

Deletes the character at the cursor. All the characters to the right of the cursor are moved left by one place. The cursor does not move.

Character deletion has no effect when the character to be deleted is part of a marker symbol, or is to the left of the leftmost column of an open fold.

Spaces may be deleted before a closed fold marker symbol to change the indentation of the fold.

DELETE TO END OF LINE

Deletes all text from the character at the cursor, to the last significant character on the line, inclusive. The cursor does not move.

DELETE WORD LEFT

Deletes the symbol to the left of the cursor. The deletion is governed by the following rules:

- A symbol is a sequence of alphanumeric characters, or of non-space non-alphanumeric characters, A line contains a sequence of symbols, separated by zero or more spaces. A symbol starting position is the position of the first character in a symbol.
- If the cursor is on or to the left of the first significant (non-space) character on the line, the characters from the cursor position to the current indentation are deleted. The cursor will move to the current indentation.
- If the cursor is to the right of the character following (immediately to the right of) the last significant character on the line, the cursor will move to the character following the last significant character on the line.
- In all other cases the cursor will move to the first symbol starting position to the left of the current cursor position, deleting all intervening characters.

DELETE WORD RIGHT

Deletes the symbol to the right of the cursor. The deletion is governed by the following rules:

- A symbol is a sequence of alphanumeric characters, or of non-space non-alphanumeric characters, A line contains a sequence of symbols, separated by zero or more spaces. A symbol starting position is the position of the first character in a symbol.
- If the cursor is to the left of the indentation of the current fold nothing will happen.
- If the cursor is to the left of the first significant (non-space) character on the line, all characters between the cursor the first significant character on the line will be deleted.
- If the cursor is on or between the last symbol starting position on the line, and the last significant character on the line, all characters up to and including the last significant character on the line will be deleted.
- If the cursor is to the right of the last significant character on the line, the cursor will not move.
- In all other cases all characters between the cursor and the first symbol starting position to the right of the current cursor position, will be deleted.

DISPLAY KEY

Displays at the cursor position a sequence of decimal integer values separated by commas corresponding to a sequence of key depressions terminated by a space. If the cursor is not in a writable position the display is suppressed. This key is designed to facilitate the construction of keyboard sections for ITERM files. The codes are the raw values received by the editor from the keyboard.

END OF LINE

Places the cursor immediately to the right of the last significant character on the current line (i.e. the last non-blank character). If the line is too long for the width of the screen the view will pan if necessary.

ENTER FOLD

When used on a fold line, clears the screen and displays the contents of the fold between the top and bottom creases. The display is adjusted to the left so that the top and bottom marker symbols start in the leftmost column. The cursor is positioned in the leftmost column of the second line on the screen. This then becomes the current view and it is not possible to move outside the confines of the fold until a corresponding EXIT FOLD has been done.

EXIT FOLD

Reverses the effect of the most recent ENTER FOLD, closing the fold, but not any open folds contained within it. The closed fold line is positioned on the same position as it was when the ENTER FOLD was done.

FINISH

Finishes an edit by saving the file and returns to the next file or to host operating system level. Also used to leave the HELP display, or the command window or to escape from a COMMENT STYLE operation.

GET MACRO

Applied to a fold whose comment starts `Key macro`, copies the keystroke sequence constructed from a sequence of integer values in the fold as the current key macro. The sequence may then be invoked using the CALL MACRO key.

HELP

Displays a map of the system function keys, which is read from the `helpfile` identified in the startup file or determined by changing the filename extension of the ITERM file to `.hlp`. It also displays a version identity message. Return to the edit by pressing FINISH.

LINE DOWN

Moves the screen one line down the current view, if there are lines in the current view below the screen without moving the cursor on the screen. Otherwise moves cursor down one line.

LINE UP

Moves the screen one line up the current view, if there are lines in the current view above the screen without moving the cursor on the screen. Otherwise moves cursor up one line.

LIST FILES

Inserts above the current line a sequence of `goto` commands identifying the currently opened files. Each line also identifies the language name of the convention used for its crease comments. These lines are then available for application of the OBEY key to switch editing to one of the other files.

LOCATE LINE

A question is displayed in the message line asking for a line number. If answered with RETURN or any string not starting with a digit the number of the current line will be displayed. If answered with an integer the editor locates to the line with that number in the current file.

MOVE LINE

Used to move a line, which may be a fold line, to another place in the document. A buffer is associated with MOVE LINE. If the buffer is empty, MOVE LINE removes the current line from the document and puts it in the buffer. If there is a line in the buffer, MOVE LINE removes the line from the buffer, puts it into the document on the line above the current line and places the cursor on it.

NEW REPLACE

The prompt **`New replace string :`** on the message line requests a new replace string; the reply to this is a replace string terminated by RETURN. If the cursor is on a match of the current search string, this string is replaced by the current replace string. If it is not on a match a message is displayed and no change occurs. The new replace string becomes the current replace string used in subsequent REPLACE operations.

NEW SEARCH

The prompt **`Search for :`** on the message line requests a new search string; the reply to this should be a search string terminated by RETURN. This becomes the current search string. A search is performed downwards in the current view for a string matching the search string. The search starts at the character to the right of the cursor position. The same search and replace strings are used by the SEARCH and REPLACE keys.

NEXT FILE

Switches to the next file in the ring of files, if there is one. This becomes the current file. Does not save the old current file to disk. The complete state of the current file is preserved.

OBEY

Obeys the current text line as a command. This is primarily for use in the command window, but can also be used in the editing window, where **`#include "filename"`**, behaves as **`f filename`**, to facilitate navigation into included files.

Care must be taken that the command obeyed does not use resources used by the editor, such as a transputer board.

OPEN FOLD

On a fold line opens the fold and inserts the contents of the fold into the current view, surrounded by top and bottom creases. The top crease appears on the line of the screen where the closed fold line was before OPEN FOLD was pressed. The cursor does not move.

PAGE DOWN

Moves the screen one page down the current view, or to the bottom of the current view, whichever is the nearest. The cursor does not move.

PAGE UP

Moves the screen one page up the current view, or to the top of the current view, whichever is the nearest. The cursor does not move.

PICK LINE

Picks up a line, which may be a fold line, so that it may be moved to another place in the document. It removes the current line from the document and appends it to the end of the pick buffer.

PREVIOUS FILE

Switches to the previous file in the ring of files, if there is one. This becomes the current file. Does not save the old current file to disk. The complete state of the current file is preserved.

PUT

Puts down the contents of the pick buffer above the current line. Inserts the sequence of lines placed in the pick buffer using PICK LINE and COPY PICK. The pick buffer is cleared. If there are no lines in the pick buffer PUT has no effect on the document.

REFRESH

Repaints the entire screen from the stored representation of the current view. If running in a windowed environment this may have the effect of taking notice of a change of window size.

REMOVE FOLD

On a fold line, opens the fold and removes the top and bottom creases, inserting the contents of the fold into the current view at an appropriate indentation.

REPLACE

If the cursor is on a match of the current search string, replaces this string by the current replace string. If it is not on a match a message is displayed and no change occurs.

RESTORE LINE

Each use of this key will restore the last line placed in the delete buffer by DELETE LINE, inserting it above the current line in the document. The restored line is removed from the delete buffer. Other previously deleted lines remain there.

RETURN

Splits a text line in two at the cursor position and creates a new line on which are placed the cursor, the character at the cursor and any subsequent characters on the line. The new line is then indented by inserting spaces until the cursor is in the same column as the first significant character of the line above.

RETURN may be used within the text of a top crease line.

RETURN will insert a blank line above the current line when the cursor is before or on the first significant character of a line.

RETURN will insert a blank line below the current line when the cursor is after the last significant character of a line.

RETURN has no effect within the fold marker on a fold line, top crease, or bottom crease.

SAVE

The current file is saved to disk.

SAVE MACRO

The sequence of keys defined using DEFINE MACRO is saved in a fold inserted above the current line. The fold comment `Key macro` is written on this fold and the cursor is positioned so that a name may be added to identify this particular saved macro sequence.

SEARCH

If a non-empty search string has been defined by NEW SEARCH, a search forward is made through the lines of the current view, for a string exactly matching the current search string. The search starts at the character to the right of the cursor position. When a match is found the cursor moves to the match, which may involve opening enclosed folds, and the current replace string is shown in the message line, so that the user may decide whether or not to replace the matched string.

START OF LINE

Moves the cursor to the first significant character of the current line (i.e. the first non-blank character). If necessary the view will pan.

TO LOWER

If the character at the cursor is an upper case letter, converts it to lower case. Then moves the cursor one place to the right.

TO UPPER

If the character at the cursor is a lower case letter, converts it to upper case. Then moves the cursor one place to the right.

TOGGLE TABS

Inverts the state of an internal flag which determines whether or not sequences of multiples of 8 leading spaces are represented in the output file as TAB characters. The editor state message shows a `!` character if tabs are being generated. The flag will be initially on if and only if any TABs are found when the file is read.

TOOL0 ... TOOL9

The tool keys are used to apply keystroke macros which have been saved in the startup file. When the editor starts, the startup file is searched for folds whose headings start with a digit and the text `Key macro`. The contents of each such fold is treated as a saved macro and the sequence of keystrokes so defined will be generated when the corresponding tool key is subsequently pressed.

TOP OF FOLD

Places the cursor on the line displaying the top crease marker of the current enclosing fold, or, if already on such a crease, the fold enclosing the current one.

WORD LEFT

Moves the cursor one symbol left, panning if necessary. The move is governed by the following rules:

- A symbol is a sequence of alphanumeric characters, or of non-space non-alphanumeric characters, A line contains a sequence of symbols, separated by zero or more spaces. A symbol starting position is the position of the first character in a symbol.
- If the cursor is at or to the left of the indentation of the current enclosing fold, the cursor will move to the extreme left of the line.
- If the cursor is on or to the left of the first significant (non-space) character on the line, the cursor will move to the current indentation.
- If the cursor is to the right of the character following (immediately to the right of) the last significant character on the line, the cursor will move to the character following the last significant character on the line.
- In all other cases the cursor will move to the first symbol starting position to the left of the current cursor position.

WORD RIGHT

Moves the cursor one symbol right, panning if necessary. The move is governed by the following rules:

- A symbol is a sequence of alphanumeric characters, or of non-space non-alphanumeric characters, A line contains a sequence of symbols, separated by zero or more spaces. A symbol starting position is the position of the first character in a symbol.
- If the cursor is to the left of the first significant (non-space) character on the line, the cursor will move to the first significant character on the line.
- If the cursor is on or between the last symbol starting position on the line, and the last significant character on the line, the cursor will move to the character following (immediately to the right of) the last significant character on the line.
- If the cursor is to the right of the last significant character on the line, the

cursor will move to the next multiple of the tabulation increment.

- In all other cases the cursor will move to the first symbol starting position to the right of the current cursor position.

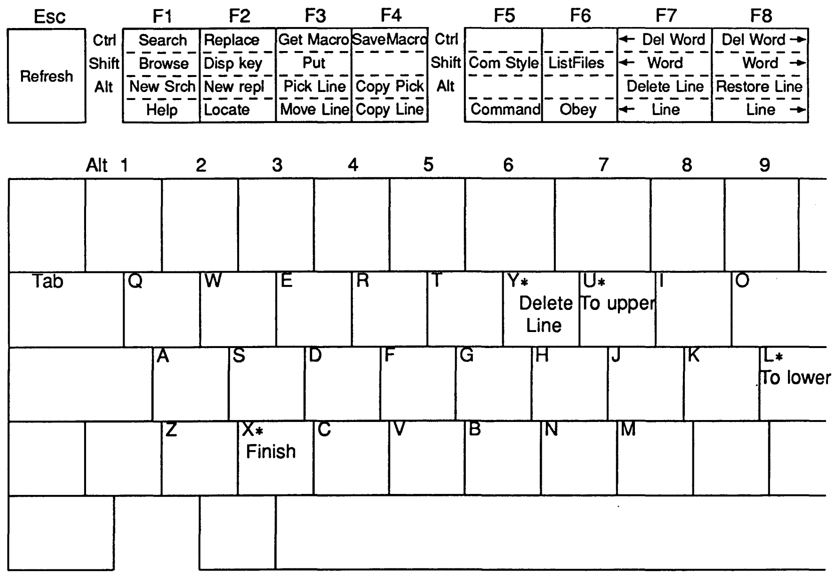
Appendices

A Keyboard allocations

A.1 IBM PC function keys

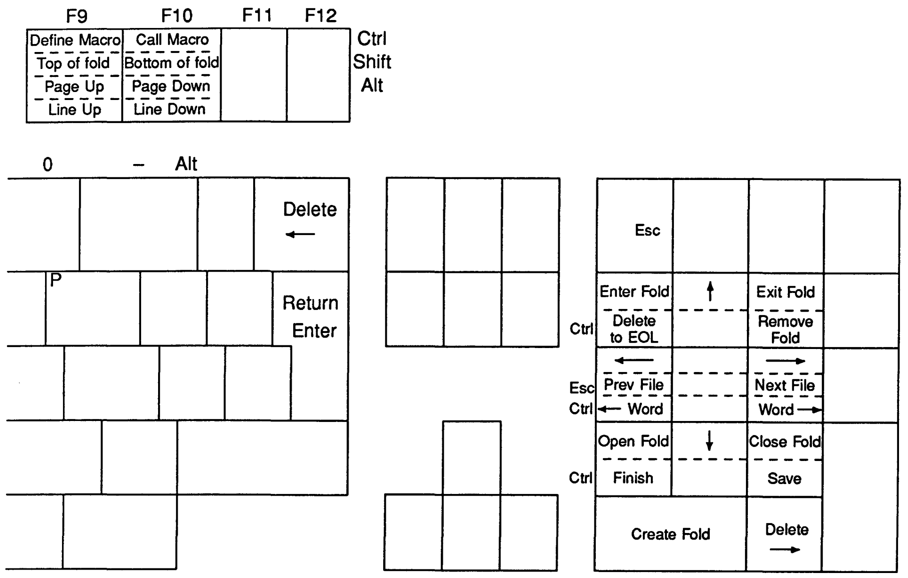
	F1	F2
Ctrl	Search	Replace
Shift	Browse	Display key
Alt	New Search	New Replace
	Help	Locate
Ctrl	Get Macro	Save Macro
Shift	Put	
Alt	Pick Line	Copy Pick
	Move Line	Copy Line
Ctrl		
Shift	Comm style	List files
Alt		
	Command	Obey
Ctrl	←Del Word	Del Word→
Shift	← Word	Word →
Alt	Delete Line	Restore Line
	← Line	Line →
Ctrl	Define Macro	Call Macro
Shift	Top of fold	Bottom of fold
Alt	Page Up	Page Down
	Line Up	Line Down
	F9	F10

A.2 IBM PC keyboard layout


Esc
Refresh
F1 F2 F3 F4
Ctrl Shift Alt
Search | Replace | Get Macro | SaveMacro
Browse | Disp key | Put
New Srch | New repl | Pick Line | Copy Pick
Help | Locate | Move Line | Copy Line
F5 F6 F7 F8
Ctrl Shift Alt
← Del Word | Del Word →
Com Style | ListFiles | ← Word | Word →
Delete Line | Restore Line
Command | Obey | ← Line | Line →
Alt 1 2 3 4 5 6 7 8 9
Tab Q W E R T
Y* Delete Line
U* To upper
I O
A S D F G H J K
L* To lower
Z
X* Finish
C V B N M

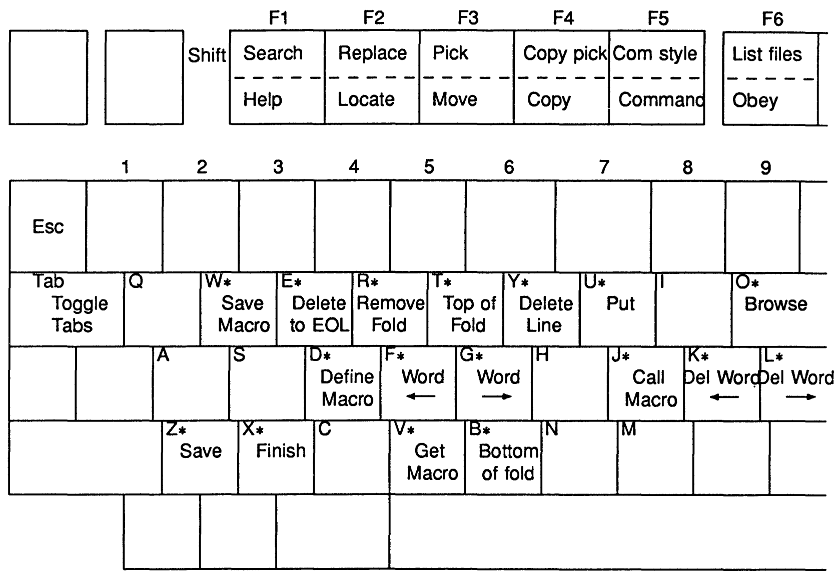
∗ Ctrl + key

Note that additional keys or combinations may be defined by modifying the ITERM file (see section B).



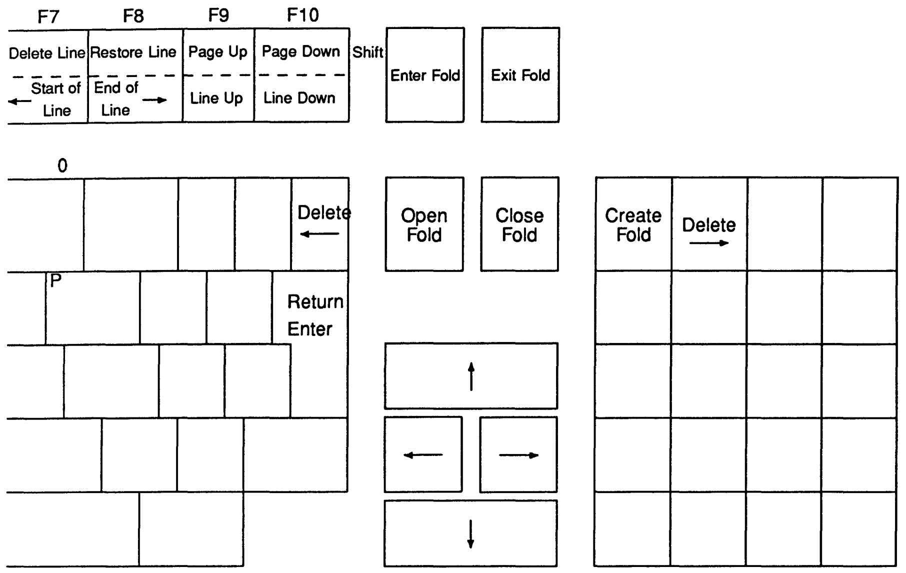
Esc Esc	= Refresh
Tab	= Toggle write tabs
Alt 0	= Tool0
..	..
Alt 9	= Tool9

A.3 NEC PC keyboard layout



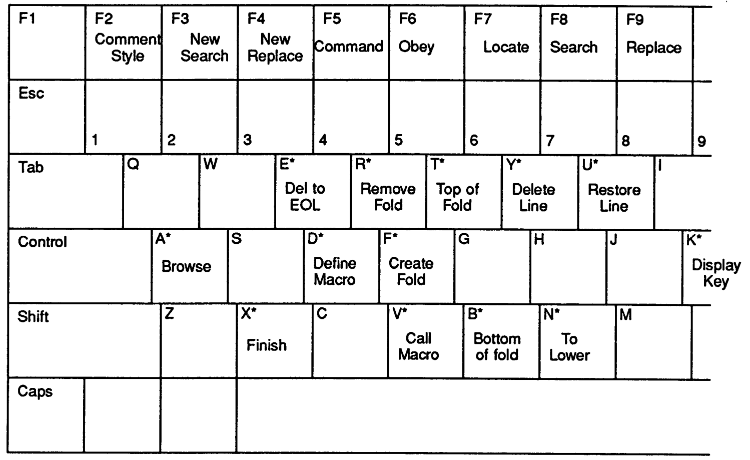
* Ctrl + key

Esc Esc	= Refresh
Esc F1	= New Search
Esc F2	= New Replace
Esc SP	= Display Key
Esc 0	= Tool0
..	..
Esc 9	= Tool9

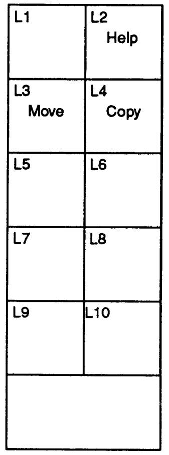


Esc	↑	= To upper
Esc	↓	= To lower
Esc	←	= Previous file
Esc	→	= Next file

A.4 SUN4 keyboard layout

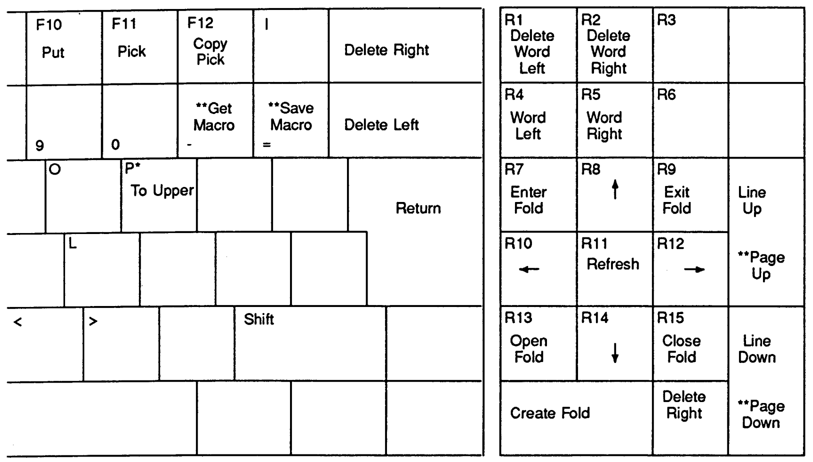


LH keypad



* = Ctrl + key

** = Esc + key



* = Ctrl + key

** = Esc + key

Esc 0	= Tool 0
...	...
Esc 9	= Tool 9
Tab	= Toggle tabs
Esc R2	= Del to EOL
Esc R4	= Start of line
Esc R5	= End of line
Esc R8	= Line up
Esc R10	= Previous file
Esc R11	= List files
Esc R12	= Next file
Esc R13	= Finish
Esc R14	= Line down
Esc R15	= Save
Esc SP	= Display key

Note: SUN3 keyboards have a subset of these keys

B ITERM - terminal configuration file

This appendix describes the format of ITERM files; it is included for people who need to write their own ITERM because they wish to change keyboard mappings or are using terminals that are not supported by the standard ITERM file supplied.

ITERMs are ASCII text files that describe the control sequences required to drive terminals. Screen oriented applications that use ITERM files are terminal independent.

ITERM files are similar in function to the UNIX *termcap* database and describe input from, as well as output to, the terminal. They allow applications that use function keys to be terminal independent and configurable.

B.1 The structure of an ITERM file

An ITERM file consists of three or more sections. The standard sections are the *host*, *screen* and *keyboard* sections. They are introduced by a line beginning with the section letters '`H`', '`S`' or '`K`'. Case is unimportant and the rest of the line is ignored. A section is terminated by a line beginning with the letter '`E`'. The *host* section must appear first; other sections may appear in any order in the file. Sections must be separated by at least one blank line. The standard sections consist of a number of lines beginning with a digit.

In order to support the F editor a fourth section, the F section is required. The starts with a line beginning '`F`' and ends with the next line beginning '`E`'.

The syntax of the lines that make up the body of the standard sections is best described in an example:

```
    3:34,56,23,7.   comments
```

Each line starts with the index number followed by a colon and a list of numbers separated by commas. Each line is terminated by a full stop ('`.`') and anything following it is treated as a comment. Spaces are not allowed in the data string and an entry cannot be split across more than one line.

Comment lines, beginning with the character '`#`', may be placed anywhere in an ITERM file. Extra blank lines in the file are ignored.

The index numbers in each section correspond to an agreed meaning for the data. In the following sections the meaning of the data in each of the three sections is described in detail.

B.2 The host definitions

B.2.1 ITERM version

This item identifies an ITERM file by version. It provides some protection against incompatible future upgrades. The F editor requires version 3 or above.

e.g. `1:3.`

B.2.2 Screen size

This item allows applications to find out the size of the terminal at startup time. The data items are the number of columns and rows, in that order, available on the current terminal.

e.g. `2:80,25.`

Screen locations should be numbered from 0, 0 by the application. Terminals which use addressing from 1, 1 can be compensated for in the definition of goto X, Y.

B.3 The screen definitions

The lists of values in the screen section represent control codes that perform certain operations; the data values are ASCII codes to send to the display device.

ITERM version 3 or later defines the indices given in table B.1. These definitions are used in the example ITERM file; for a complete listing of the file see section B.7.

For example, an entry like: '`8:27,91,75.`' indicates that an application should output the ASCII sequence '`ESC [ K`' to the terminal output stream to clear to end of line.

If a terminal initialise code is defined it will be send to the screen at startup time. If a keyboard needs to be switched to application keypad mode this may be coded here. A terminal finalisation sequence sent before F closes down may also be defined.

B.3.1 Goto X Y processing

The entry for 5, 'goto X Y', requires further interpretation by the application.
A typical entry for 'goto X Y' might be:

```
5:27,-11,32,-21,32
```

Index	Screen operation
1	cursor up
2	cursor down
3	cursor left
4	cursor right
5	goto x y
6	insert character
7	delete character at cursor
8	clear to end of line
9	clear to end of screen
10	insert line
11	delete line
12	ring bell
13	home and clear screen
14	terminal initialise
15	terminal finalise
	(other codes not used by F)

Table B.1 ITERM screen operations

The negative numbers relate to the arguments required for X and Y.

..., *-ab*, *nn*, ...

where: *a* is the argument number (i.e. 1 for X, 2 for Y).

b controls the data output format.
If *b*=1 output is an ASCII byte (e.g. 33 is output as `!`).
If *b*=2 output is an ASCII number (e.g. 33 is output as `3 3`).

nn is added to the argument before output.

As a complete example, consider the following ITERM entry in the screen section:

```
5:27,91,-22,1,59,-12,1,72. ansi cursor control
```

This would instruct an application wishing to move the terminal cursor to X=14, Y=8 (relative to 0,0) to output the following bytes to the screen:

```
Bytes in decimal: 27   91   57   59   49   53   72

Bytes in ASCII:   ESC  [    9    ;    1    5    H
```

B.4 The keyboard definitions

Each index represents a single 'cooked' keystroke. The data specified after each index defines the sequence of key values associated with that keystroke. Multiple entries for the same index indicate alternative keystroke sequences for the operation. The first integer is the internal 'cooked' keystroke code less 200. Cooked keystroke codes may be observed by opening a keystroke macro fold which was created by entering keys in `Defining macro` state. The sequence of integers between the colon and the dot are ASCII values of actual codes received by the editor.

The meanings of the keystrokes are defined in the specification of tools using ITERM.

The layout of typical physical keyboards is shown in appendix A. Sequences of key values which do not correspond to any of these cooked keystrokes are passed to programs as individual values.

B.5 F editor specific items

These items are recognised by the keyword that starts the line. The value of the named parameter is defined to be the string which follows it.

`helpkey` A message leading the user to the on-line Help facility by defining what key to press to enter the help window.

`helpfile` The name of the file containing the text of the help window. This item is only needed if the keyboard help text is not in a file whose name is derived from that of the ITERM file by changing its suffix to ".hlp".

`startupfile` The name of the file containing further F specific items. This name is superseded by the environment variable **`FSTART`**, if this is defined.

The items that may appear in a startup file are:

`command` The name of the file containing the initial text of the command window at the start of an editing session.

`foldmark` This has three string parameters. The first is a name to be used for a set of

comment conventions. The second is the comment start string. The third is the comment terminating string. See 1.3.15.

B.6 Setting up the ITERM environment variable **`FTERM`**

To use an ITERM the application has to find and read the file. The F editor uses an environment variable called '**`FTERM`**' for this purpose and so this should be set up with the pathname and filename of the file as its value. For example, under MS-DOS the command could be:

```
set FTERM=C:\F\PCFOLD.ITM
```

B.7 Example ITERMs

This is an ITERM file for F on the IBM PC using an enhanced ANSI screen driver, which supports character and line insert and delete.

```
#  ------------------------------------------------
#
#  IBM PC (ANSI) ITERM data file for
#  Support for f  JW 10-01-91
#  Special care needed on screen codes 6, 7, 9, 10, 11,
#
#  ------------------------------------------------
host
1:3.
2:80,25.
end host
screen
1:27,91,65.                  ESC [ A     up
2:27,91,66.                  ESC [ B     down
3:27,91,68.                  ESC [ D     left
4:27,91,67.                  ESC [ C     right
5:27,91,-22,1,59,-12,1,72.   ESC [ y+1 ; x+1 H  goto x y
6:27,91,64.                  ESC [ @     insert char
7:27,91,80.                  ESC [ P     delete char
8:27,91,75.                  ESC [ K     clear to end of line
9:27,91,74.                  ESC [ J     clear to end of screen


10:27,91,76.                 ESC [ L     insert line
11:27,91,77.                 ESC [ M     delete line
12:7.                        CTRL G      beep
13:27,91,50,74.              ESC [ 2 J   clear screen
end of screen stuff
keyboard            PC Key          Code            F usage
6:0,72.             Up              Nul H           up
7:0,80.             Down            Nul P           down
```

```
8:0,75.             Left           Nul K            left
9:0,77.             right          Nul M            right
10:8.               Del            BSP              delete char left
11:0,83.            KPAD .         Nul S            delete char right
12:0,110.           ALT F7         NUL n            delete line
12:25.              Ctrl-Y         Ctrl-Y           delete line
13:0,111.           ALT F8         NUL o            restore line
14:0,65.            F7             NUL A            start of line
15:0,66.            F8             NUL B            end of line
16:0,61.            F3             NUL =            move
17:0,62.            F4             NUL >            copy
18:0,67.            F9             NUL C            line up
19:0,68.            F10            NUL D            line down
20:0,112.           ALT F9         NUL p            page up
21:0,113.           ALT F10        NUL q            page down
22:0,82.            KPAD 0         NUL R            create fold
23:0,132.           Ctrl PgUp      NUL <132>        remove fold
23:18.              Ctrl R         CTRL R           remove fold
24:0,79.            End            NUL O            open fold
25:0,81.            PgDn           NUL Q            close fold
26:0,71.            Home           NUL G            enter fold
27:0,73.            PgUp           NUL I            exit fold
28:27,27.           Esc Esc        ESC ESC          refresh
29:0,94.            Ctrl F1        NUL ^            search
30:0,95.            Ctrl F2        NUL _            replace
31:0,117.           Ctrl End       NUL u            finish
31:24.              Ctrl X         CTRL X           finish
32:0,104.           Alt F1         NUL h            new search
33:0,105.           Alt F2         NUL i            new replace
34:0,59.            F1             NUL ;            help
35:0,60.            F2             NUL <            locate
36:0,63.            F5             NUL ?            command
37:0,97.            CTRL F4        NUL a            save macro
38:0,96.            Ctrl F3        NUL `            get macro
39:0,64.            F6             NUL @            obey
40:0,129.           Alt 0          NUL <129>        tool 0
41:0,120.           Alt 1          NUL x            tool 1
42:0,121.           Alt 2          NUL y            tool 2
43:0,122.           Alt 3          NUL z            tool 3
44:0,123.           Alt 4          NUL {            tool 4
45:0,124.           Alt 5          NUL |            tool 5
46:0,125.           Alt 6          NUL }            tool 6
47:0,126.           Alt 7          NUL ~            tool 7
48:0,127.           Alt 8          NUL DEL          tool 8
49:0,128.           Alt 9          NUL <128>        tool 9
50:0,90.            Shift F7       NUL Z            word left
50:0,115.           Ctrl KPAD 4    NUL s            word left
51:0,91.            Shift F8       NUL [            word right
51:0,116.           Ctrl KPAD 6    NUL t            word right
52:0,100.           Ctrl F7        NUL d            del word left
53:0,101.           Ctrl F8        NUL e            del word right
54:0,119.           Ctrl Home      NUL w            del to eol
54:5.               Ctrl E         CTRL E           del to eol
55:0,92.            Shift F9       NUL \            top of fold
55:20.              Ctrl T         CTRL T           top of fold
```

```
56:0,93.            Shift F10      NUL ]            bottom of fold
56:2.               Ctrl B         CTRL B           bottom of fold
57:0,88.            Shift F5       NUL X            comment style
58:0,89.            Shift F6       NUL Y            list files
59:0,106.           Alt F3         NUL j            pick
60:0,107.           Alt F4         NUL k            copy pick
61:0,86.            Shift F3       NUL V            put
62:9.               Tab            TAB              toggle tabs
63:0,85.            Shift F2       NUL U            display key
64:27,0,75.         Esc Left       ESC NUL K        previous file
65:27,0,77.         Esc Right      ESC NUL M        next file
66:21.              Ctrl U         CTRL U           to upper
67:12.              Ctrl L         CTRL L           to lower
68:0,84.            Shift F1       NUL T            browse
68:15.              Ctrl O         CTRL O           browse
69:0,118.           Ctrl PgDn      NUL v            save
70:0,102.           Ctrl F9        NUL f            define macro
70:4.               Ctrl D         CTRL D           define macro
71:0,103.           Ctrl F10       NUL g            call macro
71:10.              Ctrl J         CTRL J           call macro
#73                 bad
end of keyboard stuff
f editor specific items
startupfile   "fold.stp"
helpkey "Press F1 for Help"
end f
```

The screen section above assumes that the host screen driver accepts the ANSI escape sequences shown. The standard Microsoft ANSI.SYS in DOS does not implement line/character insert/delete sequences, and clear to end of screen from any position. Either a special screen driver (e.g. BANSI.SYS) must be installed which does implement these codes or an appropriately modified ISERVER must be used.

This is an ITERM file for SUN4 using SUNOS and SUNVIEW .

```
#  -------------------------------------------------
#
#    SUNOS/SUNVIEW Version of ITERM for F editor
#
#  -------------------------------------------------

host section
1:3.                          version
2:80,50.                      screen size
end of host section

screen section
1:27,91,65.                 ESC [ A     up
2:27,91,66.                 ESC [ B     down
3:27,91,68.                 ESC [ D     left
4:27,91,67.                 ESC [ C     right
5:27,91,-22,1,59,-12,1,72.  ESC [ y+1 ; x+1 H  goto x y
6:27,91,64.                 ESC [ @     insert char
7:27,91,80.                 ESC [ P     delete char
```

```
8:27,91,75.                 ESC [ K     clear to end of line
9:27,91,74.                 ESC [ J     clear to end of screen
10:27,91,76.                ESC [ L     insert line
11:27,91,77.                ESC [ M     delete line
12:7.                       CTRL G      bell
13:27,91,50,74.             ESC [ 2 J   clear screen
end of screen section

keyboard section
#                           SUN KEY         code              F usage
6:27,91,65.                 # Up            ESC [ A           cursor up
7:27,91,66.                 # Down          ESC [ B           cursor down
8:27,91,68.                 # Left          ESC [ D           cursor left
9:27,91,67.                 # Right         ESC [ C           cursor right
10:8.                       # Del           BSP            delete char left
11:127.                     # KPAD .        DEL            delete char right
11:27,91,50,52,57,122.      # KPAD .        ESC [249z      delete char right
12:25.                      # Ctrl-Y        Ctrl-Y            delete line
13:21.                      # Ctrl-U        Ctrl-U            restore line
14:27,27,91,50,49,49,122. # Esc R4          ESC [211z         start of line
15:27,27,91,50,49,50,122. # Esc R5          ESC [212z         end of line
16:27,91,49,57,52,122.    # L3              ESC [194z         move
17:27,91,49,57,53,122.    # L4              ESC [195z         copy
#18:27,91,50,48,48,122.    # L9              ESC [200z         line up
18:27,91,50,53,51,122.    # Kpad +          ESC [253z         line up
18:27,27,91,65.           # ESC Up          ESC ESC [ A       line up
#19:27,91,50,48,49,122.    # L10             ESC [201z         line down
19:27,91,50,53,48,122.    # Enter           ESC [250z         line down
19:27,27,91,66.           # ESC Down        ESC ESC [ B       line down
20:27,27,91,50,53,51,122. # ESC Kpad +      ESC ESC [253z     page up
21:27,27,91,50,53,48,122. # ESC Enter       ESC ESC [250z     page down
22:27,91,50,52,55,122.    #                 ESC [247z         create fold
22:6.                     # Ctrl-F          Ctrl-F            create fold
23:18.                    # Ctrl-R          Ctrl-R            remove fold
24:27,91,50,50,48,122.    # KPAD 1          ESC [220z         open fold
25:27,91,50,50,50,122.    # KPAD 3          ESC [222z         close fold
26:27,91,50,49,52,122.    # KPAD 7          ESC [214z         enter fold
27:27,91,50,49,54,122.    # KPAD 9          ESC [216z         exit fold
28:27,91,50,49,56,122.    # KPAD 5          ESC [215z         refresh
29:27,91,50,51,49,122.    # F8              ESC [231z         search
30:27,91,50,51,50,122.    # F9              ESC [232z         replace
31:24.                    # Ctrl-X          Ctrl-X            finish
31:27,27,91,50,50,48,122.  # ESC KPAD 1     ESC ESC [220z     finish
32:27,91,50,50,54,122.    # F3              ESC [226z         new search
33:27,91,50,50,55,122.    # F4              ESC [227z         new replace
34:27,91,49,57,51,122.    # L2              ESC [193z         help
34:27,91,49,57,54,122.    # L5              ESC [196z         help
35:27,91,50,51,48,122.    # F7              ESC [230z         locate line
36:27,91,50,50,56,122.    # F5              ESC [228z         command
37:27,61.                 # ESC =           ESC =             save macro
38:27,45.                 # ESC -           ESC -             get macro
39:27,91,50,50,57,122.    # F6              ESC [229z         obey
40:27,48.                 # ESC 0           ESC 0             tool 0
41:27,49.                 # ESC 1           ESC 1             tool 1
42:27,50.                 # ESC 2           ESC 2             tool 2
```

```
43:27,51.                   # ESC 3         ESC 3           tool 3
44:27,52.                   # ESC 4         ESC 4           tool 4
45:27,53.                   # ESC 5         ESC 5           tool 5
46:27,54.                   # ESC 6         ESC 6           tool 6
47:27,55.                   # ESC 7         ESC 7           tool 7
48:27,56.                   # ESC 8         ESC 8           tool 8
49:27,57.                   # ESC 9         ESC 9           tool 9
50:27,91,50,49,49,122.      # R4            ESC [211z       word left
51:27,91,50,49,50,122.      # R5            ESC [212z       word right
52:27,91,50,48,56,122.      # R1            ESC [208z       del word left
53:27,91,50,48,57,122.      # R2            ESC [209z       del word right
54:27,27,91,50,48,57,122.   # ESC R2        ESC ESC [209z   del to eol
54:5.                       # Ctrl-E        Ctrl-E          del to eol
55:20.                      # Ctrl-T        Ctrl-T          top of fold
56:2.                       # Ctrl-B        Ctrl-B          bottom of fold
57:27,91,50,50,53,122.      # F2            ESC [225z       comment style
58:27,27,91,50,49,56,122.   # ESC KPAD 5    ESC ESC [215z   list files
59:27,91,50,51,52,122.      # F11           ESC [234z       pick
60:27,91,50,51,53,122.      # F12           ESC [235z       copy pick
61:27,91,50,51,51,122.      # F10            ESC [233z      put
62:9.                       # Tab           TAB             toggle tabs
63:11.                      # Ctrl-K        Ctrl-K          define key
63:27,32.                   # ESC SP        ESC SP          define key
64:27,27,91,68.             # ESC Left      ESC ESC [ D     previous file
65:27,27,91,67.             # ESC Right     ESC ESC [ C     next file
66:16.                      # Ctrl-P        Ctrl-P          to upper
67:14.                      # Ctrl-N        Ctrl-N          to lower
68:1.                       # Ctrl-A        Ctrl-A          browse
69:27,27,91,50,50,50,122.   # ESC KPAD 3    ESC ESC [222z   save
70:4.                       # Ctrl-D        Ctrl-D          define macro
71:22.                      # Ctrl-V        Ctrl-V          call macro
end of keyboard section for SUN4

f editor specific items
startupfile    "fold.stp"
helpkey "Press L2 for Help"
end f
```

C Product parts lists and installation instructions

The F editor exists in versions for various host computers and for a transputer board booted from a host by ISERVER. This appendix enumerates the documents and files included in these versions, and gives advice on installation. Where multiple files with the same name are shown, they will be installed in different directories.

All versions:

1 This manual

2 READ.ME – text file describing intallation process

3 F.BTL – transputer bootable file

4 FOLD.STP – startup file

5 FOLD.CMD – command history file

6 FTUTOR.OCC – tutorial file

7 ASCII.ITM – basic ITERM file for starting with ASCII keyboard

8 ASCII.HLP – keyboard help file for basic ASCII keyboard

PC version:

1 F.EXE – DOS driver calling transputer bootable file

2 F.EXE – DOS executable compiled with Microsoft C

3 PCFOLD.ITM – ITERM file for PC/AT

4 PCFOLD.HLP – Keyboard help file for PC/AT

5 BANSI.SYS – enhanced ANSI screen driver

6 NECFOLD.ITM – ITERM file for NEC PC

7 NECFOLD.HLP – Keyboard help file for NEC PC

8 NECF.BAT – NEC PC Batch file to call F on transputer board

9 NECF.BAT – NEC PC Batch file to call F on host

10 NECINI25.LIS – Special file for NEC

11 TDS3KEYS.LD – Special file for NEC

12 TDS3KEYS.TBL – Special file for NEC

13 DOSKEYS.LD – Special file for NEC

14 DOSKEYS.TBL – Special file for NEC

SUN versions:

1 f – UNIX shell command file to load and run f.btl

2 f – UNIX executable file compiled for SUN3 or SUN4

3 sunfold.itm – SUN keyboard version for SUNOS/SUNVIEW

4 sunfold.hlp – Corresponding keyboard help file

Installation instructions for various versions are included in the READ.ME files and in the delivery manual of accompanying INMOS toolset products.

It will often be necessary after installation to adapt any path names included in batch files or in the ITERM or startup files for the desired method of use. If the editor is first called in the directory in which the ITERM files are installed, all these changes may be accomplished using the F editor itself.

On the SUN3 it may be necessary to choose new key assignments for those F functions which are mapped by default onto keys which do not physically exist on the SUN3 keyboard.

Index