6 September 1991

Version 5.3 of ALEPH Note 90-96
Modified version of CLAS-NOTE 90-009
Modified version of DØ Note # 728

EDFOR Version 5.3

User Guide and Reference Manual

"Un bon ouvrier a de bons outils..."

Olivier Callot

Laboratoire de l'Accélérateur Linéaire

91405 ORSAY-CEDEX (France)

With contributions from A.Belk (CERN), Ph.Charpentier (CERN), A.Engelhardt (Copenhagen), J.Featherly (BNL), E.Harvey (LBL), W.Love (BNL), A.Mortsell (Uppsala), F.E.Paige (BNL) C.Steward (FNAL) and S.D.Protopopescu (BNL)

This is the reference manual for EDFOR, the text editor for Vax/VMS used in Aleph, Delphi, L3, and DØ (under the name EVEDT) and other places. This editor is written in VAXTPU, a VAX language for text processing, with elements from EVE (standard DEC editor), EVEPlus (DECUS improvements to EVE), and from home-written procedures. You can use this editor with any terminal supported by DEC, i.e. VT100, VT200, VT300, and VAXstations.

This document is organised as follows : News (i.e. changes from last release), General description of the editor, list and description of commands grouped by functions, keypad descriptions, style-dependent information and user parameters.

Part of this work was done with the support of the State University of New-York at Stony-Brook (USA), and part with the support of CERN, Geneva (Switzerland).

NEWS

1) EDFOR 5.3, September 6, 1991

This is a maintenance release, to fix several bugs reported in the recent months:

1. General

- The user definition file can be defined by the logical name `MY_EDITOR_TPU`. The default when this name is not defined is `SYS$LOGIN:MY_EDITOR.TPU`.
- The initialization file is now working. See HELP EVE INITIALIZATION inside EDFOR for more details on how to use this feature
- The message 'Journaling started for buffer xxx' has been removed.
- Gold-Y saves the PASTE buffer without an extra blank line at the end.
- New command `SET DIRECTORY XXX` to change the default directory.
- The word separators can now be defined independently for each style.
- The KP7 command (find next form-feed) now display the found form feed on the top line of the scrolling region on the screen, so that the new page is visible. Notice also that the form-feed character is not remembered as next string to search.
- The HELP command now works like the VMS help, with navigation up and down in the subtopic tree. Next_Screen and Previous_Screen allow to navigate up/down in the help buffer. A subtopic can be specified at any time, Return when no more to show goes up one level in the tree.

2. Spelling

- The SPELL BUFFER command accepts qualifiers for the spawned Speller.
- SPELL works only for certain styles. This means that if you go from a buffer with spelling enabled to a FORTRAN buffer, the spelling of your FORTRAN statements will not be checked any more.

3. Fortran

- Faster error positioning after compilation, and it now process correctly also files with ASCII TAB.
- The entry point list file is now defined by the logical name `ENTRY_POINT_LIST`. It is easy to create or update such a list with the command file `EDFOR$DIR:BUILD_ENTRY_POINT_LIST.COM`.
- FORTRAN header generation can be suppressed if you put `edf$default_header := 0;` in your `MY_EDITOR_TPU` file.
- ALIGN process correctly inline comments (the '!' is aligned for consecutive comments) and continuation lines : they have to be indented with respect to the first line, and all by the same amount.
- Handling of Fortran inline comment has been improved : They are no more generated by a TAB at End of Line, but by Enter-! in Vax_Fortran style. The '!' is aligned to the previous inline comment.
- New Enter command : Enter plus '.' generates an IF(first) THEN / first = .FALSE. / ENDIF structure at the current position, with the correct declaration of first with the other declaration statements

4. Other styles

- When Replying or Forwarding in MAIL, you are asked if you want to include the original message in the buffer, default is NO.
- ASCII TAB are generated and supported in DCL, if you set the variable 'edf$DCL_ASCII_TAB := true;'.
- A new style for Vax Macro (extension .MAR) and for RCP files (extension .RCP) in DØ.

2) EDFOR 5.2, January 7, 1991

This is a maintenance release, to fix several bugs reported in the recent months: The check at column 72 for fortran files was not performed in the commands **RELEASE_BUFFERS** and when using Gold-S. The **DRAW_LINE** was not setting the Overstrike mode, creating some unexpected effect. Spelling an empty buffer created an endless loop after typing two spaces. Some commands inserted in the DOCUMENT style header where invalid. Two changes : When using Captive EDFOR, you are not prompted for a new file when attaching back to EDFOR. The Help command now allows fast access to VMS utilities (routines names xxx$yyy, e.g. LIB$xxx or SYS$xx) by just typing **HELP xxx$yyy**.

Minor changes in various styles. When used with REPLY/EXTRACT in MAIL, the original message is displayed in the bottom window while the text of the reply is edited in the top window.

One new commands is implemented, **INDENT number** to indent a selected range or the whole buffer by the givent amount, negative to de-indent, processing only the fortran source lines.

In order to be compatible with the VMS 5.4 version of EVE, the Rectangular commands are renamed to BOX, and new functionality is added, like the **BOX COPY** and the possibility to work in insert mode. The toggle between normal and 'BOX' mode is still done by **set [no]rectangular**.

Two new facilities are provided for style design: The FORTRAN style is renamed to FORTRAN_77. This allows to force files to be FORTRAN_77 or VAX_FORTRAN, and to let the system decide (depending on file name and of **edf$default_vax_fortran**) for FORTRAN default style. The TAB key can also just insert an ASCII TAB, without indentation and so on, by defining **edf$ascii_tab := true;** in the question 2 of the style procedure (see chapter EXPANDING EDFOR).

3) EDFOR 5.1, October 5, 1990

This is a maintenance release, with a few bug fixes: Tabulation in C language, line split in DCL after a **$!**, TAB in Fortran on a line where the first 6 characters where deleted. The Help was corrected for inversion of lines in the PF2 display, information was added for SPELL and corrected for the wildcard search. The SPELLER now ignores words containing a '.' or a '$', and words starting with an accentued capital, or with a '<' as used by Vax Document. The content of the dictionary has been updated with a few missing technical words, and can be increased again on request. The '<' character is also removed from the list of word separators.

Two new styles are available, for Vax Document (file type .SDML) and for LATeX (file type .LATEX). An optional procedure allows to automatically include a logo in the Vax Document header page. To avoid ambiguous style names, the old 'Document' style is now named 'Text' style, and the style for Vax Document is named 'Document'. The files NEW_RELEASE.NOTES are now correctly processed with style RELEASE NOTE, the file types .SFO (Fortran) and .SC (C language) are accepted for SQL users.

The **SHOW BUFFER** can now write the selected buffer, using the Insert Here key. A new variable **edf$scroll_margin** is used to define the fraction of the screen where the cursor can not go because the screen will scroll. Another new variable **edf$c_tab_size** allows to control the display of ASCII tabs in the C style.

4) EDFOR 5.0, June 30, 1990

1. Major restructuration

This version has been restructured to allow an easy expansion for new styles and new experiments. See the corresponding chapter for instructions on how to expand EDFOR. This restructuration implies that the section file contains only one experiment flavour (the one selected by the user who created the section file), and only the styles NEUTRAL, FORTRAN and VAX_FORTRAN. If you use another style, the corresponding file in the `EDFOR$SRC` area will be loaded and used. The same apply if using a different experiment. The net result is a smaller file, and a somewhat faster start-up. But as loading a new style uses some CPU, it may be faster to compile the most commonly used styles. You just need to create a file `EDFOR$SRC:COMPILED_STYLES.LIST` with one style name per line. These styles are then compiled when building EDFOR. If you change this file, just type at DCL level:

```
@EDFOR$DIR:BUILD_EDFOR
```

2. Bug fixes

This version fixes also some small problems: the `WHAT LINE` was not giving the correct column; In the C style, with ASCII Tabs, the tab stops where not set correctly; In the `BUFFER` command, the ambiguous case was not handled correctly; A buglet in the `CLEAN` command for VAX_FORTRAN style is fixed. A bad behaviour of `Fill Paragraph` in a shifted window has been fixed. Grave and acute accents where inverted in SGML style.

This version fixes also 2 problems specific to VMS 5.3 systems, namely the ambiguous file name on the command line, and the /RECOVER option. Notice that VMS 5.3 systems use the 'buffer journaling' mechanism, and that individual buffer are recovered using the command `RECOVER BUFFER xxx` or `RECOVER BUFFER ALL`, see `HELP EVE RECOVER`.

3. New or improved commands

The command ALL allows 2 new choice: 'Find' the next line in the list of selected lines with the wanted string, and 'Select' all the line in the list which contain the wanted string. You can compile your buffer in the 'C' style using Enter-V. The indentation in the 'C' style was redesigned. When quiting, the list of buffers is displayed before prompting for quiting, if some modified buffer exist.

4. New user variables

Two new user variables are provided to control the action of the command `CLEAN` on VAX_FORTRAN source file: `edf$vax_keyword_case` describes the case of fortran keywords, can be "UPPER", "LOWER", "CAPITAL" for uppercase, lowercase, and capitalized words, and `edf$vax_variable_case` describes the case of the rest of the source, with possible values of "UPPER" and "LOWER". The default values are respectively "UPPER" and "LOWER".

Two new user variables are provided to control the default style: `standard_extension{".XXX"} := "style";` defines the requested style for all files with extension .XXX (with the dot and in uppercase); The possible values of 'style' are the various style names, 'NEUTRAL', 'FORTRAN', 'VAX_FORTRAN' or any 'ZZZ' with a corresponding file `EDFOR$SRC:STYLE_ZZZ.TPU`. And the variable `standard_file{"XXXX.YYY"} := "style";` defines the requested style for the given file name.

If you have problems, send MAIL to VXCERN::CALLOT with all relevant information. You may also send suggestions, remarks, complaints, new procedures, and so on. Any feed back will be gratefully appreciated.

GENERAL DESCRIPTION

This editor is a FULL SCREEN editor. This means text you type is inserted directly at the current cursor position. You can enter "line command mode" using a special key. But this is only a ONE line command, and you are back to SCREEN editing after completion of the command. This is somewhat different from EDT, where you can stay in line mode.

This editor can work with many BUFFERS (a text you are editing. In general, it is the content of a file. You can then edit as many files as you want), and with many WINDOWS (a portion of the screen where part of a buffer is visible). Standard lay-out is a big text window (all the screen minus two lines) plus a one line window for commands, plus a one line window for messages. You can easily have two or more text windows instead of one, showing the same or different buffers. These (one or two) text window(s) have a STATUS LINE, showing the buffer name, the editing style, the mode (INSERT / OVERSTRIKE) and the direction (FORWARD / REVERSE).

This editor can work with various STYLES, depending on the kind of file you are editing. Current styles are FORTRAN, VAX_FORTRAN, Historian, PATCHY, PASCAL, C, ASM, RTF, TPU, Vax Macro, DCL, MMS, RUNOFF, TeX, LATeX, Text, SCRIPT, SGML, Document, MAIL, Compack, ZEB, RCP, Structure Charts and Release Notes. There is also a neutral style, without any special flavor. The current style is displayed on the status line. More details on each style will be given in the corresponding section.

Another feature is the RECTANGULAR mode. This is related to the way you can define a part of a file: In normal mode, you can define a block of text which is all characters between two characters in the text. In RECTANGULAR mode, this block is a rectangle whose opposite corners are defined by two points. This is used for Select, Remove, Insert, Copy Text commands, see later. Rectangular mode is displayed as '□' on the status line.

Another choice is the experiment flavor (see user variables), you can see its value in the startup message, or with the `VERSION` command. This defines the Fortran / VAX Fortran headers and extensions. We now support ALEPH, AMY, D0, DELPHI, L3, LBL, OPAL and a "no" experiment, giving a somewhat neutral Fortran flavor. See the chapter 'Expanding EDFOR' to know how to create a new experiment flavor.

1) Usage

1. Standard EDFOR

This editor is called from DCL level by a statement like `$ EDF file_name` or `$ EVE file_name` (DØ) You can add any command qualifier allowed for `EDIT/TPU`, e.g. `/RECOVER` to restore an aborted session.

The first time you use this editor on a VAX/VMS system, you are prompted for your name, and your experiment. Your answer are written in the file `SYS$LOGIN:MY_EDITOR.TPU`, or in the file pointed at by the logical `MY_EDITOR_TPU` if defined, and this file is read each time you call this editor. This allows the system to insert your name when needed, mainly as a comment when creating procedures, subroutines, etc. You can of course edit this file to change your name. You can also add to this file some private VAXTPU procedures in front of what exists. You can also add some commands to select various options in EDFOR, see the last chapter in this document. New information is also added to this file when you use the MAIL style for the first time. The use of the logical name `MY_EDITOR_TPU` allows you to switch settings if you are working on two different experiments.

2. In VMS utilities: MAIL, DEBUG, ...

In order to use EDFOR in various VMS utilities, you have to look at the corresponding Help file or manual. In general, you will need that the logical name `TPU$SECTION` points to the editor section file `EDFOR$DIR:EDFOR.TPU$SECTION`. Then, you have to do something:

`MAIL` You have to define:

`$ DEFINE MAIL$EDIT CALLABLE_TPU` or `SET EDITOR TPU` inside MAIL.

`$ MAIL := MAIL/EDIT=(SEND,REPLY)`

Then, each time you send or reply to Mail, you will be in EDFOR, with a special style (see the corresponding chapter). If you EXIT, the file will be sent, if you QUIT, no mail will be sent. All editor commands to include files, to spawn, ... are available. The Speller works only if the EDFOR related logical names are system defined, due to the way callable images (the speller) are activated from privileged images (MAIL). If you encounter this problem, you can circumvent it by defining `MAIL$EDIT` as a command file which executes EDFOR in a subprocess, see `SYS$SYSTEM:MAILEDIT.COM`.

`AMSEND` (`ALEPH`) You have to define the symbol `ALEPHMAIL$EDITOR :== 'EDF`, and that's it.

`DEBUG` You have to have an initialisation file, and assign the logical name `DBG$INIT` to this file. In this initialisation file, put the following line :

```
SET EDITOR/CALLABLE_TPU
```

Then, the Debugger command `EDIT` will call EDFOR for the source of the current module, if available.

`Any FORTRAN program` You can edit a file with a statement like:

```
STATUS = TPU$EDIT( input_file, output_file )
```

where the two arguments are character variables. On return, you can even check if you EXITed or QUIT by the value of STATUS:

```
IF( IAND( STATUS, 'FFFFFFF'X) .EQ. %LOC(TPU$_EXITING) ) THEN ! exit
ELSE ! quit
ENDIF
```

No special link command has to be added, `TPU$_EXITING` has only to be declared EXTERNAL.

3. Captive EDFOR

You can use EDFOR in a 'captive subprocess' to shorten the start-up time. You have to create a file `SYS$LOGIN:RUN_SUB.TPU` which contains the following statement: `edf$run_in_subprocess := 1;`. Then, execute the following command (you can define a symbol in your LOGIN file to be this command):

```
SPAWN/PROC=EDFOR_your_name/INPUT=SYS$COMMAND EDIT/TPU/COMMAND=SYS$LOGIN:RUN_SUB
```

Then, you are in EDFOR. The only difference is that when you QUIT or EXIT you are attached back to the main process. The files are written before EXIT, and you are prompted (if any modified buffers exist) before QUIT. Modified buffer are DELETED when quiting if not written before... To go back to the editor, use `ATTACH EDFOR_your_name` and it starts very fast. You can even attach this command to a key on the keypad, like `DEFINE/KEY/TERMINATE/NOLOG PF1 "ATTACH EDFOR_your_name"` and just hit PF1 to call the editor.

Of course, there is a price to pay for that speed increase :

- o The subprocess knows the current default directory at the time he was spawned, and not the current one. This can be changed by the command `SET DIRECTORY xxx`.
- o The subprocess knows the symbols and logicals defined at the time he was spawned, and not any you created afterwards.
- o If something crashes, recovery will be very difficult : you have to restore to their previous state all files modified by this editing session, and play again the full thing, including any commands performed outside editing (but EDFOR doesn't know these commands...)
- o It doesn't work well if you are in a subprocess and spawn this editor: you will exit and go to the parent process, the one at the top of the process tree...

That's why this way of working has to be reserved for specific needs. It's also a way to load your computer, by adding processes which use memory even when they are not in use.

2) Documentation

This reference manual is updated for every new release. You can obtain your own copy by processing through TeX the file `EDFOR$SRC:EDFOR_REFERENCE.TEX` (some change in the first few lines may be needed, to select fonts existing on your local printer):

The internal help file, `EDFOR$DIR:EDFOR.HLB` can be looked at and printed (the source of the help file is `EDFOR$SRC:EDFOR.HLP`).

3) Installation

EDFOR is installed on every DØ Vax by the standard distribution mechanism of the D0LIBRARY. To install it on any other VAX, you need to

- o Define the logical name `EDFOR$DIR` for the binary files.
- o Define the logical name `EDFOR$SRC` for the source files (they can point to the same directory)
- o Copy the command file `VXCERN::DISK$ALEPH:[CALLOT.EDFOR]EDFOR.IMPORT` to this `EDFOR$SRC` directory on your local machine.
- o Run this command file.

The command file copies the sources and processes them to create the binary file and the help file. You need to process the .TEX file yourself, since TEXimplementation is site dependent. The command file also sends a mail message to VXCERN::CALLOT to keep track of the nodes where this editor is installed, for distribution of information on bugs and new releases.

In order to define the logical names and the symbol, you have only to run the command file `@EDFOR$DIR:SETUP_EDFOR` if it is not already done in your system or group login. See inside this file for the value of the logical names.

4) Terminal setting

Your terminal has to be an ASCII terminal. And the terminal type (VT100, VT200, VT300) has to be correctly known by VMS (DCL command `SHOW TERMINAL`) in order to have the best results. A VT100 treated as a VT200 by VMS will see strange characters...

When working with a VAXstation, you will see that scrolling is relatively slow, slower than re-drawing the screen. This is very clear when using large windows, i.e. with 40 or 50 lines. You can use the DCL command `SET TERMINAL/NODEC_CRT` to instruct the screen manager to redraw the whole screen instead of scrolling. This is faster by an order of magnitude. However, this is not a perfect solution, as the screen will be completely redrawn even if you have to scroll for one line, and as it is erased and redrawn, it's not comfortable.

LIST OF COMMANDS

We will describe all the commands. Commands can be either keypad commands (you have to type one or two keys to execute the command) , or line commands (you have to type the 'DO' key (either PF1 followed by KP7 or the 'Do' key on the VT200 keypad) and then the name of the command, which may be abbreviated as long as it is unambiguous on a word by word basis) . In the following description, **line commands will be typed in UPPER CASE, and keypad commands in lower case** (if they are mainly used as keypad commands or are not available as line command) followed by the key name between brackets []. The key names are defined for the numeric keypad: KP0 to KP9, PF1 to PF4, 'minus', 'comma', 'dot' and 'enter'. On the VT200, there exist also pre-labelled keys, and the upper row is labelled from F6 to F20.

A new meaning can be associated to any key if you hit first PF1, which is called the 'GOLD' key. This appears in the following description like '`[Gold-KP4]`', this is : Hit PF1, then KP4. You have also key names like '`[Ctrl-K]`', this means : Hold down Ctrl key and press 'K' key. There are finally special keys, like TAB, Backspace, line-feed and ESC, which are labelled on your keypad.

If a command needs an argument (number, string), it is indicated as 'nn' or 'xxx' in the description. If you omit an argument (or for keypad commands), you will be prompted for it.

1) GENERAL COMMANDS

`EXIT [Ctrl-Z], [F10]` saves the current buffer if it has been modified, then asks for every modified buffer whether you want to save it, and then exits. If a buffer has Fortran style, a check is performed to veto the exit if any line is longer than 72 characters (except for comments). If this occurs, you are prompted and ask if you want to continue exiting or to abort. In the later case, you are positioned on the (first) faulty line.

`QUIT [Gold-Q], [F9]` doesn't save any buffer. If any buffer has been modified, the list of user buffer will be displayed, and you will be prompted for confirmation before losing all your work.

`[Ctrl-Y]` is fatal. You interrupt the editor. Two ways of recovering : Immediately after the interrupt, try the DCL command `$ CONTINUE` to restore your editing session. Or later (or after a system crash), call this editor with `$ EDF/RECOVER file_name` (up to VMS 5.2, use `EDF file_name` and then the command `RECOVER BUFFER file_name` with VMS 5.3 systems) with the same terminal. You will see a replay of a major part of your editing session, the last few actions are not always saved. See the DCL help on `EDIT/TPU` for more information and restrictions on the recovery procedure.

`[Ctrl-C], [F6]` cancels execution of a command. Mainly if VAXTPU is hanging in an endless loop. You can abort the loop. WARNING, the Ctrl-C is not 'journaled', i.e. recovery will not be possible later on. So after Ctrl-C, it's a good idea to Exit.

`[Ctrl-W]` repaints the screen completely. Useful in some rare circumstances where some trouble occurred (messages with bell or communication problems).

`[Ctrl-B]` recalls the previous line_command (when editing the text). It also works when waiting for a command (i.e. at the `Command:` prompt), but then you can use the up/down arrows. You have to use the arrows for `Find` , at the `Forward Find:` prompt, to retrieve the previous searched strings.

`REPEAT nn [Gold-(0123456789)]` allows you to repeat the next character, keypad or line command 'nn' times. The Gold-numeric is the fastest way to repeat a command: to move 5 characters right, type `[PF1], [5], [right arrow]` and that's all. You can also repeat learn sequences.

2) DISPLACEMENTS

This is one major feature of a full screen editor. There are a lot of possible ways to move the cursor, depending on what you want.

1. Geographical displacements

Cursor displacements can be performed by the **`LEFT, RIGHT, UP and DOWN arrows`** on the keypad. The effect is evident, with the following explanations: When in INSERT mode, the cursor is bound to text. You can not go beyond end-of-line, the next character to the right of the end of line is the first of the next line. When you move vertically, the cursor tries to keep the same offset on every line. But if a line is shorter, the cursor seems to move left. When moving to a long line, the cursor will be again on the old column. When in OVERSTRIKE mode, the cursor is just free on the screen. You can go beyond the last character of the line, and when moving vertically, the cursor offset is unchanged. In both cases, we keep the cursor away from the first and last lines, except when at beginning or end of the buffer. There is a guard zone where the cursor can not be: This is 15% of the screen size on the main window (3 lines on a 24 line screen), 5% of the total screen size for the smaller windows (1 line on 24 line screen).

2. Amount displacement

The cursor can also be moved by a 'logical' amount in the current direction: you can

`Move by word [KP1]` , this is go to the beginning of the next (or preceding if in REVERSE direction) word. Default words separators are defined by the user variable **`edf$word_separators`**, see the last chapter for default value. They can be redefined for each style, see the chapter on expanding EDFOR.

`Move to previous word [KP3]` moves to the beginning of the previous word (or next word if in REVERSE direction).

`Move by end-of-line [KP2]` , this is go to the right of the last character of the current line. If already at end of line, go to the next (previous if in REVERSE direction) end of line.

`Move by beginning-of-line [KP0]` , this is go to the first character of the line. If already at beginning of line, go to the next (previous if in REVERSE direction) beginning of line.

`Move by line [F12]` is move by end-of-line if current direction is Forward, and move by beginning-of-line if current direction is Reverse.

`Move by screen [KP8]` , this is go to the next (previous if in REVERSE direction) page of the buffer, the page size being the window size minus two lines. The cursor stays at the same screen position (if possible, this means if there is some text at this point, and if we don't reach the beginning or end of the buffer). The two next keys have the same behaviour :

`[Prev Screen]` Move by one screen (window size minus 2 lines) in the Reverse direction.

`[Next Screen]` Move by one screen (window size minus 2 lines) in the Forward direction.

3. Positioning

You can go to a specific position, like:

`Go to Start of line [Backspace],[CTRL-H]` , not the same as 'move-by-beginning-of-line', because here you will stay on the same line, with a warning if you are already at start of line.

`Go to End of line [CTRL-E]` is the equivalent of the previous command, but in the other direction. You will receive a warning if you are already at the end of the line.

`Top [Gold-KP5]` is go to beginning of the buffer. You will have a warning if you are already there...

`Bottom [Gold-KP4]` is go to end of buffer. You will have a warning if you are already there...

`LINE nn` allows you to go to line nn of the buffer. This line may not be the one indicated by some error messages (e.g. from FORTRAN) due to include files.

`ROUTINE xxx nn` for a Fortran file: position to the line number 'nn' (default 1, you will be prompted if needed) in the routine 'xxx' (default to the current one, you are prompted for) in the current buffer. The line number is the FORTRAN LINE NUMBER, i.e. the line number given by the compiler error report or by a traceback. This count the line in all the include files and not only in the present source file.

`GO TO xxx` positions you on a MARK defined earlier by the command `MARK xxx` .

`[Gold-J]` positions you on the position marked previously by `[Gold-X]` , in fact a `GO TO EDFOR_MARK` mark.

4. **Text search**

This is one very nice feature of this editor. You can define the text you want to find in different ways.

`FIND xxx [Gold-PF3] , [Find]` : You are prompted for a string, if it is not included in the line command, and then this string is sought in the current direction. Unlike EDT, the string is terminated by a return. If the string is not found in the current direction but in the other direction, you are prompted for accepting a change in the current direction. This search is CASE SENSITIVE if the string you typed contains any UPPER CASE letter, and case insensitive if you gave only lower case letters. You can force case sensitivity with the next command.

`SET CASE (NO)EXACT` Allows you to change from EVE standard choice for case sensitivity (NOEXACT, see description on previous command) to full case sensitivity (EXACT, useful for finding/replacing special characters with ASCII code over 128). This works for FIND and REPLACE commands.

`WILDCARD FIND xxx [Gold-Find]` allows you to search text with wildcards, i.e. define only part of what you want. You give a text to search for, but some characters have special meaning (notice that `WILDCARD FIND` is identical to `FIND` if you do not use wildcards). But the search becomes CASE SENSITIVE if you use one of the following wildcard (use the EDFOR command `SHOW WILDCARD` to list all possibilities) :

*	multi character wildcard on the current line
%	single character wildcard
\<	beginning of line
\>	end of line
**	Multi character, multi line wildcard
\	Quote next character

`SHOW WILDCARD` Shows the various wildcard characters and their function.

`Find Next [PF3]` searches for the next occurrence of a previously defined item (either `FIND` or `WILDCARD FIND`). You have the same mechanism for change of search direction as described for FIND.

`Page, [KP7]` is a search for an ASCII Form-feed. This character can be inserted by the command `[Ctrl-F]` , or may exist in your file, if it is the output of some processor. If found in the current direction, the line with this character is displayed on the topmost line of the scrolling area.

`Fortran page, [Gold-P]` is a search for a '1' in first column.

`Next input place [Gold-KP3]` is a search to the next two consecutive spaces on the current line. This is used to go to the next place where an input is expected after a Enter-xx extension. You can not repeat this search thru the Gold-Numeric mechanism.

`ALL str1 op str2` displays all lines where some (combination of) string(s) occurs. You can either give one string (between quotes if it contains spaces), or two strings with an operator: you can request the AND (op = '&'), i.e. both strings on the same line in any order, the OR (op = 'vertical bar'), i.e. either of the two strings selects the line, or the NOT (op = '-'), i.e. the first string but not the second one occurs in the line. If any match occurs, you will see a list of the matching lines, with the line number in front. You have to select one of the lines (with up/down arrows or next / previous screen, or with the Find key to select the next line containing the string you are prompted for), and type return. Then you will be positioned to this line in the original buffer.Or you can restrict the list of line using the 'Select' key and retain only the line containing the string you are prompted for.

`Match Bracket [Gold-)],[Gold-(]` : When the cursor is on a `([{` (resp. `)]}` , search for the corresponding closing (resp. opening) character at the same nesting level in the forward (resp. reverse) direction. Message if not on a bracket character.

3) ADDING TEXT

1. Typing characters

The standard characters are those on the main keyboard. When you are in INSERT mode, the character is inserted before the cursor, and the text to the right is shifted right by one column. In OVERSTRIKE mode, the character replaces the current character. A check is performed for every character to see if it is inside the margins. If not, an action is taken, as described below for the space bar. Some keys on the keyboard have special effects :

`[Space bar]` is used to add a space in the text. But if you are on too long a line (the current position is after the right margin or after column 72 for a Fortran source line), the line is split to keep the line length inside the margin. The split occurs after the last word included in the margins specifications. The new line can be indented (if left margin is not 1) and, depending on the style, some specific pattern can be added in front of the line. For example, a Fortran continuation is generated if the style is FORTRAN. The way this pattern is computed depends on the buffer's style, and is described for each style in the corresponding section.

`[Return]` is used to split the line at the current position. The text to the right of the cursor, if any, is put on a new line. The same mechanism as for space bar is used if you're over the right margin. The behaviour is sometimes confusing when you type a text, and you reach the margin. You are tempted to type a RETURN, to force a new line. But then the line will be split and the last word put on a new line, and the cursor on another new line. That may be one line more than you wanted. Conclusion, never use return when typing a continuous text. Use it only for paragraph breaks.

`[Gold-return]` You can force generation of the continuation even for shorter lines by typing gold-Return instead of Return. The continuation and indentation is generated with any line length.

`Open Line [Gold-KP0]` is used to create a new line. If used in column 1, this just creates an empty line and the cursor is on this empty line. If typed in a line, the part of the line from cursor to end of line is put on a new line, with indentation and continuation if needed, and the cursor is not moved. This is quite similar to `[Gold-return]`, but the cursor stays where it was.

`[Tab]` The behaviour of this key is very style dependent. This key generally inserts a certain number of spaces in the text. The actual number of spaces inserted is computed as follows: We compute from the current and previous line the position where the first valid character should be : This is column 4 in most styles, but is like previous Fortran line in Fortran (see later for more details). If the actual position is before this first character position, we add the number of spaces needed to go to this position. If after, we go to the next TAB, i.e. multiple of 8 columns, if not changed by `SET TAB EVERY nn`. This means that we go to the VMS standard TAB stop. A TAB in column 1 generates more complex patterns in DCL (a $ and 3 spaces), ZEB and some other styles. See exact description in the corresponding section.

`[Gold-tab]` inserts an ASCII tab character. This editor doesn't like the ASCII TAB, because he can not handle the indentations with TABs. So, in general, don't use this key, and use commands `CLEAN` or `ELIMINATE TABS` to replace TABs by the correct amount of spaces.

`Quote [Ctrl-V]` is used to introduce special characters in the text, characters you can not type on the keyboard. This is mainly for control characters not trapped by VMS (this excludes Ctrl-Y, Ctrl-O, Ctrl-Q and Ctrl-S, Ctrl-C). After typing Ctrl-V, you type the character you want to insert, like a Return or ESC or Line-feed.

`[Ctrl-F]` Adds a Form-Feed at the current position. This is a short form for `[Ctrl-V]+[Ctrl-L]`.

`Special insert [Gold-KP3]` If you need to add one of the previously listed strange characters, then use the following mechanism: You have to know the decimal value of the ASCII character you want to insert. Then type the following sequence: Gold, the ASCII value with the numeric keys of the main keypad, then Gold, then KP3.

2. **Saved sentences**

`Restore Line` `[Gold-PF4]` is used to introduce before the cursor position the most recently erased text (through erase-line, erase-start-of-line and erase-to-end-of-line). You can insert as many times as you want. This is extremely useful for duplicating one line of text.

`Restore word` `[Gold-minus]` is used to introduce before the cursor position the most recently erased word (through erase-word or erase-previous-word).

`Restore Character` `[Gold-comma]` will be used to restore the most recently erased character.

`Insert Here` `[Gold-KP6]`, `[Insert Here]` copies the content of the Insert Here buffer before the cursor. This Insert Here buffer is filled by Remove and Append commands, and is used mainly for large pieces of text.

3. **External inputs**

`Style Extensions` `[Enter]` This is a hook to define frequently used pieces of text which are style dependent, like the SUBROUTINE header in Fortran. You have to type Enter followed by a letter, the result is style dependent. For each style, you can type `[Enter]` followed by `[PF2]` to obtain a list of valid extensions.

`DRAW BOX` `[Gold-|]` draws a box whose corners are the Selected point and the current cursor. Works only in Overstrike mode.

`DRAW LINE` `[Gold- -]` draws a line using the 4 arrows to give the direction of the motion, until you type any other character. Works only in Overstrike mode.

`INCLUDE FILE xxx` `[Gold-I]` is used to copy a disk file before the current line. You will be prompted for the file name. Wildcards are allowed, but if there is more than one file, the list will be displayed on the screen, and you will be asked to select one of these files.

4) ERASING THINGS

You can erase (i.e. remove from the text) various elements:

1. **Character, word, line**

`Delete` `[<` X`|]` `[delete]` erases the character to the left of the cursor. If the cursor is at beginning of a line, the line separator is removed, and the line is appended to the end of the previous one.

`Erase Character` `[comma]` erases the character where the cursor is. If the cursor is at end of a line, the next line is appended to the current one. In these two commands, the remainder of the line is shifted left one position after character removal.

`Erase Word` `[minus]`, `[F13]`, erases the word where the cursor is, a word being delimited by any of the character given in `edf$word_separators`. This erases also the space(s) after the word, and the cursor is put at beginning of next word. If the cursor is on a string of many spaces, this is reduced to one space. If the cursor is on the only space between words, the next word is erased.

`Erase Previous Word` `[Ctrl-J]`, `[line-feed]` is identical to the previous one, except for the behaviour when the cursor is between words, where the previous word is erased (no processing of a string of spaces).

`Erase Line` `[PF4]`, this erases the whole current line, including line terminator. The cursor is left at the beginning of the next line.

`Erase Start Of Line` `[Ctrl-U]`,`[Ctrl-X]`,`[Gold-Delete]`, this erases from start of line to the cursor position.

`Erase To End Of Line` `[Gold-KP2]`, erases from the cursor position to end of line, but not the line terminator.

You can restore the erased text using the (previously described) commands `Restore Line`, `Restore Word`, `Restore Character` depending on the object you just erased.

2. Blocks of text

You can also delete a bigger piece of text, by a mechanism we will use many times: The Select mechanism.

`Select [dot], [Gold-dot], [Select]` puts a marker at the current cursor position (you can cancel this marker by typing Select another time). You then move the cursor to the other end of the text you want to act on, and then type

`Remove [KP6],[Remove].` The 'selected' text is removed from the current buffer (and saved in the INSERT_HERE buffer).

`Store Text [Gold-Remove]` is used to copy the selected range to the INSERT_HERE buffer, but without modifying the current buffer.

`Append [KP9]` has the same effect as `Remove` on the current buffer, but the removed text is put at the end of the 'insert here' buffer. This can be used for example to pick up all FORMAT statements of a subroutine, and put them at the end of the routine.

3. Handling boxes: the rectangular mode.

When you are in RECTANGULAR mode, the part of text you select is a rectangle (and not all the text between the two points). The exact behaviour depends on the settings of the buffer, i.e. if the mode is INSERT or OVERSTRIKE. When setting the rectangular mode by `set rectangular,` the buffer is also set to overstrike. The following commands are then replacing the standard ones: `BOX SELECT` replaces `Select,` `BOX CUT` replaces `Remove,` `BOX PASTE` replaces `Insert Here` and `BOX COPY` replaces `Store Text.` The key binding is valid only after a `SET RECTANGULAR,` and is displayed on the statue line with a '[]'.

`BOX COPY [Gold-Remove]` copy the selected box to the INSERT HERE buffer. The current buffer is not modified.

`BOX CUT [Remove], [KP6]` move the selected box to the INSERT HERE buffer. In overstrike mode, the space is filled by spaces.

`BOX CUT INSERT` is identical to `BOX CUT,` but forces the 'insert' behaviour.

`BOX CUT OVERSTRIKE` is identical to `BOX CUT,` but forces the 'overstrike' behaviour, i.e. the removed text is replaced by spaces.

`BOX PASTE [Insert Here], [Gold-KP6]` copy the content of the INSERT HERE buffer as a box whose upper left corner is the cursor position. The copied text overstrikes the existing text in overstrike mode, or is inserted in insert mode.

`BOX PASTE INSERT` is identical to `BOX PASTE` but forces the 'insert' behaviour.

`BOX PASTE OVERSTRIKE` is identical to `BOX PASTE` but forces the 'overstrike' behaviour.

`BOX SELECT` is identical to `Select,` but the selected range is shown in bold instead of reverse video. Notice that the full text range is highlighted, not only the box, due to VAXTPU limitations.

5) MODIFICATION

One way to modify the text is to delete and/or insert characters, but there are some powerful commands:

1. Replace

`REPLACE xxx xxx [Gold-Enter],[Gold-KP9]` will prompt for a string to be sought (Like the Find command, case sensitive if some uppercase letters are in the string, or if you have turned on the `SET CASE EXACT` option), and for a string to replace the original one. A search is performed in the current direction (and in the reverse direction if needed, see Find), you are positioned on the string occurrence and asked for action:

Y means Replace and search for the next occurrence. The replacement is also case sensitive if needed, i.e. the inserted string has the same case as the removed one (upper, lower, capitalized). This is the default action, i.e. the one performed if you type Return.

N means don't change and search for the next.

L means replace and stop here.

Q means don't replace and stop here.

A means replace all occurrences without prompt (be careful!). You will be prompted only for a change in search direction.

2. Change case

`Change case [Gold-KP1]` changes the case of the current character, if this is a letter, and advances one character. If SELECT is active, the case of all letters in the selected range is changed.

`UPPERCASE WORD [Gold-U],[F20]` changes the case of every letter in the current word (or of the whole selected range if any) to UPPER, and the cursor is left at beginning of the next word.

`LOWERCASE WORD [Gold-L],[F19]` changes the case of every letter in the current word (or of the whole selected range if any) to lower, and the cursor is left at beginning of the next word.

`CAPITALIZE WORD [F18]` Change the case of the current word (or of all the words in the selected range if any) such that the first letter is UPPER, and the rest lower. The cursor is left at beginning of the next word. This doen't work well in a selected range if the beginnig of the range is at the end of a line.

3. Cleaning

`Center Line [Gold-C]` centers the current line between right and left margins.

`FILL PARAGRAPH [Gold-F]` aligns the text between the right and left margins for the current 'paragraph', defined as the part of text containing the cursor and limited by empty lines. Or as the selected range if one is active at this time. This is extremely dangerous in non-text files, like Fortran source... So you are asked for confirmation on any file other than MAIL and Text styles. A string of spaces is replaced by only one space, then the paragraph is filled within the buffer margins unless the paragraph starts with a non alphabetic character, in which case the left margin is defined by the position of the first alphabetic character. Then we add extra spaces between words to right justify the text, and finally position the cursor at the beginning of the next paragraph.

`Fill Area [Gold-KP8]` works only in the selected area, from the last SELECT character to the current one. (This is another example of the use of the select mechanism). But Fill Area doesn't right justify nor indent the text.

`FIX CRLF` is used to suppress the ASCII CR and LF characters, included in some output as line end marker (e.g. RUNOFF output). This suppresses CR + LF if at end of line, and replaces them by a line split if not at end of line.

`FILTER` replaces spurious 8 bit ASCII by 7 bit (after confirmation) and non printable characters by some string you give after a prompt.

`ELIMINATE TABS` replaces the TAB characters in the current direction by the number of spaces needed to have the same display if tabs were every 8 columns.

`TRIM BUFFER` is used to suppress all trailing spaces, i.e. those at end of line. This can reduce the size of a file.

`CUT nnn` is used to cut all columns over the given one (default is 73, for Fortran). It is followed by an internal call to `TRIM BUFFER`.

`INDENT number` is used to insert 'number' spaces, or delete '-number' spaces if 'number' is negative, at the beginning of every line in the selected range or in the whole buffer. With a negative number, only spaces are deleted: if the line does not contain the requested number of spaces, only the existing spaces are deleted. For FORTRAN files, the operation is performed only on the columns over 7, and the comment statements are not touched.

`ALIGN` is used (on Fortran files only) to indent the DO loops and IF blocks. It works either in the entire buffer, or in the selected range if any is active at this time. Each FORTRAN statement is align on the previous statement with indentation in DO loops and IF blocks, comments are untouched, continuation of inline comments are align, and continuation lines are indented at least as the previous statement plus one step, and all consecutive continuation are aligned.

`CLEAN` is the most powerful tool to clean a buffer, mainly a Fortran buffer. It replaces tabs by spaces, deletes trailing spaces, and, for FORTRAN files, right adjusts the statement labels, indents the DO loops and IF THEN/ELSE/ENDIF clauses through the ALIGN command, changes all statements (excluding comments and strings) to UPPER CASE in FORTRAN77, or independently the keywords and variable if VAX_Fortran, according to the values of `edf$vax_keyword_case` and `edf$vax_variable_case`, and then checks for lines longer than 72 characters.

6) MODE CONTROL

This section describe how you can change some modes of this editor.

`INSERT MODE` sets the mode to insert for the current buffer.

`OVERSTRIKE MODE` sets the mode to overstrike for the current buffer.

`Change mode [Ctrl/A], [Gold-A], [F14]` is used to change from INSERT to OVERSTRIKE, or vice-versa.

`Forward [KP4]` sets the direction to forward for the current buffer.

`Reverse [KP5]` sets the direction to reverse for the current buffer.

`Change Direction [F11]` changes the direction.

`SET RECTANGULAR [Gold-[ ]` Sets the mode to rectangular. This changes the behaviour of the Select, Remove, Insert Here and Copy Text commands: the text is selected as a rectangle with Select point and current point as two opposite corners (and not as a contiguous string in between), the removed text is replaced by spaces, and, when inserting, the previous text is overwritten. Sets also the OVERSTRIKE mode.

`SET NORECTANGULAR [Gold-] ]` return to the standard way. Sets also the INSERT mode.

`STYLE xxx` selects the editing style, you give the name, or an unambiguous abbreviation thereof, and your style will change. See corresponding sections for details. If your buffer is empty, the empty_buffer action will be performed.

`Set Margins [Gold-M]` asks for right and left margins and sets them in the current buffer.

`SET LEFT MARGIN` sets the left margin to a new value.

`SET RIGHT MARGIN` sets the right margin to a new value.

`SET STEP` sets the tab_step for this buffer, i.e. the value used when indenting a line. See the default value for each style, and see also how to change this default for Fortran files in the last chapter.

`SET TABS EVERY n` sets the tab position every 'n' columns.

`MOUSE ON/OFF` selects if the mouse keys can be used to edit. The function of the mouse keys is described when you use them, as it changes when you use them.

7) BUFFERS, FILES AND WINDOWS

1. Buffers

A buffer is the name given to a text, and is generally the content of a file. VAXTPU uses system buffers (like the message buffer and the command buffer) and user buffers, whose contents are the edited files. The buffer related commands are:

`BUFFER xxx [Gold-B]` is used to go to a buffer. You are prompted for the name, and prompted again to solve ambiguous names. If the name doesn't correspond to any existing buffers, a new buffer is created. This buffer is not related to a file, has the neutral style and the default settings.

`SHOW BUFFERS [Gold-?]` lists the sorted list of user buffers, with associated length and status. You can then position (using the up or down arrows) to one name in this list, and either go to this buffer with [Select] or [period], write this buffer with [Insert Here], or destroy this buffer with [Remove] or [KP6]. You can also use any other command to go to another buffer.

`SHOW SYSTEM BUFFERS` does the same job for system buffers.

`Go To Default Buffer [Gold-D]` puts you again in the default buffer, i.e. the buffer you edited first.

`Return To Last Buffer [Gold-R], [F17]` put you in the previous user buffer, i.e. the one before the last buffer change.

`RELEASE BUFFERS` writes out all writable and modified buffers, to their standard output file. No questions...

`DELETE BUFFER` is used to destroy the buffer and buffer content. (It is bound to [Remove] and [Gold-KP6] after a `SHOW BUFFERS`). This is useful if you want to reread the same file you are editing, to restore some error. This can not be done if the buffer with that file name has not been previously destroyed, because EDFOR will detect that you are already editing the same file. May be useful also when you have made an error in the directory of a file: this creates an empty buffer with the wanted file name. If you now read the correct file, you will need to provide a new buffer name. Destroying the buffer beforehand avoids this problem.

2. Files

When calling this editor, you can give a file name, which is the input file name of the first buffer, and that buffer's output file name. When editing, you can start working on another file.

`GET FILE xxx [Gold-G]` creates a new buffer with file name as name (you will be prompted for a new name if this name is already used) and reads the file. Wildcards are allowed in the file specification, and you will have to choose if ambiguous. You can not read a file you are already editing. The style is set according to file_type. This is useful if you want to create a new buffer with a certain style: issue the get file command on a non existent file instead of the buffer command.

`READ xxx` is a GET FILE in read-only mode, i.e. you will not be allowed to modify the buffer.

`Get source containing a standard entry [Gold-E]` prompts for the entry name, or reads it from the current line if this is a FORTRAN line with a CALL . Then, if needed, we load the local software index file `D0$DOCS:ENTRY_POINT.LIS` for DØ, `EDFOR$DIR:ENTRY_POINT.LIS` or the logical name `ENTRY_POINT_LIST` elsewhere, and search this file for the requested entry (full name only, no wildcard). If only one source file contains this entry, we load this file and point to the entry point. If there is more than one file, we display the list of such files, and wait for a choice. We then proceed as before.

`ZEB FILE xxx` prompts for the name of the .ZEB file to be read from the `D0$ZEB` area. It then loads this file and puts it in a new window on the screen.

`Show Include File [Gold-N]` loads the file described on the current line by the Fortran INCLUDE statement, and put it on the screen in a new window. Protected if the file doesn't exist.

`INCLUDE FILE xxx [Gold-I]` reads the file and puts it in the current buffer before the current line.

`Save file [Gold-S]` writes the current buffer to a file, you are prompted for the file name. The buffer default output file is changed to the given name if you accept this choice at the prompt.

`WRITE FILE` writes the current buffer to his currently defined output file (in general the same as the input file). You are prompted for a file name only if the output file is not defined. But you can always give a file name, and this names becomes the new output file name if you accept this choice at the prompt.

`Write INSERT HERE buffer [Gold-Y]` prompts for a file name, and then writes on this file the contents of the INSERT HERE buffer, also called the PASTE buffer.

`Set Directory xxxxx` changes the VMS default directory for reading or writing files. The previous default is NOT restored when exiting from EDFOR. The change is propagated to the DCL subprocess if one is active.

Other file manipulation commands can be performed via DCL or SPAWN commands, see later.

3. Windows

ONE WINDOW sets the standard screen, i.e. one big window displaying the current buffer. Warning if only one window on the screen.

TWO WINDOWS splits the screen in two windows, each one with the current buffer.

Change Number Of Windows [Gold-T] is used to change from one to two windows and vice-versa.

SPLIT WINDOW [F7] splits the current window in two windows, with the same buffer content. You can split a window as long as the new windows will be at least 2 lines high.

DELETE WINDOW [Gold-F7] deletes the current window. The cursor is put in the previous window.

NEXT WINDOW [Gold-O], [F8], [Gold-Next-Screen] goes from one window to the next one (i.e. below, or to the top one if in the bottom window) when there are many windows on the screen.

PREVIOUS WINDOW [Gold-Prev-Screen] goes from the current to the previous window (i.e. upper, or the bottom one if in the top window).

SET WIDTH [Gold-W] sets the width of the window. Any character after the width is displayed as a diamond. The default width is the "small window size", defined as the WIDTH of your terminal before calling EDFOR as long as it is less than 125 characters (Vaxstation limit for 'large' characters), and 80 characters if not; but the width is updated according to the buffer right margin each time the status line is updated, if you have not disabled the **AUTOWIDTH** feature. (The buffer right margin is set by default to some style_dependent value, see later). If **AUTOWIDTH** is in effect (the default), it's useless to change the width with this command (the width will be recomputed from margin settings the next time you update the status line). Use the **[Gold-Left],[Gold-Right]** to force width and margins simultaneously.

SET AUTOWIDTH enable automatic width selection ("small window size", see previous command, or 132) depending on right margin value for the buffer, this margin being computed when the buffer is first used by looking at the longest line. This is the default.

SET NOAUTOWIDTH disables this feature, useful if you run on a terminal without 132 column support. You can make this behaviour the default for all subsequent editing sessions by adding one line in your **MY_EDITOR_TPU** file, see the last chapter "USER VARIABLES".

Set Width And Margins To Small Window Size [Gold-Left arrow] is used to set simultaneously the width and margin to the "small window size", 80 columns on a standard terminal. The effective margin is not changed for Fortran statements, where it is fixed to 72 for line splitting.

Set Width And Margins To 132 [Gold-Right arrow] is used to set simultaneously the width and margin to 132. Same remark as before for Fortran files.

SHIFT RIGHT nn shifts the window right nn columns. This allows you to look at the ends of lines on a terminal with only 80 columns.

SHIFT LEFT nn shifts the window left nn columns. You can not shift more than the previous right shift.

ENLARGE WINDOW [Gold-Up] increase the number of lines in the current window, when two windows share the screen. You are prompted for the number of lines needed, and the current window is enlarged, the other shrunk, if possible, by the requested amount.

SHRINK WINDOW [Gold-Down] decrease the number of lines of the current window, when two windows are on the screen. You are prompted for the number of lines, and the current window is shrunk, the other enlarged, if possible.

8) INFORMATION

Display Character [Gold-K] gives the current character's value in ASCII, decimal, octal, hexadecimal and control-code if needed.

Describe Key [Gold-H] displays a one line meaning for the next key you type, including Gold and Control keys.

WHAT [Gold-=] gives you a message with the actual column and line number (in the file).

BIG Put the cursor at the end of the biggest line of a buffer, with a message displaying its length.

SHOW displays a page of information per user buffer, showing file name, size, status, ...

VERSION displays the version of EDFOR and the selected experiment flavor.

Keypad Help [PF2] displays on the screen the numeric keypad layout and the meaning of the control keys.

Keypad VT200 Help [Help] displays the VT200 keypad definition (which includes the numeric keypad).

Keypad Gold Help [Gold-PF2] displays the definition of the Gold keys. For any of these 3 helps, you can have help on any key by just typing it, as explained by the status line.

HELP This is the access to a general description of all line commands, and of the user variables, or if you type **HELP VMS xxx** to the standard system help files, or if you type **HELP TPU xxx** access to VAXTPU BUILT-IN help file, or if you type **HELP EVE xxx** access to the standard DEC EVE help file. A short way to access the description of system services and othe system utilities in VMS is to type **HELP xxx$yyy**. This works for the VMS-Help topics 'Lexical', 'RTL' and 'system'. This is as similar as possible to the VMS HELP command syntax, but you can use Next Screen and Previous Screen to scroll through the help information.

NEWS Gives you the changes since the last released version of the editor. This is identical to HELP RELEASE_NOTES.

MESSAGE puts on top of the current window the last part of the message buffer, with Carriage-Return and Line-Feed suppressed. The screen is restored after you answer the prompt.

9) SPELLING

You can check the correct spelling of your files in two ways: Either using your system speller (DCL command `SPELL file_name`) or the in-flight spell checker. The latter can be automatically activated for TeX, Runoff, Mail, Text and Document styles if you set the variable `edf$spell_default` to 1 in your initialization file. You can enable or disable the spelling at any time, thru the command `[Gold-']`. The basis of this spell checker is a program using hash functions and a lookup table which comes from Maki Sekigushi at Fermilab. Some logical names have to be defined, see the INSTALLATION paragraph in the chapter GENERAL DESCRIPTION. This works when called from CALLABLE TPU if the logical names are system defined.

1. Method of operation

When activated, the speller is called for each word you end with a TAB, space or Return. The word is first filtered, to eliminate Runoff, TeX and Vax Document keywords, and to eliminate file or logical names (presence of a period or a dollar sign). If it survives, then the word is sent to the callable speller, a program written in C which checks the word against a dictionary.

It is claimed that it contains 25000 words in its dictionary. The hart of this spelling checker is the hash-search method. The hash method uses eleven different functions, each of which maps a word (ASCII values) to a certain bit location in a 400000 bit table. For a given word, it checks all 11 bits whose addresses are calculated from those functions; a word corresponds to a certain bit-pattern. If all the eleven bits are on, the word is declared to be there. In this way the 400000 bit table acts as a 25000 word dictionary. This method requires only small CPU time and a very little disk space for the program image and dictionary. The 25000 word dictionary is just 99 blocks. This is why it can perform the interactive checking in the editor, where the minimum overhead and CPU time are required. In addition, the program uses very efficient handling for both prefixes and suffixes, by which it can reduce the number of words in the dictionary. This dictionary recognise British AND American spelling.

Any kind of combination of correct-spelled prefixes + right word is valid such as: superparticle, super-superparticle, prequark, preprequark, superprequark, electroweak, electromagnetic, superinteraction, pseudoelectroweak. The followings are the valid prefixes: "anti", "bio", "dis", "electro", "en", "fore", "hyper", "intra", "inter", "iso", "kilo", "magneto", "meta", "micro", "milli", "mis", "mono", "multi", "non", "out", "over", "photo", "poly", "pre", "pseudo", "re", "semi", "stereo", "sub", "super", "thermo", "ultra", "under", "un",

The case of the word is relevant: A word entered in lowercase like 'test' in the dictionary (general or personal) is right spelled as 'test', 'Test', and 'TEST'. A word entered as 'Test' is right spelled as 'Test' and 'TEST', a word entered as 'teSt' is right spelled as 'teSt' and 'TeSt', and if entered as 'TEST', this is the only spelling.

If a word contains non-alphabetic characters between words, they act as separators, and each individual component is tested. The word is recognised as correctly spelled only if all elements are correct. Single alphabetic letters are by definition correctly spelled.

You may have a personal dictionary, which is created the first time you use the speller. This is the file `SPELL$PERSON_DICT` which is usually `MY_DICTIONARY.DICT` in your login area. You can enter new words in this dictionary either by the command `SPELL INSERT`, or just by editing the file. If you find missing words of general interest, please send them to `VXCERN::CALLOT`, for inclusion in the next release of the dictionary.

2. Commands

`SPELL ON [Gold-']` turns spelling on. It requires a few seconds to load the dictionary. After that each time you hit Space, Tab or Return, the previous word is checked against the dictionary. Any unknown word is tagged in reverse video, and a message appears in the message window.

`SPELL OFF [Gold-']` turns spelling off. Turning it on again later is faster than the first time.

`SPELL ACCEPT word` Add a word to the dictionary temporarily (i.e. during the editing session). You are prompted for the word.

`SPELL INSERT word` Add a word in the personal dictionary. You will then have this word each time you use spelling check.

`SPELL IT` to spell the current buffer from the current position to the end of the buffer. It stops after the first misspelled word.

`SPELL BUFFER` Saves the buffer, and call the system SPELL command for the file. This of course requires that a DCL command SPELL is available on your system... Because the speller can change the file, the new version of the file is read into the buffer at the end of the command.

10) VMS SERVICES

`DCL [Ctrl/D]` followed by any valid DCL command : This command is executed in a subprocess, and the output is put in the DCL buffer. This buffer is mapped as follows: If you had only one window, the screen is split and the DCL buffer is mapped onto the lower window, the cursor stays in the current buffer. If there were already two windows, the DCL buffer is mapped in the other window, i.e. the one where you are not, and the cursor is not moved. This command can not be used for interactive commands, i.e. you can not answer any question. You have to use one of the following commands for the most frequent cases, or SPAWN.

`MAIL` allows you to read/send mail as from DCL level. When exiting from MAIL, you will return to your editing session.

`PHONE` runs PHONE in a subprocess. You can optionally give an argument, as from DCL, like ANSWER for answering an incoming call.

`CMS xxx` calls CMS in a subprocess and (if xxx exists) executes this command and exits. If xxx is not given, you stay in CMS until you explicitly exit.

`DIRECTORY file_specs` to put in the DCL buffer the result of the command `$ DIR file_specs`.

`PURGE file_specs` to Purge the requested files. Any output (if you defined `PURGE` with `/LOG`) is shown in the DCL buffer.

`SPAWN`, `[Gold-Z]` suspends temporarily your editing session, and puts you in a subprocess. You can do anything you want, and you will return to your editing work upon LOGOUT.

`ATTACH` lets you go back to your parent process when you called this editor from a subprocess.

`COMPILE` (Fortran style only) saves the current buffer and then compiles it. You can give options to the compiler, like `/D_LINES`, if you want, but separated by at least a space from the command itself. You can review the errors (if any) by `[Ctrl-N]` (next error) and `[Ctrl-P]` (previous error). The errors are stored in the DCL buffer, where you can look at them at once if you prefer. Positioning at the error works with the following restrictions: If you have include files that call other include files, we will not point inside this second level nor look for a third level of include files. Any error related to something after an END statement (labelled generally as Program MAIN$PROGRAM by the compiler) will not be found correctly, because we rely on the module name to point to the correct routine. If you don't understand the error message, or didn't find the source of the error, you can exit the editor and use the `FORTRAN/LIST` command to find the problem.

`SCRIPT` process (at CERN) the current SCRIPT or SGML buffer to obtain a formatted output.

11) OTHERS

1. Key definitions

`DEFINE KEY [Ctrl-K]` allows you to (re)define the function of one key: type the line_command you want to perform, and then the key you want to type to perform this command.

`LEARN [Ctrl-L]` starts the definition of a sequence of operations you want to remember. When the sequence is complete, type `Remember [Ctrl-R]`, and then the key you want to type to execute again this sequence. You can speed-up any repetitive editing task by this mechanism. Be careful in the use of string search in such a sequence: The sequence will continue even if the string was not found. Try to put the search at the end, and you can then decide to continue or not.

2. VAXTPU

`TPU xxx [Gold-Do]` allows you to execute any single line VAXTPU statement, like a call to non-EVE procedures. Please read the VAX/VMS Text Processing Utility VAXTPU reference manual...

`EXTEND TPU xxx` searches for a VAXTPU procedure named 'xxx' (* is a valid wildcard), and compiles it. You are supposed to be editing a buffer containing VAXTPU procedures.

`SAVE EXTENDED TPU xxx` saves the complete actual editor contents (the normal EDFOR plus all your key definitions and learn sequences) in the file xxx (Full file name specification, please), for later use as a section file in the DCL command EDIT/TPU/SECTION=xxx . Be careful, this file is very big (around 1500 blocks). If you want to keep some changes, you can put them in your `MY_EDITOR.TPU`. But you have to write them in VAXTPU...

3. Others...

`SORT BUFFER xxx` sorts the lines of a given buffer in alphabetic order. The default is to sort the current buffer, after confirmation.

KEYPAD DESCRIPTION

1. Numeric Keypad

[PF1] : This is the Gold key to generate a second function for other keys.
[PF2] : Displays Help on the numeric keypad lay-out.
[PF3] : Finds the next occurrence of a previously searched string.
[PF4] : Deletes the line from cursor to end of line, appending the next line.

[KP7] : Finds an ASCII form-feed.
[KP8] : moves to the next or previous screen, depending on current direction.
[KP9] : Appends the selected range to the INSERT-HERE buffer.
[KP-] : Deletes the current word, or the space to the next word if between words.

[KP4] : Sets the direction to FORWARD.
[KP5] : Sets the direction to REVERSE.
[KP6] : Removes the selected range, and puts it in the INSERT-HERE buffer.
[KP,] : Deletes the current character.

[KP1] : Moves to the beginning of next (previous if REVERSE) word.
[KP2] : Goes to the next (previous if REVERSE) end of line.
[KP3] : Moves to the beginning of the previous word.
[Enter] : Generates some Style dependent extensions. See next chapter.

[KP0] : Goes to the next (previous if REVERSE) beginning of line.
[KP.] : Begin a Selected range, or cancels it if one is active.

2. Gold numeric keypad

[Gold-PF2] : Helps on the Gold-alphabetic extensions.
[Gold-PF3] : Finds a string.
[Gold-PF4] : Undeletes the previously erased line.

[Gold-KP7] : Enters the line command mode.
[Gold-KP8] : Fills the selected range
[Gold-KP9] : Replace, prompts for Old and New strings.
[Gold-KP-] : Undeletes the previously erased word.

[Gold-KP4] : Goes to the bottom (end) of the buffer.
[Gold-KP5] : Goes to the top (beginning) of the buffer.
[Gold-KP6] : Inserts here the content of the INSERT-HERE buffer.
[Gold-KP,] : Undeletes the most recently erased character.

[Gold-KP1] : Changes the case of the current character or Selected range if any.
[Gold-KP2] : Deletes to the end of the line, without appending the next one.
[Gold-KP3] : Go to the next place_holder on the current line But after a Gold-number sequence, Inserts the character whose ASCII decimal value was given.
[Gold-Enter] : Replace, prompts for Old and New strings.

[Gold-KP0] : Opens a new line at the cursor position.
[Gold-KP.] : Cancels the current selection.

3. Control keys

`[Ctrl/A]` : Changes the Mode INSERT <==> OVERSTRIKE.
`[Ctrl/B]` : Recalls previous command(s).
`[Ctrl/C]` : Aborts a VAXTPU procedure. To terminate endless loops...
`[Ctrl/D]` : Executes a DCL command.
`[Ctrl/E]` : Goes to end of the current line.
`[Ctrl/F]` : Inserts a Form-Feed.
`[Ctrl/H, Backspace]` : Goes to start of the current line.
`[Ctrl/I, TAB]` : Inserts some string (generally spaces) at the beginning of a line. If typed in the middle of a line, goes to the next tab stop (every 8 columns).
`[Ctrl/J, Line-feed]` : Erases the previous word.
`[Ctrl/K]` : Defines a key: Associate a line_command to a key.
`[Ctrl/L]` : Starts a Learn sequence, end by Ctrl/R.
`[Ctrl/M, Return]` : Splits the line at the cursor position. Indents if the cursor was after the right margin.
`[Ctrl/N]` : Positions on the next Fortran error after compilation.
`[Ctrl/P]` : Positions on the previous Fortran error after compilation.
`[Ctrl/R]` : Terminates a Learn sequence.
`[Ctrl/T]` : Gives your process status.
`[Ctrl/U,Ctrl/X]` : Erases from cursor to start of line
`[Ctrl/V]` : Quotes a character, insert the next key typed without processing.
`[Ctrl/W]` : Refreshes the screen.
`[Ctrl/Y]` : Interrupts. Type CONTINUE to recover the aborted session.
`[Ctrl/Z]` : Exits, writing modified Buffers to their outputs.

4. Gold-typing keys

`[Gold-A]` : Changes mode: INSERT <==> OVERSTRIKE
`[Gold-B]` : Goes to the Buffer whose name you are prompted for.
`[Gold-C]` : Centers the current line.
`[Gold-D]` : Goes to the Default buffer, the one you edited first.
`[Gold-E]` : Loads the source of the given Entry point.
`[Gold-F]` : Fills the current paragraph, with left and right alignment.
`[Gold-G]` : Gets a file whose name you are prompted for. Wildcard allowed.
`[Gold-H]` : Helps on the next typed keys
`[Gold-I]` : Inserts the file you are prompted for.
`[Gold-J]` : Jumps to the position marked by the previous [Gold-X].
`[Gold-K]` : Displays the current character codes (Decimal, Octal, ASCII)
`[Gold-L]` : Lowercases the current word.
`[Gold-M]` : Sets Margins, you are prompted for left and right values.
`[Gold-N]` : Gets in a new window the file in the current INCLUDE statement.
`[Gold-O]` : Goes to the other window, if there are two windows on the screen.
`[Gold-P]` : Searches for the next Fortran page, a "1" in column 1.
`[Gold-Q]` : Quits, without saving the buffer. Prompts if there are modified buffers.
`[Gold-R]` : Returns to the previous buffer.
`[Gold-S]` : Saves the buffer in the file you are prompted for.
`[Gold-T]` : Toggle two windows <==> one window.
`[Gold-U]` : Uppercases the current word.
`[Gold-W]` : Sets Width to the value you will give.
`[Gold-X]` : Marks the current position, use [Gold-J] to come back here.
`[Gold-Y]` : Writes the INSERT HERE buffer to a given file.
`[Gold-Z]` : Spawns a subprocess. Returns upon Logout.

5. Gold special keys

[Gold-DEL] : Deletes to the beginning of line.
[Gold-TAB] : Inserts an ASCII tab.
[Gold-number] : Executes the next command (number) times.
[Gold-Up] : Increases the size of the current window by the requested amount of lines.
[Gold-Down] : Decreases the size of the current window by the requested amount of lines.
[Gold-Left] : Sets width and margin to the "small window size", 80 on a VT100/200 terminal.
[Gold-Right] : Sets width and margin to 132.
[Gold-Return] : Opens a new line with indentation.
[Gold-[ ] : Sets RECTANGULAR mode, changes the Insert/Remove behavior.
[Gold-]] : Sets the normal (NORECTANGULAR) mode.
[Gold-(] : Matches the corresponding closing/opening bracket.
[Gold-)] : Matches the corresponding closing/opening bracket.
[Gold-?] : Shows all the user buffers.
[Gold-=] : Gives the current column and line.
[Gold--] : Draws a line. The direction is given by typing on the arrows, and typing any other key ends this line drawing mode.
[Gold-|] : Draws a box in rectangular mode, from selected point to current cursor position.
[Gold-'] : Sets the in-flight speller ON (or OFF if it was ON before).

6. VT200 keypad

[F6] : Interrupts, back to DCL level. Type CONTINUE to recover.
[F7] : Splits the current window in two windows (works only from one to two in this version).
[Gold-F7] : Deletes the current window (decrease the number of windows on the screen)
[F8] : Goes to the other window.
[F9] : Quits, without saving buffers.
[F10] : Exits, saving all modified buffers.
[F11] : Changes the current direction, FORWARD <==> REVERSE.
[F12] : Moves by line, by end-of-line in FORWARD, by Beginning-of-line in REVERSE.
[F13] : Erases the current word.
[F14] : Changes the mode, INSERT <==> OVERSTRIKE.
[Help] : Draws the VT200 keypad, allows help on any key.
[Do] : Prompts for a line command, and executes it.
[Gold-Do] : Executes a VAXTPU command.
[F17] : Returns to the previous buffer.
[F18] : Capitalizes the current word.
[F19] : Lowercases the current word
[F20] : UPPERCASES the current word.

[Find] : Finds the string you will give to the prompt.
[Gold-Find] : Wildcard search of a string or an expression.
[Insert Here] : Inserts at the current position the content of the INSERT-HERE buffer.
[Remove] : Removes the selected range, if one exists.
[Gold-Remove] : Stores the selected text (without removing it) in the INSERT-HERE buffer.
[Select] : Starts a selected range, or cancels one if one exists.
[Gold-Select] : Resets the selection.
[Prev Screen] : Moves the cursor one screen backwards.
[Gold-Prev Screen] : Goes to the previous window (up on the screen)
[Next Screen] : Moves the cursor one screen forwards.
[Gold-Next Screen] : Goes to the next window (down on the screen)

STYLES

A style defines the behaviour of this editor in the following circumstances : TAB, Indentation when a line is split, Default settings, action on empty buffers, and commands related to the [Enter] key. It also defines a string to be put on the status line. This string is used as a match string for the command `SET STYLE xxx`, used to force a style. By default, the style is determined from the file_type of the output file, and the correspondance will be described for each style.

1) NEUTRAL

This style has NO name on the status line, and NO file type. In fact, NEUTRAL is the default style for all buffers where no other style is defined. This includes system buffers, and new buffers if they don't have an output file. By default, the TAB stops are set every 8 columns. Width is automatic. There is no action on empty file and there are no [Enter] extensions.

1. TAB

The first TAB position is in column 5 and the other every 8 columns, starting in column 9, like the standard DEC setting. The first position can be changed by the command `SET STEP nn`, default is 4, and the other positions can be changed by the command `SET TAB EVERY nn`.

2. Indentation

When the line is split, we indent the 'continuation' line by the same number of spaces as the previous line. This allows you to have indented paragraphs automatically. If the current line is the beginning of a paragraph (i.e. the previous line is empty), we decrease the indentation by the step (default 4), unless the paragraph starts with a non alphabetic character. In the latter case, we align the text on the first alphabetic character of the first line in the paragraph.

2) Text

This is NEUTRAL style, but with optional automatic starting of the speller (if `edf$spell_default` is equal to 1), and without asking for validation in the `FILL PARAGRAPH [Gold-F]` command. It is the default for files with extension .TXT and .DOC.

3) FORTRAN

Style name is FORTRAN and is chosen for file types .FOR or .INC, but we have to select between FORTRAN and VAX_FORTRAN: For DELPHI, all Fortran files are VAX_FORTRAN style. If you have defined `edf$default_vax_fortran` to be '1', then every fortran is VAX_FORTRAN. If you have defined it to be '-1', then every fortran file is FORTRAN. If you have the default value of '0', then the style is FORTRAN only when the file name is at most 6 characters and doesn't contain an underscore. Otherwise the style is VAX_FORTRAN.

1. **Default**

The margin is set to 79, and the TAB step to 2 (variable `edf$default_step_fortran`. The first tab position is in column 7 by default (i.e. for the first line). TAB stops are every 8 columns. If the file is empty, we generate a frame: You choose `PROGRAM` (in this case you can have the experiment's logo),`FUNCTION`, `SUBROUTINE`, `COMMON`, with name taken from the file name, or none. If you select FUNCTION, you are prompted for the type of function (INTEGER, REAL, LOGICAL, CHARACTER).

2. **TAB**

We try to determine the correct indentation: We look for the previous Fortran line that is neither a comment (`C` `!` or `*` in column 1) nor a continuation line (non space in column 6). If no previous line is found, the first tab is in column 7, otherwise, we keep the same indentation for the new line, and look to see if we want more indentation for the new line. This occurs after an `IF(...)  THEN` statement, after a `DO ..  = ..  , ..` statement, after an `ELSE`, `ELSEIF` statement, and after `STRUCTURE`, `MAP` and `UNION` statements, where the indentation is incremented by the tab step.

If the TAB is typed in any column before the first TAB position , except for column 1, we look to see if the beginning of the line is a statement label. If so, we right align the label in column 5, and a normal indentation is performed, as described in the previous paragraph.

3. **Indentation**

When a Fortran line goes over column 72 (or the right margin value for a comment line), we split the line at the end of the last complete word We avoid splitting inside a string, i.e. if there is an unbalanced number of quotation marks in the line, then we go back to the last opening quote (quotes in an inline comment, and the possible quote used in a read/write statement to access a file by records are not misinterpreted). The remaining text is put on a new line, with some prefixed text which depends on the type of the current line.

- If the current line contains an inline comment character `!`, then the new line will be a comment line with the `!` comment character at the same offset.
- If the current line is a comment (`C` or `*` in column 1), the new line will be a comment with up to 40 leading characters copied from the previous line, as long as these characters are spaces or `= + - *` . This is done to keep the indentation of the comment lines.
- If the current line is a continuation (i.e. has any non blank character in column 6), then we keep the same beginning of line up to the first non blank character after column 6.
- For the remaining normal fortran lines, we generate a continuation line indented by tab_steps to match the current line.

All this technical description seems very complicated but in fact, it works well, and you usually don't need to know all these things. Just in case of problems ...

4. **ENTER extensions**

There are a lot of these. After typing `[Enter]`, you are prompted for another character, whose value determines the action. It can be :

[A] For DØ: This saves the current buffer, then calls the D0FLAVOR/CHECK code checker to validate your programming style. This is mandatory before submitting code to the DØ librarians...

For others, this generates the DIMENSION declaration.

[B] Generates a BYTE declaration, but not for DØ.

[C] Inserts a `CHARACTER*` declaration statement.

[D] Generates a DO loop after the current line. We try to choose a LABEL, usually 10 more than the previous label (or, if there is no previous label, the next one). If the new label already exists, we try every value starting from the previous label plus one, by steps of one. We propose this value, but you are always free to give another one. But in any case, we check that the label is not already used. The statements `DO nnn =  ,` and `nnn CONTINUE` are then generated with the current indentation. The cursor is positioned before the = sign so you can insert the index variable name, then go to the comma by the `[Gold-KP3]` key to fill the mandatory strings.

[E] Generates the `EQUIVALENCE` declaration.

[F] Generates the FUNCTION header, with comments, RETURN, and END statements. Your name and the date are used to generate a comment line in the header. You are prompted for the type of function (INTEGER, REAL, LOGICAL, CHARACTER).

[H] Help on the current word, supposed to be a Fortran keyword.

[I] Generates an IF (...) THEN / ELSE / ENDIF structure with the current indentation. You're positioned inside the parentheses to describe the condition. If you were at the end of an ELSE statement, that statement is changed to an ELSEIF(...) THEN and another ELSE is generated.

[J] Introduces the INTEGER declaration statement.

[K] Generates a comment block.

[L] Introduces the LOGICAL declaration statement.

[N] Introduces the INCLUDE '...INC' statement, and you fill in the file name. It is prefixed by DO$INC: for DØ, and followed by /LIST for DELPHI.

[P] Introduces the PARAMETER (... =) statement.

[R] Introduces the REAL declaration statement.

[S] Generates a SUBROUTINE header and frame, like FUNCTION (see [F]).

[U] Adds a line in the current routine header, with Update date and author name, after the previous create (and update) line. If we can't find such a line, we insert the Update stamp at the current position.

[V] Saves the buffer (checking whether any line extends over column 72), any other fortran buffer after a prompt, and compiles it. If compilation errors occur, you can go to the source line at fault by using [Ctrl-N] to find the next one and [Ctrl-P] to go back to the previous one.

[X] Generates a Main Program structure.

[Z] DØ only: Introduces the INCLUDE 'DO$LINKS:...LINK' statement, and you fill in the file name.

[=] gives you the current routine name.

[.] creates and IF(first) THEN / first = .FALSE. / ENDIF structure at the current location, and declares first as logical, SAVE and preset as true after the previous declarations in the same routine.

[PF2] Displays a list of these commands.

4) VAX_FORTRAN

The style name VAX_FORTRAN is chosen for file types .FOR or .INC if the file_name contains underscores, or is bigger than 6 characters, or if you put edf$default_vax_fortran := 1; in your MY_EDITOR_TPU file, or if your experiment is DELPHI. The main differences from style FORTRAN are in the Enter extensions, and the way CLEAN uppercases the file.

1. Default

The margin is set to 79, and the TAB step to 2 (variable edf$default_step_vax_fortran) . The first tab position is in column 7 by default. TAB stops are every 8 columns. If the file is empty, we generate a frame: You choose whether you want a PROGRAM (in this case you can have the experiment's logo),FUNCTION, SUBROUTINE, COMMON, with a name taken from the file name, or none. If you select FUNCTION, you are prompted for the type of function (INTEGER, REAL, LOGICAL, CHARACTER).

2. TAB

This is exactly the same as in FORTRAN style.

3. Indentation

This is exactly the same as in FORTRAN style.

4. ENTER extensions

There are again a lot of these. After typing [Enter], you are prompted for another character, whose value determines the action. It can be :

[B] Introduces the BYTE declaration statement.

[C] Introduces the CHARACTER* declaration statement.

[D] Generates a DO loop after the current line, terminated with an ENDDO statement.

[E] Generates an EQUIVALENCE statement.

[F] Generates the FUNCTION header, with comments, IMPLICIT NONE, RETURN, END statements. It uses your name and the date to generate a comment line in the header. You are prompted for the type of function (INTEGER, REAL, LOGICAL, CHARACTER).

[G] for DELPHI, declares a Global Parameter. You will be positioned at the correct place. Gold-J will put you back where you were.

[H] Help on the current word, supposed to be a Fortran keyword.

[I] Generates an IF (...) THEN / ELSE / ENDIF structure with the current indentation. You're positioned inside the parentheses to describe the condition... If you were at the end of an ELSE statement, that statement becomes an ELSEIF(...) THEN and another ELSE is generated.

[J] Introduces the INTEGER declaration statement.

[K] Generates a comment line, with one empty comment before and after.

[L] Introduces the LOGICAL declaration statement, or (DELPHI) declares a Local variable (see [G]).

[M] Generates a MAP / END MAP structure.

[N] Introduces the INCLUDE '...INC' statement, and you fill in the file name. For DØ, the name is prefixed by DO$INC:

[O] Generates the RECORD / / declaration statement.

[P] Introduces the PARAMETER (... =) statement, or (DELPHI) declares a formal Parameter (see [G]).

[Q] Generates the UNION / END UNION structure, with a first MAP / END MAP structure inside.

[R] Introduces the REAL declaration statement.

[S] Generates a SUBROUTINE header and frame, like FUNCTION (see [F]).

[T] Generates the STRUCTURE / / END STRUCTURE frame.

[U] Add a line in the current routine header, with Update date and author name, after the previous create (and update) date.

[V] Saves the buffer (checking whether any line extends over column 72), and compiles it. If compilation errors occur, you can go to the source line at fault by using [Ctrl-N] to find the next one and [Ctrl-P] to go back to the previous one.

[W] Generates the DO WHILE () / ENDDO construct for another kind of DO loop.

[X] Generates a Main Program structure.

[Z] Only DØ : Introduces the INCLUDE 'DO$LINKS:...LINK' statement, and you fill in the file name.

[=] gives you the current routine name.

[!] declares an inline comment with the same indentation as previous ones (default at column 41).

[.] creates and IF(first) THEN / first = .FALSE. / ENDIF structure at the current location, and declares first as logical, SAVE and preset as true after the previous declarations in the same routine.

[PF2] Displays a list of these commands.

5) PATCHY

This style is used for .CRA files in DØ, and is very close to FORTRAN, except in some minor aspects : The SUBROUTINE frame contains a PATCHY `+DECK` card, and indentation is not propagated over PATCHY control cards. It is also used for .CRA and .CAR files for DELPHI, with the offline subroutine frame. The [G], [L] and [P] extensions are the DELPHI version as described for Vax Fortran.

6) HISTORIAN

The style name is HISTORIAN and is chosen for file types .CRA, .CORR, and .INPUT in ALEPH. This is a modified version of the FORTRAN style, with small differences in the way the SUBROUTINE,... frames are generated. They now include the HISTORIAN control cards. All else is as in FORTRAN style, except you can't compile the buffer (no [V] extension) but you can call a comdeck ([N] extension) or create one with a [Q] extension. Indentation is not propagated thru the Historian control records *I or *D.

7) PASCAL

The style name is "PASCAL" and is given to file type .PAS

1. Default

The TAB step is set to 2, the margin is automatic. If the file is empty, we generate the frame for a PASCAL PROGRAM, FUNCTION or PROCEDURE as you choose, whose name is the actual output file name.

2. TAB

We try to compute the correct indentation : From the previous line (starting with an alphabetic character), we extract the first character position. If the previous line doesn't contain any ';', we will increment the TAB position by TAB_step. This works well until there is some 'continuation' line. The next line will then not be correctly indented.

3. Indentation

We look to see if we are in a comment stream. If yes, we keep the same indentation as the previous line. If no, we indent by TAB_step more than the previous line.

4. ENTER extensions

`[A]` Declares an `ARRAY [] OF`

`[B]` A `BEGIN END;` block is created.

`[C]` A `CASE` block is created.

`[D]` A `FOR xxx := 0 TO .... DO` block is created.

`[E]` Declares an external procedure.

`[F]` Creates a FUNCTION module, with header.

`[I]` Creates an `IF THEN ELSE END;` block.

`[J]` Declares an INTEGER variable.

`[K]` Starts a Comment.

[M] Generates a PROGRAM frame.

[N] Generates an INCLUDE statement.

[O] Generates a RECORD structure.

[P] Creates a PROCEDURE module, with header.

[R] Declares a REAL variable.

[T] Declares a TEXT variable.

[X] Calls the PASCAL compiler for the current buffer.

[PF2] Displays a list of these commands.

8) C_Language

The style name is C_Language and is given to .C and .H files. You can select to have real TAB (default, if `edf$default_step_c := 0;`), or have the number of space given by this variable as indentation.

1. Indentation

When using the Return key, a new line is created and automatically indented, like if you used the TAB key. Indentation is the same as the previous line, is incremented inside a block or if this is a continuation line (previous line not ending with ';', '/', ' ' or TAB), and aligns the beginning of the text in a comment block.

2. Enter extensions

[B] Starts a new block ({} structure).

[C] declares a CHAR variable.

[D] defines a DO loop.

[F] starts a FOR construct.

[I] starts a IF construct.

[J] declares an INT variable.

[K] generates a comment.

[N] adds an INCLUDE statement.

[P] generates a PROCEDURE frame.

[S] generates a SWITCH structure.

[V] Saves the file and calls the CC compiler.

[W] creates a WHILE structure

[PF2] Displays a list of these commands.

9) Vax Macro

This style is for Vax Macro language, and is chosen for file types .MAR .

1. Default

It uses ASCII TAB, every 8 columns. Margins are set to 80 columns. On empty file, we generate a standard frame.

2. Indentation

The comments are recognised, and their indentation kept.

3. ENTER extensions

[E] Creates an ENTRY with subtitle.

[K] Creates a comment.

[L] Includes system macro library SYS$LIBRARY:LIB.MLB.

[M] Creates a standard frame.

[P] Page break

[S] Defines a subtitle.

[V] Compiles the buffer.

[PF2] Display a list of the commands.

10) 68000 Assembler

The style name is Assembler and is given to .ASM files. Its main purpose is to fill the frame. The only extensions are [P] to generate a procedure frame, and [U] to update the modification records.

11) 68000 Real Time Fortran

The style name is RTF and is given to .FTN files. This is a Vax Fortran style, plus the facility (Enter +) to switch to Assembler mode, and vice versa.

12) VAXTPU

The style name is TPU and is chosen for file type .TPU

1. Default

The margin is set automatically and the TAB step is set to 3. If the file is empty, we generate a PROCEDURE structure.

2. TAB

We try to determine the correct indentation, by reference to the previous normal VAXTPU line (start with spaces (or not) then any alphabetic character, or contains '] :' for the CASE statement). The indentation will be the number of spaces up to the first alphabetic character, and will be incremented by Tab_step if the previous line doesn't contain a ";" (This indents the LOOP, IF THEN, ELSE lines, and also continuation lines. But each new continuation line is more indented than the previous one... Nobody's perfect.). Notice that the TAB can result in nothing being put in the buffer if the previous line starts in column 1.

3. Indentation

First, check whether we are in a comment (a `!` character in the line). If so, we generate a continuation line with a `!` at the same column, and with a space after. If not, we take the same indentation as that of the previous line, plus tab_step.

4. ENTER extensions

[C] Generates the `CASE xx FROM 1 TO / [1] : / ENDCASE;` structure with current indentation.

[E] Extends TPU with all procedures in the buffer.

[I] Generates the `IF THEN / ELSE / ENDIF` clause with current indentation.

[K] Generates a comment.

[L] Generates the `LOOP / EXITIF / ENDLOOP` structure with the current indentation.

[P] Generates the procedure header and body using your name and the current date.

[PF2] Displays a list of these commands.

13) DCL

The style name is DCL and is given to file type .COM and .LNK

1. Default

The margin is automatically computed, and the TAB step is 4. If the file is empty, we generate the frame for DCL, i.e. comment plus `ON_ERROR` statements.

2. TAB

This generates a `$` as first character, followed by as many space as needed to keep indentation: Same as the previous line beginning with a `$`, except after a `THEN` or `ELSE`. If you have defined `edf$DCL_ASCII_TAB := 1;` then one uses ASCII TAB instead of spaces.

3. Indentation

If the previous line contains the comment character `!`, we generate a continuation line with `$ !`, the two comment characters being vertically aligned. If there is no comment character, and if the previous line is the first line of a statement (starts with a `$`), we indent by tab_step. If the previous line is already a continuation, we keep the same indentation. And in both cases, we add a "-" at the end of the previous line.

4. ENTER extensions

[G] Puts a `GOTO` command on next line.

[I] Inserts an `IF ... / THEN / ELSE / ENDIF` construct.

[K] Generates a comment Box.

[M] Generates a frame for DCL command files.

[U] Updates the creation date and author.

[W] Adds a `WRITE SYS$OUTPUT` statement.

[PF2] Displays a list of these commands.

14) RUNOFF

The style name is Runoff and is given to file types starting with .RN as the first two characters.

1. Default

The margin is automatically set. If the file is empty, we generate a standard set of initial RUNOFF commands. The TAB and Indentation are the same as in Neutral style.

2. ENTER extensions

You have to know the syntax of RUNOFF to use these extensions fully. Type `[Enter]` then a single character in response to the prompt.

[A] Runoff command `.APPENDIX`, starts an appendix.

[B] Runoff command `.BLANK nn`, skips nn lines.

[C] Runoff command `.CHAPTER`, starts a chapter.

[E] Adds a new element in a list.

[F] Starts a Footnote.

[H] Generates the `.HEADER LEVEL` command, to start a new subdivision in the text.

[L] Starts a new list, use `[Enter-E]` to generate elements in the list.

[M] Generates the standard RUNOFF header and settings.

[N] Starts a NOTE, this is a piece of text with restricted margins.

[P] Starts a new paragraph.

[R] Saves the buffer, and process it by RUNOFF in a subprocess. You will see the RUNOFF error messages (if any) in the lower window of the screen.

[PF2] Displays a list of these commands.

15) TeX

The style name is "TeX" and is given to file type .TEX.

1. Default

The TAB step is set to 4; the margin is automatic. If the file is empty, we generate the frame for a simple TeX file. TAB behaviour and Indentation are the same as the neutral style.

2. ENTER extensions

[B] Selects Boldface characters.

[C] Generates the command to center a line.

[E] fills the remainder of the page with spaces, then starts a new page.

[I] Selects Italic characters.

[M] Generates the frame for a new TeX file.

[P] Starts a new paragraph.

[R] Generates the command to right adjust a sentence.

[T] Selects TeleType characters.

[V] Generates a vertical skip of some space.

[X] Executes TeX on the current buffer, in a subprocess.

[PF2] Displays a list of these commands.

16) LaTeX

1. Default

This style is selected for file types ".LATEX". You can select it for .TEX files if you place the following line in your `MY_EDITOR_TPU` file : `standard_extension{".TEX"} := "LATEX";` The TAB step is set to 4; the margin is automatic. If the file is empty, we generate the frame for a LATeX file. The document style is `article` and the size of output is set to A4. TAB behaviour and Indentation are the same as the neutral style.

2. ENTER extensions

[1] Creates a new (unnumbered) section.

[2] Creates a new (unnumbered) subsection.

[3] Creates a new (unnumbered) subsubsection.

[B] Selects boldface characters.

[C] Center environment.

[E] Select emphasized characters.

[F] Formula (in-text).

[G] Formula (displayed).

[I] Itemize environment.

[M] Generates the frame for a new LATeX file.

[P] Creates a new paragraph title.

[Q] Quotes (“ ”).

[R] Generates the command to right adjust a sentence.

[S] Subscript.

[T] Selects teletype characters.

[U] Superscript.

[V] Insert figure.

[X] Executes LATeX on the current buffer, in a subprocess.

[PF2] Displays a list of these commands.

17) Vax Document

The style name is DOCUMENT and is given to .SDML files, which are then processed by Vax Document. The automatic speller is activated by default if you have set `edf$spell_default` to 1 in your `MY_EDITOR_TPU` file. The TAB is as for the neutral style, the automatic indentation adds spaces at the beginning of the new line if the current line starts with a Vax Document tag. If the file is empty, a standard header page is generated after confirmation. If the procedure `vax_document_logo` is defined, this procedure is executed when creating the header page, just at the beginning of the page. This procedure is intended to create the SDML text to produce your favorite logo on the front page of your document.

1. ENTER extensions

[A] Creates an Argument description, to be used when describing a routine.

[B] Bolds a string.

[C] Creates the description of a COMMAND. This provides the standard options to describes the parameters, the qualifiers, the restrictions, etc.

[D] Creates a definition list. [.] adds an item in the definition list.

[F] Creates a figure.

[I] Creates a conditional section of text, to be processed only if the given keyword is defined.

[J] Italicises a string.

[K] Creates a comment.

[L] Starts a new list. [E] adds an element in the list.

[N] Creates a NOTE in the text, which will appear in bold.

[P] Starts a new paragraph.

[Q] Creates a qualifier description, in the context of a command.

[R] Reference a symbol, whose value will be output in the document.

[S] Creates a SUBROUTINE description, with templates for argument, return value, etc.

[T] Creates a table. [-] adds a row in the table.

[V] Describes the return Value of a subroutine.

[X] Creates an example.

[<] Tags a string.

["] Quotes a string.

[1] Creates a <HEAD1> section

[2] Creates a <HEAD2> section

[3] Creates a <HEAD3> section

[PF2] Displays a list of these commands.

18) SCRIPT CERN paper

The style name is CERNpaper and is given to .SCRIPT files. It is mainly characterized by the generation of a file header, you are prompted for various options as in document style. TAB and indentation are as for neutral style.

1. Enter extensions

[A] Starts an Appendix.

[B] Starts a bullet, close it with extension [E].

[C] Starts a Chapter.

[E] Ends a bullet.

[F] Reserves some room for a Figure; you are prompted for the name and the size of the Figure.

[S] Starts a new section.

[T] Reserves some room for a Table; you give the name and the size.

[Z] Starts a subsection.

[PF2] Displays a list of these commands.

19) SGML

The style name is SGML and is given to .SGML files. Like SCRIPT, its main purpose is to fill the empty buffer.

1. ENTER extensions

[C] Starts a chapter.

[F] Commence un texte en francais.

[accents,tilda,quote,..] create an accentued letter.

20) MAIL

This style name is "MAIL" and is given to file type .MAIL or to file with name starting by MAIL_, because that's the internal name of MAIL generated files. In ALEPH, this style is selected for the file ALEPH$EDITOR.TMP, the AMSEND working file.

1. Default

The width is automatic. On an empty file, we generate the location and time stamp at top right, the signature at the bottom, and you are positioned to start your message. Those two pieces of information (your location and signature) are kept in your MY_EDITOR_TPU file. See NEUTRAL style for TAB and Indentation.

2. ENTER extensions

None.

21) STRUCTURE CHARTS

The style name is "Structure Charts" and is given to file types starting with .STR .

1. Default

The TAB step is set to 8, the margin to 132, the mode to OVERSTRIKE and to RECTANGULAR. This allows you to draw and move boxes more easily. If the file is empty, we generate the frame for structure charts, with file name, author name, notations, ..., and 60 empty lines. See NEUTRAL style for TAB and Indentation.

2. ENTER extensions

[B] Draws a box from a previously selected point to the current cursor position.

[L] Enters the 'line drawing' mode, i.e. when you move the cursor with the arrows keys, a line is drawn on the screen. You exit from this mode by typing any key other than an arrow.

[R] Processes the file thru RUNOFF to produce the .STRUC result.

[*] Draws a small arrow from the current cursor position, with a star as first character. Type one arrow key to give the arrow's direction, any other key cancel the command. This is used to show CONTROL information.

[@] Also draws a small arrow, but with this first character. This is used to show ZEBRA banks.

[o] Also draws a small arrow, but with this first character. This is used to show DATA information.

[PF2] Displays a list of these commands.

22) RCP files

This style is given to .RCP files. The TAB and margins are identical to the neutral style. A default header is generated in case the buffer is empty. Notice that the DCL command RCPSIZE is run on each RCP buffer upon exit.

1. ENTER extensions

[A] Defines an array.

[C] Defines a character string.

[H] Creates the default header.

[K] A comment.

[R] Run RCPSIZE on the buffer.

[U] Add update time and name.

23) ZEB files

This style is given to .ZEB files, the documentation on ZEBRA banks.

1. Defaults

The behaviour is like the neutral style, with margins at 79. A standard frame is generated for empty buffers.

2. TAB, Indentation

A TAB in column 1 copies the beginning of the previous line, as long as it contains only spaces, plus and minus (and C in column 1). Indentation is as in FORTRAN style.

3. ENTER extensions

[B] add a new item at the end of the bank, with the type B, i.e. bit string.

[F] add a new item at the end of the bank, with the type F, i.e. a floating point number.

[H] add a new item at the end of the bank, with the type H, i.e. a Hollerith string of 4 characters

[I] add a new item at the end of the bank, with the type I, i.e. an Integer variable.

[L] add a new link before the existing one. You will have to define it as Structural or Reference yourself.

[M] Add at the beginning of the buffer the standard ZEB frame, to help clean up old ZEB files.

24) Release notes

This style is used only for the file NEW_RELEASE.NOTES. If the file is not empty, a separation line is put at the end. We always add a line with date and author name, and position on the first paragraph to describe the changes in this release. Other properties are the same as the Document style.

EXPANDING EDFOR

1) Adding a new style

Adding a new style 'YYY' can be done in a simple way. You have to provide one procedure, called `edf$YYY_settings( the_buffer, question_number )`, which is called for various values of the second argument, the first one being the buffer for which this style applies. This procedure (and any other procedure you may call from this one) has to be put in a file named `STYLE_YYY.TPU` and placed in either your own login area, or in the `EDFOR$SRC:` area. You are strongly encouraged to look at existing procedures before writing your own. The expected answer for the various values of 'question_number' are:

1: You return the string which will appear on the status line.

2: This is called at initialisation time. The return value is irrelevant, but you can put here actions like margin, width and TAB position settings. You can define `edf$ascii_tab := 1;` to define the TAB key as 'insert an ASCII TAB here'. In this case, automatic indentation can also be performed.

3: You return an integer value, usually 0, but 1 means that this is a fortran file (for column 72 checks, ...) and -1 means that the paragraph filling will be done without prompting for validation. The Speller will be automatically started (if `edf$spell_default` is non zero) and will only work if the value is less than zero, so that you do not spell FORTRAN.

4: This is called when you crate an empty buffer. This is where you put the default content of your buffer. Look at some existing files for idea and tools...

5: This is called when the TAB key is typed. You have to define the variable `edf$tab_string` as being the string to insert at beginning of line, and return the length of the string. The NEUTRAL style is obtained by the statement : `return( edf$default_tab );`.

6: This is called when a line is split, in order to perform indentation and continuation mark generation. The current position is the beginning of the line to be split, and you are not allowed to change the position without restoring it. You return the text to be inserted at the beginning of the new line. The NEUTRAL style is obtained by the statement : `return( edf$auto_indent );`.

7: This is called when you press the Enter key. Look at existing files for a template if needed, the NEUTRAL style is to output the message "No special extensions".

8: This defines a new list of word separators, which overwrite the one given by `edf$word_separators`. If this case is absent, or returns 0, the default value `edf$word_separators` is used.

In order to use this style for some file types, you have to add in your `MY_EDITOR_TPU` file a statement like `standard_extension{".XXX"} := "YYY";`, where XXX is the extension of the file in uppercase, and YYY your new style name. Of course, if you think that this new style may be usefull for other users, please mail the corresponding file to VXCERN::CALLOT, with some explanations. This file can then be distributed to all other users.

2) Compiling styles

The method we use to access a style (read and compile the corresponding file at execution time) allows user expansions. But as loading a new style uses some CPU, it may be faster to compile the most commonly used styles. You just need to create a file `EDFOR$SRC:COMPILED_STYLES.LIST` with one style name per line. These styles are then compiled when building EDFOR. If you change this file, just type at DCL level:

```
    @EDFOR$DIR:BUILD_EDFOR
```

3) Adding a new experiment

An 'experiment' is used to define the FORTRAN and VAX_FORTRAN styles, mainly the subroutine lay-out and the Enter extensions. Adding a new experiment 'XXX" is relatively simple. You must provide a file nammed `EDFOR$SRC:EXPERIMENT_XXX.TPU`, with a few procedures. The procedures needed are :

`for$program_logo` to create the experiment logo in front of a FORTRAN main program.

`for$header` to create the subroutine header for the FORTRAN style. Look at existing experiments files for more details.

`vaxfor$header` to create the subroutine header for VAX_FORTRAN style, it can be a call to the previous routine...

`for$comment` to define the lay-out of a comment obtained by Enter-K.

`for$update` to process the Enter-U key and add an update tag in the current fortran module.

`for$action` to process the Enter extensions in FORTRAN.

`vaxfor$action` to process the Enter extensions in VAX_FORTRAN.

You have better time to modify an existing experiment file than to create one from scratch. To use this style, just edit your `MY_EDITOR_TPU` file and change the variable `edf$experiment`. Send a copy of this file to VXCERN::CALLOT to have this file distributed to other nodes.

USER VARIABLES

This section describes the various variables you can modify by adding a statement in your initialisation file `MY_EDITOR_TPU` which is by default `SYS$LOGIN:MY_EDITOR.TPU`. Variables are defined by a statement like `variable_name := "string";` or `variable_name := 123456;` depending on their type, string or integer. For each of these, the default value is given here.

`edf$experiment := "no"` is a mandatory variable. You are prompted for a value the first time you use this editor. You have to choose a valid value ("ALEPH", "AMY", "D0", "DELPHI", "L3", "LBL", "OPAL" or "no", in fact all the files like `EDFOR$SRC:EXPERIMENT_*.TPU`), or the behaviour may be strange.

`edf$author_name := "string"` is a mandatory variable. If not defined or empty, then you are prompted for your name and a new initialisation file is written. This variable is used to put your name in the header of subroutines, procedures, etc.

`edf$author_signature := "string"` for MAIL style only. Mandatory, you will be prompted if empty. This is the signature to be put at the end of your MAIL. A suggested value is your first name.

`edf$author_location := "string"` for MAIL style only. If not empty, this string followed by the date is put in the top right corner of the file.

`edf$automatic_width := 1;` Select if you want (1 = true) an automatic setting of width of the screen to "small window size" (the size of your terminal when entering EDFOR if less than 120 columns, 80 if not) or 132 columns according to the value of the right margin each time the status line of the buffer is updated. Default is yes. If the right margin is bigger than "small window size", we set the width to 132. If the width is bigger than "small window size" and the right margin less, we set the width to the "small window size".

`edf$word_separators := "',.:];) "; (TAB included)` defines the list of characters used as word separators. These character are part of the words (at the end of the word), and are not suppressed when splitting lines. Only 'whitespace' characters (space and TAB) are removed when splitting a line. Notice that this list can be overwritten for certain styles.

`edf$spell_default := 0;` should be 1 if you want the speller to be started for .TEX, .RNO, MAIL, .DOC and .TXT files.

`edf$default_mouse_on := 0;` Determines if the mouse is by default on or off.

`edf$run_in_subprocess := 0;` to indicate that you run the editor in a subprocess. This changes the way EXIT and QUIT works. If 1, then we ATTACH to the parent process after saving (EXIT) or deleting (QUIT) the buffers, with the same logic as the standard EXIT and QUIT.

`edf$scroll_margin := 15;` defines the fraction (in percent) of the screen where the cursor can not go, because the screen will scroll.

`edf$default_vax_fortran := 0;` To choose between FORTRAN and VAX_FORTRAN styles. If equal to 1, every Fortran file is VAX_FORTRAN. If equal to -1, every Fortran file is FORTRAN. If equal to 0, then all Fortran77 conforming file names (6 characters, no underscores) are FORTRAN style, others are VAX_FORTRAN.

`edf$default_step_fortran := 2;` is the default indentation step for FORTRAN style.

edf$default_step_vax_fortran := 2; is the default indentation step for VAX_FORTRAN style.

edf$fortran_compile := "FORTRAN"; defines the Fortran command line, i.e. your default qualifiers.

edf$fortran_extended_source := 0; defines the check of column 72. 0 means check and prompt if a bad line is found. -1 means check and refuse to write the file. +1 means no check.

edf$fortran_continuation_char := "&"; defines the continuation character to be put on column 6 when a line is split.

edf$vax_keyword_case := "UPPER"; defines how the fortran keywords are modified by the CLEAN command in the VAX_FORTRAN style. Possible values are "UPPER", "LOWER" and "CAPITAL", all in uppercase.

edf$vax_variable_case := "LOWER"; defines how the fortran source statements are modified (on request) by the CLEAN command in the VAX_FORTRAN style. Possible values are "UPPER" and "LOWER".

edf$default_header := 1; defines if you want to be prompted for the type of header (subroutine, function, ...) you want to create when your buffer is empty. If set to 0, no header is ever generated.

edf$default_step_c := 0; defines the way we indent C code: 0 means using ASCII tab. Any positive number means indent by this number of spaces.

edf$c_tab_size := 4; defines the size of an ASCII tab when used for indentation. Use the value 8 to display correctly code written with the standard VAX tab settings.

edf$DCL_ASCII_TAB := 0; Defines if one wants real ASCII TAB for DCL files, the default being to use 4 spaces.

edf$xscript := "XSCRIPT/EXP"; defines the command to run SCRIPT.

edf$sgml := "SGML/EXP"; defines the command to run SGML.

edf$scr_dist := ""; defines the CERN distribution code for SCRIPT outputs.

edf$scr_dev := ""; defines the CERN device for SCRIPT output.

edf$dev_name := ""; **edf$bat_name** := ""; **edf$i3812** := "TC3812A"; are used to describe the output destination for SCRIPT processing at CERN.

standard_extension{".XXX"} := "style"; defines the requested style for all files with extension .XXX (with the dot and in uppercase); The possible values of 'style' are the various style names: 'NEUTRAL', 'FORTRAN', 'VAX_FORTRAN' or any 'ZZZ' with a corresponding file EDFOR$SRC:STYLE_ZZZ.TPU.

standard_file{"XXXX.YYY"} := "style"; defines the requested style for the given file name.