MECAFF - tools

Multiline External Console And Fullscreen Facility - Tools

for VM/370 R6 SixPack 1.2

User Manual

Version 1.1.0

Dr. Hans-Walter Latz

WARNINGS:

This software is delivered as-is with no promise or commitment to be usable for any particular purpose.
Use it at your own risks!
The MECAFF software and documentation has been written by a hobbyist for hobbyists and should not be used for any important or even critical tasks.

MECAFF is work in progress and its current implementation may differ from this documentation.

Contents

1 Overview

The MECAFF tools were initially developed with the MECAFF modification-free non-invasive extension to VM/370 R6 providing full screen capabilities and a serial-style console on 3270 terminals. MECAFF uses an external (Java-) program which provides the fullscreen support as a kind of additional external facility.

With the availability of the DIAG58 CP extension for VM/370R6, the fullscreen-based programs of MECAFF were ported to the now native fullscreen support.

These tools are now available in 2 suites for each variant of fullscreen support for VM/370R6. At the same version level, the tools provide the same functionality in both suites, with possibly marginal differences due to the basic technical differences of both fullscreen support approaches.

The MECAFF tools consist of the following programs:

- **EE**
  fullscreen editor allowing to edit multiple files simultaneously
- **FSLIST**
  fullscreen file lister
- **FSVIEW**
  fullscreen file viewer
- **FSHELP**
  fullscreen online help viewer
- **IND$FILE**
  host side file transfer support (CUT-mode only)

The fullscreen tools EE, FSLIST, FSVIEW and FSHELP support (almost) arbitrary screen sizes of the connected 3270 terminal emulator. The screen size is defined by the emulated model type:

- Model 2 (IBM-3278-2-E, IBM-3279-2-E, IBM-3278-2, IBM-3279-2)
  → 80 cols x 24 lines
- Model 3 (IBM-3278-3-E, IBM-3279-3-E, IBM-3278-3, IBM-3279-3)
  → 80 cols x 32 lines
- Model 4 (IBM-3278-4-E, IBM-3279-4-E, IBM-3278-4, IBM-3279-4)
  → 80 cols x 43 lines
- Model 5 (IBM-3278-5-E, IBM-3279-5-E, IBM-3278-5, IBM-3279-5)
  → 132 cols x 27 rows
- Dynamic /oversize [naming emulator dependent] (IBM-DYNAMIC)
  → arbitrary screen geometry within the 14-bit buffer addressing range (16384 character cells) with at least 24 rows and 80 columns (theoretically allowing screen sizes between 80 cols x 204 rows and 682 cols x 24 rows)

The following sections describe the functionality and usage of these tools. The installation as well as considerations specific to a fullscreen infrastructure (MECAFF-console or DIAG58) is described in separate documents.

2 The MECAFF tools for CMS

The fullscreen tools EE (editor), FSLIST (filelist viewer) and FSVIEW (file browser) are in fact integrated into a single program to allow the navigation between basic functional modes EE, FSLIST and FSVIEW. Depending on how it is invoked from CMS, the program starts in one the EE, FSLIST or FSVIEW modes, allowing different transitions between the functional modes depending on the initial mode. For this reason, the modes EE, FSLIST and FSVIEW will be described as separate items (as if they were separate programs).

The program FSHELP displays the CMS help files on disk U of VM/370 SixPack 1.2 in fullscreen mode, also providing an overview menu page and navigation to other help topics.

The program IND$FILE allows file transfers between the VM inside VM/370 and the computer running the 3270 terminal emulator. Only the CUT-mode of the IND$FILE protocol is supported by the MECAFF-tool IND$FILE.

2.1 Handling disconnect and reconnect

When the terminal connection is lost (the terminal window is closed inadvertently or the MECAFF process is terminated), the VM session is disconnected. If a fullscreen MECAFF tool was running, it will stay in the fullscreen read state until a 3270 terminal is reconnected to the VM.

The program will try to re-establish the fullscreen session when the user reconnects to the VM. When entering the BEGIN command on the CP level after the reconnect, the EE, FSLIST, FSVIEW and FSHELP programs will continue to work and try to restart the fullscreen interaction depending on the fullscreen infrastructure used.

2.1.1 MECAFF-console based tools

For the MECAFF-console based tools, re-establishing the fullscreen session will succeed if the3270 terminal is again connected via MECAFF, allowing to proceed with fullscreen mode after leaving the "More…" state. However, the editing changes made on the last EE screen before disconnection will be lost.

If the 3270 terminal used is directly connected to Hercules (or the terminal is not a 3270 emulation), fullscreen mode cannot be restored. The programs FSLIST, FSVIEW and FSHELP will terminate with an appropriate error message. The editor EE will enter a minimal command line loop ("rescue mode") allowing to save modified files, although no editing is supported (see 2.3.8).

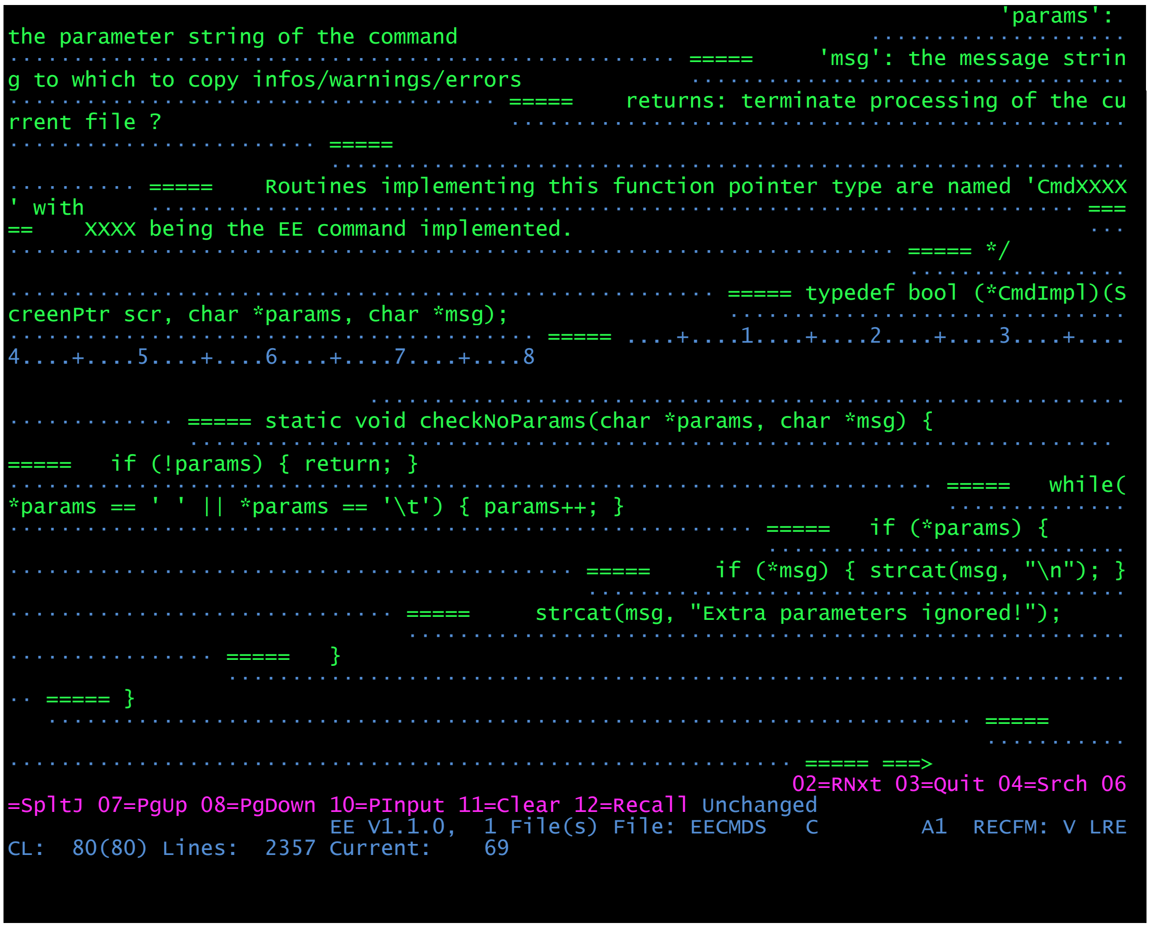
2.1.2 DIAG58 based tools

For the DIAG58 based tools, re-establishing the fullscreen session will succeed if the 3270 terminal is directly connected to the VM/370 host. However, the editing changes made on the last EE screen before disconnection will be lost.

The visible appearance of the 3270 screen depends on the terminal characteristics after reconnect:

- If it has the same screen characteristics (screen size) as the terminal that was disconnected, the screen layout stays unchanged.
- If the disconnected terminal was a model-2 type terminal (80x24), reconnecting to an arbitrary 3270 terminal will also work, however only the top-left 80x24 screen part will be used.

- Disconnecting from a standard 3279 terminal (80 columns) and reconnecting with a non-standard screen size with more columns with same or more lines will result in a scrambled screen after entering the BEGIN CP-command to proceed in the fullscreen program and pressing ENTER to redraw the screen, for example:



```
                                                                             'params':
the parameter string of the command                                ....................
.................................................... =====     'msg': the message strin
g to which to copy infos/warnings/errors             ..................................
...................................... =====    returns: terminate processing of the cu
rrent file ?                           ................................................
........................ =====
                          .............................................................
.......... =====    Routines implementing this function pointer type are named 'CmdXXXX
' with     ........................................................................ ===
==    XXXX being the EE command implemented.                                       ...
..................................................................... ===== */
                                                                     ..................
....................................................... ===== typedef bool (*CmdImpl)(S
creenPtr scr, char *params, char *msg);                 ...............................
......................................... ===== ....+....1....+....2....+....3....+....
4....+....5....+....6....+....7....+....8

                            ...........................................................
............. ===== static void checkNoParams(char *params, char *msg) {
              ........................................................................
=====   if (!params) { return; }
........................................................................ =====   while(
*params == ' ' || *params == '\t') { params++; }                         ..............
.......................................................... =====   if (*params) {
                                                           ............................
............................................ =====     if (*msg) { strcat(msg, "\n"); }
                                             ..........................................
.............................. =====     strcat(msg, "Extra parameters ignored!");
                               ........................................................
................ =====   }
                 ......................................................................
.. ===== }
   ........................................................................ =====
                                                                            ...........
............................................................. ===== ===>
                                                             02=RNxt 03=Quit 04=Srch 06
=SpltJ 07=PgUp 08=PgDown 10=PInput 11=Clear 12=Recall Unchanged
                         EE V1.1.0,  1 File(s) File: EECMDS   C        A1  RECFM: V LRE
CL:  80(80) Lines:  2357 Current:    69
```

- In many other cases, the screen may simply stay blank instead of displaying the file content.

In all cases, pressing **PA3** will force the program to re-query the screen characteristics and rebuild the screen layout according to the new screen geometry, provided it is a 3270 terminal.

If the VM is reconnected through a non-3270-terminal (for example plain telnet), then fullscreen mode cannot be restored. The programs FSLIST, FSVIEW and FSHELP will terminate with an appropriate error message. The editor EE will enter a minimal command line loop (“rescue mode”) allowing to save modified files, although no editing is supported (see 2.3.8).

2.2 Handling of file identifications in EE, FSLIST and FSVIEW

The MECAFF tools EE, FSLIST and FSVIEW allow specifying file identifications with 2 alternative conventions:

- with the “traditional” CMS convention with the components *filename*, *filetype* and (optionally) *filemode* separated with white space (one or more blanks):

  fn ft [*fm*]

- with the dot-convention used by many other OSes, separating the 3 components with a dot, joining them to a single character string:
  fn.*ft*[.*fm*]

Both conventions can even be mixed, so the following identifications specify the same file:

```
PROFILE EXEC A
PROFILE.EXEC.A
PROFILE EXEC.A
PROFILE.EXEC A
```

When specifying a new file identification in EE or a new file pattern in FSLIST, it is possible to reference components from the context (the current file identification or pattern) by giving the equal sign (=) for the corresponding components. For example:

• while editing the file:	`FOO C A1`
specifying:	`= h`
will open the file:	`FOO H A1`
• having the pattern:	`X* MODULE S`
in FLIST and giving:	`=.=.a`
will list the files:	`X* MODULE A`

As a single file identification in dot-notation may be more than 8 characters in length, it can be used for invoking one of the commands EE, FSLIST or FSVIEW only if the extended parameter list is available.
See suite-specific manual for the installation options to ensure that the dot-notation can be used on the command line by all MECAFF fullscreen programs.

2.3 EE – Fullscreen editor

2.3.1 Introduction

EE is a MECAFF-based fullscreen editor allowing to edit files with a LRECL up to 255. Usage and screen layout resemble to other 3270 fullscsreen editors and supports editing several files in parallel.

The editor EE is invoked from CMS with

```
EE  fn ft [ fm ]
```

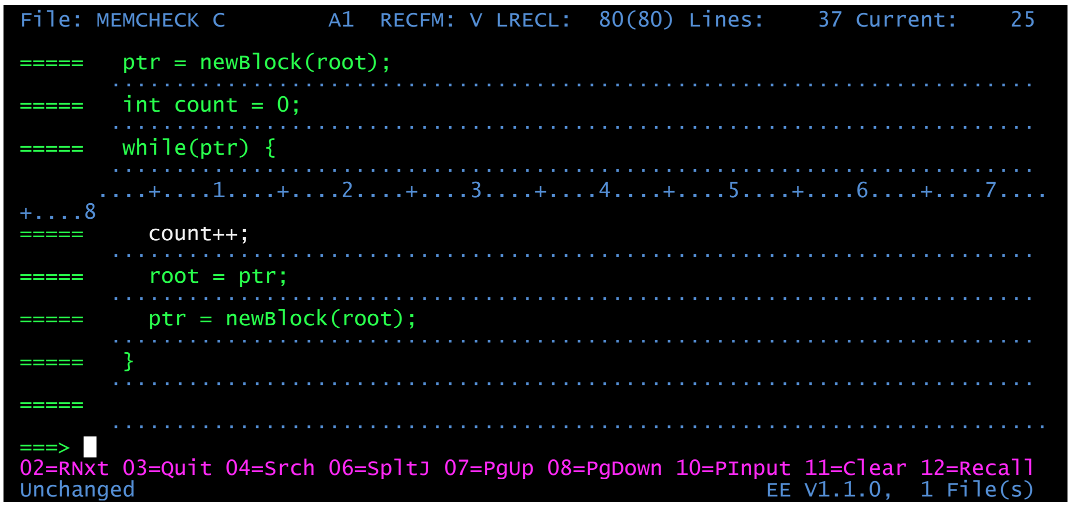
or

```
EE  fn.ft[.fm]
```

If the CMS file `fn ft [fm]` does not exists, the characteristics (RECFM, LRECL, case handling) for the new file are taken from the FTDEFAULTS command (see 2.3.6.4) for the file type (specified in one if EE's profiles, see 2.3.7).

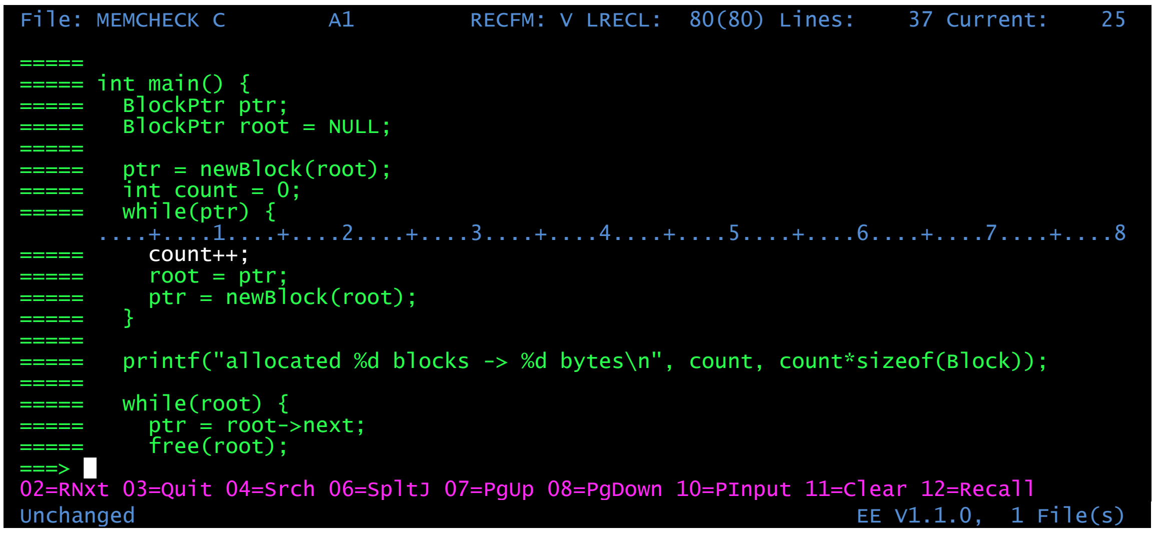
2.3.2 Choosing the "ideal" terminal size for the LRECL edited

Unlike other editors, when editing files with the LRECL larger as the screen is wide (less the prefix zone), EE does not provide "shift left" or "shift right" functionality. Instead, the lines of the file are wrapped at the screen boundaries, each line having a single input field with full LRECL length spanning 2 or more lines on the screen. For an easier orientation, the gap from the end of a line input field to the next border (screen border or the prefix zone if placed at the right) can be marked with fill chars. With DOT as fillchar (see command GAPFill), editing a file with LRECL 80 on a standard 3270 terminal (3279-2 with 80x24 screen size) will have a screen setup similar to the following:



```
File: MEMCHECK C         A1  RECFM: V LRECL:  80(80) Lines:    37 Current:    25
=====   ptr = newBlock(root);
       ........................................................................
=====   int count = 0;
       ........................................................................
=====   while(ptr) {
       ........................................................................
      ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....
+....8
=====     count++;
       ........................................................................
=====     root = ptr;
       ........................................................................
=====     ptr = newBlock(root);
       ........................................................................
=====   }
       ........................................................................
=====
       .........................................................................
===> █
02=RNxt 03=Quit 04=Srch 06=SpltJ 07=PgUp 08=PgDown 10=PInput 11=Clear 12=Recall
Unchanged                                              EE V1.1.0,   1 File(s)
```

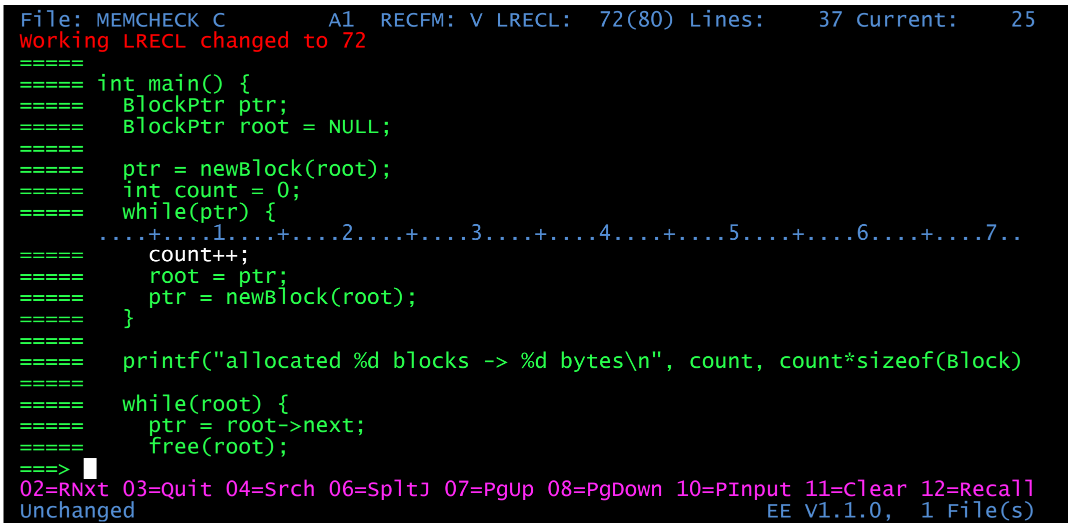
As EE supports non-standard terminal screen sizes, it is possible to choose the terminal size according to the most used LRECL. For example for LRECL 80 files, a terminal with 87 columns is suitable to edit these files in a “natural” way:



```
File: MEMCHECK C         A1         RECFM: V LRECL:  80(80) Lines:    37 Current:    25

=====
===== int main() {
=====   BlockPtr ptr;
=====   BlockPtr root = NULL;
=====
=====   ptr = newBlock(root);
=====   int count = 0;
=====   while(ptr) {
      ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
=====     count++;
=====     root = ptr;
=====     ptr = newBlock(root);
=====   }
=====
=====   printf("allocated %d blocks -> %d bytes\n", count, count*sizeof(Block));
=====
=====   while(root) {
=====     ptr = root->next;
=====     free(root);
===> █
02=RNxt 03=Quit 04=Srch 06=SpltJ 07=PgUp 08=PgDown 10=PInput 11=Clear 12=Recall
Unchanged                                                     EE V1.1.0,   1 File(s)
```

For some type of files, only the first characters up to a fixed column of a larger LRECL are relevant for using the file. For example only the first 72 columns are significant for ASSEMBLER, COBOL, PL/1 or FORTRAN files, where columns 73 to 80 only contain line numbering data. For such file types, it is possible to restrict displaying and editing the file to the first relevant columns with the command `WORKLRECL`. The default working line width for a file type can be predefined with the `FTDEFAULTS` command in the profile files of EE. If the working width specified for a file allows to completely display the editable line subset including the prefix zone in a screen width. Files with up to 73 significant columns will be editable in a “natural” way on a standard terminal screen size like 80x24 (model 2), 80x32 (model 3) or 80x43 (model 4). The following screen shows the same editing situation as the first screen (3279-2 with 80x24 screen size) after issuing the command `WORKLRECL 72`:



```
File: MEMCHECK C          A1  RECFM: V LRECL:  72(80) Lines:    37 Current:    25
Working LRECL changed to 72
=====
===== int main() {
=====   BlockPtr ptr;
=====   BlockPtr root = NULL;
=====
=====   ptr = newBlock(root);
=====   int count = 0;
=====   while(ptr) {
      ....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
=====     count++;
=====     root = ptr;
=====     ptr = newBlock(root);
=====   }
=====
=====   printf("allocated %d blocks -> %d bytes\n", count, count*sizeof(Block)
=====
=====   while(root) {
=====     ptr = root->next;
=====     free(root);
===>
02=RNxt 03=Quit 04=Srch 06=SpltJ 07=PgUp 08=PgDown 10=PInput 11=Clear 12=Recall
Unchanged                                                EE V1.1.0,   1 File(s)
```

However, setting the WORKLRECL smaller than the significant line width may result in truncated lines if lines already having content in the hidden part of the line are edited. This is the case in the above example of a C file where the `printf` statement will be crippled if modified, so setting the WORKLRECL should be used if the file type really allows it.

2.3.3 Opening binary files for edit

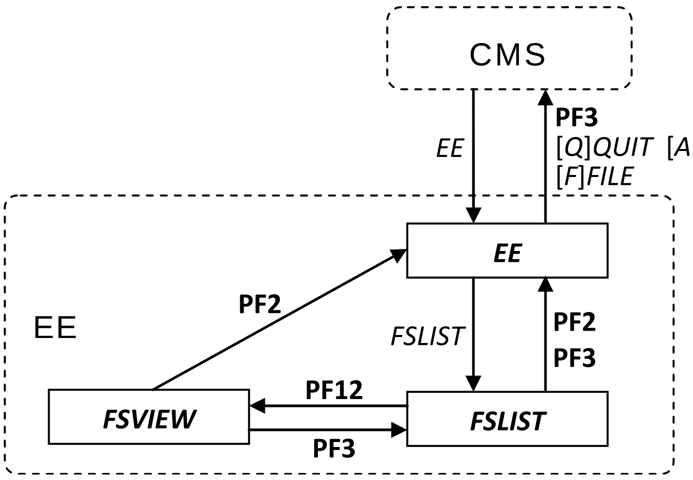
If a file loaded into EE contains binary characters (codes 0x00 to 0x3F or 0xFF), these characters are replaced with a dot ('.') in the file buffer, the file is marked as binary and EE will prevent all write operations to the original disk file. If necessary, this protection can be removed (see UNBINARY).

2.3.4 Editing multiple files and mode transitions

EE allows editing more than one file at a time. After starting EE with the first file, more files can be opened for editing or created with the EE subcommand or by selecting a file for editing from the FSLIST or the FSVIEW modes.

All files opened for editing are held in a ring, from which one file is the currently edited. The commands `RINGNext` or `RINGPrev` switch the currently edited file by stepping through the ring of files loaded.

The following diagram shows the transitions between the modes of the EE program to start or end editing files, also showing with the commands (*CMD*) or PF keys (**PFxx**) to change from mode to mode:



2.3.5 Prefix commands

EE supports the following (case insensitive) commands in the prefix area:

/	→	set this line to be the current line
.*c*	→	set the line mark *c* on this line, replacing a possible existing mark with the same name; line mark names consist of a single letter A .. Z, so up to 26 line marks can be defined per file in an EE session.
@	→	ignore and discard modifications made to this line in the file area on the screen
I[*n*]	→	insert an empty line after this line; if the optional repetition-count *n* is given, *n* lines will be inserted
D[*n*]	→	delete this line; if the optional repetition-count *n* is given, up to *n* lines will be deleted, but deleting at most the lines up to (before) the next prefix command on the screen
DD	→	mark one of the boundaries of a line block to be deleted
"[*n*]	→	duplicate this line; if the optional repetition-count *n* is given, the block of *n* lines (including the line with the prefix command) will be duplicated
""	→	mark one of the boundaries of a line block to be duplicated
*[*n*]	→	duplicate this line; if the optional repetition-count *n* is given, this line is duplicated *n* times (i.e. repeated *n* times)
<[*n*][*o*]	→	shift this line to the left (see below for the optional parameters)
<<[*n*][*o*]	→	mark one of the boundaries of a line block to be shifted to the left (see below for the optional parameters)
<[*n*][*o*]	→	shift this line to the right (see below for optional parameters)
>>[*n*][*o*]	→	mark one of the boundaries of a line block to be shifted to the right (see below for optional parameters)
M	→	move this (single) line to the target specified by either a F or a P prefix command
MM	→	mark one of the boundaries of a line block to be moved to the target specified by either a F or a P prefix command
C	→	move this (single) line to the target specified by either a F or a P prefix command
CC	→	mark one of the boundaries of a line block to be copied to the target specified by either a F or a P prefix command
F	→	place the line(s) to copy or move after (following) this line
P	→	place the line(s) to copy or move before (preceding) this line

The shifting prefix commands support the following optional parameters (which can be specified on any of the boundaries for block shifting prefix commands):

- *n* : the number of character positions to shift by. This must be a single digit from 1 to 9, with the default specified with the `SHIFTCONFig` command if *n* is not specified.
- *o* : the shiftmode for handling shifting beyond the left or right file border, overriding the default given with the `SHIFTCONFig` command. This must be one of the following characters identifying one of the shiftmodes (see the command `SHIFTCONFig` in 2.3.6.4):

?	→	CHECKall
:	→	MINimal
#	→	LIMit
!	→	TRUNCate

If a line block needing a target position (MM .. MM or CC .. CC) is not specified at once on the same screen, the target position (F or P) must be specified after both boundaries of the block are selected (or together with the second boundary). If the block is defined but the target is not, the selected block (both the prefix and the file zones) will be made read-only until the command is finalized by giving the target.

Prefix move and copy operations can be performed between files edited in EE by specifying the source (M, C, MM .. CC or CC .. CC) and target (F resp. P) of the operation in different files (see RINGNext / RINGPrev).

As long as prefix commands do not conflict, more than one single line command and possibly one line range command can be issued on one screen input. Single line commands are processed before a possibly given block command and target position.
If EE cannot ensure a meaningful execution, all prefix commands on the screen are ignored and removed.

After executing prefix commands, EE will try to place the cursor at a meaningful place in the file area. The placement line depends on the prefix command (e.g. the line inserted with the I-prefix command, but the line after the deletion for D/DD-operation). If this line is displayed on the screen, the cursor will be placed at the current indentation (of the last previous non-blank line) for the I-command resp. on the first non-blank character of the line. If the line for the cursor placement is not visible (for example a prefix command was entered but the file content was also scrolled with a PF key), the cursor is placed at the command prompt. If more than one prefix command is given, the cursor is placed by the first (top most) command.

2.3.6 Commands

Commands can be entered at the command prompt arrow with a total command length of 120 characters.

The command and parameter keywords may be abbreviated, the minimal part of the keywords is given in the following descriptions in uppercase.

If the text given on the input prompt is not recognized as a valid command, the input will be treated as a location target (see command `Locate`) if the first token starts with a digit or one of the characters + `-` `.` `:` or /

2.3.6.1 Default PF settings

For the fullscreen editing mode, the meaning of the PF keys can be freely defined by assigning a command to a PF key with the command PF (see 2.3.6.4), either in the profiles for EE (see 2.3.7) or on the EE command prompt. EE uses the following default PF key assignments:

PF01	:	TABFORWARD
PF02	:	RINGNEXT
PF03	:	QUIT
PF04	:	SEARCHNEXT

PF06	:	SPLTJOIN
PF07	:	PGUP
PF08	:	PGDOWN
PF09	:	MOVEHERE
PF10	:	PINPUT
PF11	:	CLRCMD
PF12	:	RECALL
PF13	:	TABBACKWARD
PF16	:	REVSEARCHNEXT
PF19	:	PGUP 66
PF20	:	PGDOWN 66

For the input-mode and the programmers-input-mode, the PF key command assignments are fixed (see the description of these modes in 2.3.6.2).

2.3.6.2 Editing and scrolling commands

`BOTtom`

Move the current line to be the last line of the file.

`Change` */string1/string2/* `[CONFirm] [` *count1* `[` *count2* `] ]`

Replace occurrences of *string1* by *string2*.

The parameter *count1* specifies how many occurrences in each line are to be replaced. If *count1* is omitted, the value 1 is assumed, i.e. only the first occurrence of *string1* is replaced. Specifying * as *count1* replaces all occurrences on *string1*.

The parameter *count2* specifies how many lines from (and including) the current line are to be processed. If *count2* is omitted, the value 1 is assumed, i.e. only the current line is processed. Specifying * as *count2* processes the current and all remaining lines of the file.

As separator character for *string1* and *string2*, any non-alphanumeric and non-blank character may be used (with / being the traditional separator character).

If the optional parameter `CONFirm` is given, each match found is displayed and the substitution must be accepted or denied. For each match:

- the screen changes into read-only mode with the scale and the current line centered
- the cursor is placed on the start of the matched string,
- the position of the match on the current line is marked on the scale by a sequence of #-characters
- the action to be taken for each match is given with one of the following PF keys:
  → PF12: execute the substitution and search for the next match
  → PF04: skip this match and search for the next match
  → PF03: terminate the change operation prematurely at this point

`DELete [` *count* `]`

Delete *count* lines beginning with the current line. If *count* is omitted, 1 line is deleted.

```
GET  [ fn [ ft [ fm ] ] ]
     [fn[.ft[.fm]]]
```

Read the specified file and insert its lines after the current line. If the file inserted contains binary characters (codes 0x00 to 0x3F or 0xFF), these characters are replaced with a dot ('.') in the file buffer, the current file is marked as binary and EE will prevent all write operations of the edited file to the original disk file. If necessary, this protection can be removed (see UNBINARY).

The default filetype is EE$BUF. If *fm* is omitted, the filemode A is assumed. When no file-id is given, the file PUT EE$BUF A is read.

```
GETD  [ fn [ ft [ fm ] ] ]
      [fn[.ft[.fm]]]
```

Like the GET command, but additionally deleting the file after inserting its content.

```
Input [ line-text ]
```

If a *line-text* is given, a new line with *line-text* is inserted after current line and this new line becomes the new current line.

If no text is given after the command, EE enters input-mode, allowing multiple lines to be entered after current line by displaying a free space below the current line. When pressing ENTER after entering lines, EE makes the last entered line to the new current and stays in input-mode, allowing for more lines to be entered. The input-mode is terminated either by pressing the ENTER key without a change of the file content or by pressing the PF03 key.

The input-mode has the following fixed command assignments to PF keys:

PF01	:	TABFORWARD
PF03	:	QUIT (leave input-mode)
PF13	:	TABBACKWARD
PF15	:	QUIT (leave input-mode)

Except for changes in the file zone of the editor, no further operations are possible while in input-mode (no commands, no PF keys others than described above, no prefix commands).

```
Locate target1 [ target2 [ … ] ]
```

Move the current line to be the line specified by the sequence of targets. In most cases, only one target will be used, but several targets may be used instead of consecutive `Locate` commands. Relative and absolute moves are bounded to the begin resp. the end of the file. The current line is only moved if the whole target sequence can be executed, i.e. if given line-marks exist and the given search strings are found.

The following move targets are supported:

n, +*n*, -*n*
→ relative move of the current line by the specified number of lines, giving a positive value (without sign or explicitly with a + character) moves toward the file end, giving a negative value moves to the file begin.

:*n*
→ absolute move to line *n*.

.c
→ move to line mark *c*.

/string/
→ move to the next line in direction to the file end containing *string*. Instead of /, any non-alphanumeric character different from blank and the target-introducing characters (+ - : .) can be used as separator for the search *string*.

-/string/
→ move to the next line in direction to the file start containing *string*. Instead of /, any non-alphanumeric character different from blank and the target-introducing characters (+ - : .) can be used as separator for the search *string*.

```
MARK .c
MARK CLear .c | * | ALL
```

Set (or replace) the specified line-mark *c* resp. remove the specified line-mark or all line-mark(s).

```
MOVEHere
```

If the cursor is in the file area, the line under the cursor will be made the current line, with the cursor being moved with this line and staying at the position inside the line.

```
Next [ count ]
```

Move the current line down by *count* lines (in direction to the file end) if *count* is positive, or up (to file start) if *count* is negative. If no *count* is given, the current line is moved by 1 line.

```
PGDOwn [ pct | lesscount ] [ MOVEHere ]
```

Move the current line one page height down (in direction to the file end). The page height is the number of the visible file lines on the screen.

If a *pct* in the value range 33..100 is given, scrolling will occur for this percentage of the visible file line count.

If a negative value *lesscount* is given, the number of lines scrolled will be the visible file line count minus the number of lines specified (absolute value of *lesscount*). However, scrolling will be done at least by a 1/3 screen height.

If the optional parameter `MOVEhere` is specified and the cursor is in the file area when the command is issued, the line under the cursor will be made the current line, with the cursor being moved with this line and staying at the position inside the line.

```
PInput
```

Enter programmers-input-mode. If the cursor is currently in the file area or the prefix area, the cursor's line is made the current line before the programmers-input-mode is entered. The `PInput` command is assigned to PF10 per default, allowing to start programmers-input-mode directly from the file area.

In programmers-input-mode, a single empty line is inserted after the current line and this line is made current. All file lines on the screen are editable and the SPLTJOIN functionality is available through PF06/PF18.

A new input line is automatically inserted with the ENTER key when text was entered on the input line or when the cursor is on the input line. When inserting the new input line, the cursor is placed at the indentation of the (first non-empty) line preceding the input line, allowing to enter program text in a natural way when working with nested blocks.

Pressing ENTER when no text was entered on the input line and the cursor was moved away from this line simply places the cursor back to the input line at the indentation column without creating a new input line.

The programmers-input-mode can be left with the PF03 or PF15 keys. If the current input line was left unchanged, it is removed when programmers-input-mode ends. If possible, the cursor will be placed in the file area, preferably leaving the cursor at the current position or if this not in the file area by placing it on the same or the last inserted line at the first non-blank column.

The programmers-input-mode has the following fixed command assignments to PF keys:

PF01	:	TABFORWARD
PF03	:	QUIT (leave programmers-input-mode)
PF06	:	SPLTJOIN
PF10	:	PINPUT (move programmers-input-mode to the line after the cursor’s line)
PF13	:	TABBACKWARD
PF15	:	QUIT (leave programmers-input-mode)
PF18	:	SPLTJOIN FORCE

Except for changes in the file zone of the editor, no further operations are possible while in programmers-input-mode (no commands, no PF keys others than described above, no prefix commands).

`PGUP [ ` *`pct`* ` | ` *`lesscount`* ` ]`

Move the current line one page height up (in direction to the file start). The page height is the number of the visible file lines on the screen.

If a *pct* in the value range 33..100 is given, scrolling will occur for this percentage of the visible file line count.

If a negative value *lesscount* is given, the number of lines scrolled will be the visible file line count minus the number of lines specified (absolute value of *lesscount*). However, scrolling will be done at least by a 1/3 screen height.

If the optional parameter `MOVEhere` is specified and the cursor is in the file area when the command is issued, the line under the cursor will be made the current line, with the cursor being moved with this line and staying at the position inside the line.

`Previous [ ` *`count`* ` ]`

Move the current line up by *count* lines (in direction to the file start) if *count* is positive, or down (to file end) if *count* is negative. If no *count* is given, the current line is moved by 1 line.

```
PUT  [ count [ fn [ ft [ fm ] ] ] ]
     [ count [fn[.ft[.fm]]] ]
PPUT [ count [ fn [ ft [ fm ] ] ] ]
     [ count [fn[.ft[.fm]]] ]
```

Write *count* lines beginning with the current line to the specified file. If no *count* is given, one line is written.

The default filetype is EE$BUF. If *fm* is omitted, the filemode A1 is assumed. When no file-id is given, the file PUT EE$BUF A1 is written.

When using PUT for an existing target file, it will be only be overwritten when the filetype and the filemode are defaulted (or the default values EE$BUF A1 are explicitly given). Overwriting an existing file with arbitrary filetype or filemode is forced by using the command PPUT.

```
PUTD  [ count [ fn [ ft [ fm ] ] ] ]
      [ count [fn[.ft[.fm]]] ]
PPUTD  [ count [ fn [ ft [ fm ] ] ] ]
       [ count [fn[.ft[.fm]]] ]
```

Like the PUT resp. PPUT commands, but the lines written are additionally deleted from the file edited if writing the target file was successful.

```
RESet
```

Ignore and remove all prefix commands on the screen, also cancel any pending prefix block command. This command is typically assigned to a PF key.

```
REVSEArchnext
RSEArchnext
-/
```

If the last `Locate` command entered had exactly one target and this target was a search pattern, then the search direction (down or up) is inversed and the previous text search is repeated in the new search direction. The command `REVSEArchnext` is assigned to PF16 per default.

The single token `-/` (without any further characters on the command line) is a shortcut for `REVSEArchnext`. Giving any character (even a blank) after the `-/` will start a new `Locate` with this text as search pattern.

```
SEArchnext
/
```

If the last `Locate` command entered had exactly one target and this target was a search pattern, then the previous text search is repeated in the last search direction (down or up) used. The command `SEArchnext` is assigned to PF04 per default.

The single token `/` (without any further characters on the command line) is a shortcut for `SEArchnext`. Giving any character (even a blank) after the `/` will start a new `Locate` with this text as search pattern.

```
SHIFT  [by]  Left|Right  [ count | :line | .mark ] [ shiftmode ]
```

Shift the content of the current line or a block of lines to the left by removing characters or to the right by inserting spaces at the beginning of the line. The direction of indentation shifting must be explicitly given as parameter.

The amount of the indentation shifting is specified by the optional parameter `by`, defaulted by the value specified with the `SHIFTCONFig` command resp. shifting by 2 characters if no `SHIFTCONFig` command was given.

If more than only the current line is to be shifted, a block of lines starting or ending with the current line can be specified by either with a

- relative line *count*
  the count gives the number of additional lines before (negative value) or after (positive value) the current line are to be shifted; the count 0 shifts only the current line;
- or absolute line number (*:line*) or a line mark (*.mark*)
  the lines between and including the current and the specified line will be shifted.

How shifting is handled if non-space characters would be lost (moved beyond the left or right file border) is controlled by the default behavior defined with the `SHIFTCONFig` command, which can be overridden if the optional `shiftmode` is specified on the `SHIFT` command. See the `SHIFTCONFig` command (in 2.3.6.4) for the possible shift behaviors when shifting beyond a border. If neither a `SHIFTCONFig` command was given nor the `shiftmode` parameter for `SHIFT` is specified, the shiftmode `LIMit` will be used.

`SPLTJoin [ Force ]`

If the cursor is placed in a file line, the line is

- splitted at the cursor position if the cursor is placed before the end of the line, with the new line starting with the character under the cursor; the new line will have the same indentation as the original line;
- joined with the next line if the cursor is placed after the last non-whitespace character of the line, the first non-space character of the next line being placed under the cursor and the gap from the old line end to this position being filled with blanks.
  If the resulting line would be truncated, the join will not happen, unless the parameter `Force` is given.

`TABforward`

If invoked while the cursor is in the file area (usually via the assigned PF-key), the cursor is moved to the next defined tab following the current cursor position. If no tab position is defined on the right of the current position, the cursor is not moved.

If the `TABforward` is invoked while the cursor is in the command line, EE attempts to move the cursor to the file line and cursor position on that line where the cursor was last known to be before it was placed on the command line. If this “last known position” is visible on the current screen, the cursor is placed there. If the screen has been scrolled away from this last known position or this file line has been deleted, then the cursor is placed in the file area on the first column of the current line.

The command `TABforward` is assigned to PF01 per default.

`TABBackward`

If invoked while the cursor is in the file area (usually via the assigned PF-key), the cursor is moved to the next defined tab preceding the current cursor position. If no tab position is defined on the left of the current position, the cursor is moved to the first column (left file border).

The command `TABBackward` is assigned to PF13 per default.

```
TOp
```

Move the current line before the first line of the file.

2.3.6.3 File control commands

```
CASe U | M | R
```

Set the case handling for the file, with the following options:

- U : Uppercase
  → convert all input to the file in upper case, string searches (commands Locate, Change) are case insensitive.
- M : Mixed case
  → do not convert case for text entered to the file, but do case insensitive searches.
- R : Mixed case, respect case for searches
  → do not convert case for text entered to the file, doing exact (case sensitive) searches.

```
EEdit  fn [ ft [ fm ] ]
       fn[.ft[.fm]]
```

Start editing another file in EE. If *ft* or *fm* are omitted, the filetype resp. filemode of the current file are used.

The specified file edited becomes the current file edited. If the file specified is already loaded in EE, this file becomes the current file without re-loading the file content from disk.

```
EXIt
```

Save all modified files (see SAVE) and terminate EE, returning control directly to CMS, even if editing was started from FSLIST or FSVIEW.

```
FILe [ fn [ ft [ fm ] ] ]
     [ fn[.ft[.fm]] ]
FFILe [ fn [ ft [ fm ] ] ]
      [ fn[.ft[.fm]] ]
```

Save the file and terminate editing it (removing from the ring) if the file was successfully saved. If no filename is given, the file is saved with filename originally loaded resp. the last filename previously given to a SAVE/SSAVE command.

If a filename is given and a file with this name exists, using `FILE` will not save the file, but `FFILE` will force overwriting the file with the editor's content.

If the current file is the last file edited, the EE session itself is terminated, returning control to the environment from where the EE session was started, i.e. returning to FSLIST if editing was started from a FSLIST (or FSVIEW) screen or else to CMS.

Writing the file is done in the following steps:

- Write the new file with the filetype EE$TMP.
- If present, rename the current file to the filetype EE$OLD.

- Rename the new file from the filetype `EE$TMP` to the intended filetype.
- Delete the old file with the file type `EE$OLD`.

(should the editor crash, there are chances to find one of the files mentioned to retrieve at least one of the file states)

`FSLIST [ `*`pattern`*` ]`

Open a FSLIST view to browse and view files, allowing to select a file for editing in EE (see 2.4).

`LRECL `*`lrecl`*

Change the logical record length of the file to *lrecl*, but limited to 255, values lower than 1 will be ignored. If the file currently has a larger record length, the file content may be truncated.

```
Quit
QQuit [ ALL ]
```

If prefix commands are present on the current screen or are currently pending from a previous screen: cancel these prefix commands. In this case `[Q]Quit` works as the command `RESet`.

In the normal case with no prefix commands on the screen or pending: terminate editing the current file without writing the file content back to disk. If the file was modified, `Quit` will not be executed. In this case, terminate editing without saving the file can be forced with `QQuit`.

If the current file is the last file edited, the EE session itself is terminated, returning control to the environment from where the EE session was started, i.e. returning to FSLIST if editing was started from a FSLIST (or FSVIEW) screen or else to CMS.

If the parameter `ALL` is given to `QQuit`, all files currently edited are closed without saving and control returns directly to CMS, even if editing was started from FSLIST or FSVIEW.

`RECFM V | F`

Change the record format to variable (`V`) or fixed (`F`) line length.

```
RINGNext
RN
```

Switch to the next file in the ring as current edited file. This command has no effect if only one file currently loaded in EE.

```
RINGPrev
RP
```

Switch to the previous file in the ring as current edited file. This command has no effect if only one file currently loaded in EE.

```
SAVe [ fn [ ft [ fm ] ] ] ]
     [ fn[.ft[.fm]] ]
SSAVe [ fn [ ft [ fm ] ] ] ]
      [ fn[.ft[.fm]] ]
```

Save the current file (but do not terminate editing this file). If no filename is given, the file is saved with filename originally loaded resp. the last filename previously given to a SAVE/SSAVE command.

If a filename is given and a file with this name exists, using SAVE will not save the file, but SSAVE will force overwriting the file with the editor’s content.

(see the command `File` on how the file new file content is created)

`UNBINARY`

When loading the file, non-printable characters (code 0xFF and codes lower than 0x40 = blank) are replaced by a dot character and the file is supposed to be a binary file, preventing it from being saved to avoid data loss.

This command resets the internal binary flag, allowing to overwrite the file.

2.3.6.4 Configuration commands

`ATTR` *`screen-object`* *`color`* `[ HIlight ]`

Define the text rendering attributes for the screen element types of the screen setup. Besides the color, passing `HIlight` allows to specify the highlighted display on monochrome displays. These attributes are used for all functional modes (EE, FSLIST, FSVIEW) of the program.

The following parameters can be given as *screen-object*:

FILe	the file content (not being the current line)
CURRline	the current file line
PREFix	the prefix zone
GAPFill	the gap between the file part and the prefix resp. the screen border
CMDline	the command line input area
CMDARRow	the arrow before the command line area
MSGlines	the messages lines (one line, dynamically up to 3 lines)
INFOlines	the info lines set with command INFOLines
HEADline	the title line (top line) on the screen
FOOTline	the footer line (bottom line) in the screen
SCALEline	the scale line

The following *color* values can be given:
BLUe , REd , PInk , GREen , TURquoise , YELlow , WHIte ,
MOno (default color for color displays)

`CMDLine TOP | BOTtom`

Define the placement of the command line on the screen.

`CURRLine Top | MIddle`

Define the placement of the current line of the file on the screen.

`FTDEFaults` *`ft`* *`recfm`* *`lrecl`* *`casemode`* `[`*`worklrecl`*`]`

Define the default record format and the logical record length to use when creating a new file with the given file type, also defining the default case setting when opening a file with this file type. Optionally the default working LRECL for the filetype can be specified (also used for editing existing files, see also command WORKLRECL).

This command can be used repeatedly, a command for an already defined filetype overwrites the previous settings.

As this information is needed when opening a file, the `FTDEFaults` commands must be given in the profile files for the editor.

The following values can be given:

recfm: V | F

lrecl: 1 .. 255

casemode: U | M | R

worklrecl: 1 .. 255 (with *lrecl* being the effective max. value), default: *lrecl*

`FTTABDEFaults  `*`ft  tabpos1`*` [ `*`tabpos2`*` […] ]`

Define the default tab positions for the specified file type. The tab positions must be integer values in the range 1..255. Up to 16 tab positions can be given. It is not necessary to specify a tab position for column 1, as the line start is an implicit tab position for `TABBackward`.

As this information is used when opening a file, the `FTTABDEFaults` commands must be given in the profile files for the editor. If no `FTTABDEFaults` is present for the file type of a file being opened, then no tabs will be defined for that file.

`GAPFill  NONE | DOT | DASH | CROSS`

Define the character to fill the gap between each input field for a file line and the right border (screen boundary or prefix zone placed on the right).

```
INFOLines  OFf | TOp | BOTtom
INFOLines  CLEAR
INFOLines  ADD infoline-text
```

Control the permanently displayed information lines, usually describing the PF key settings. OFF will remove the infolines from screen, TOP will display the lines in the head area of the screen, BOTTOM will display them on the footer area of the screen.

EE can display up to 2 infolines with max. 80 characters. Defining a new infoline with ADD appends a new infoline, dropping the current first infoline if there already are 2 lines. CLEAR resets the infolines (although not set to OFF, no infoline will be displayed).

`MSGLines  TOp | BOTtom`

Set the placement of the message lines. TOP will display the message lines in the head area of the screen, BOTTOM will display it on the footer area of the screen.

EE displays at least one (possibly empty) message line, dynamically growing up to 3 lines.

`NULls  ON | OFf`

Control how the editable space after the last non-blank character of the file lines is rendered. ON will write NUL characters (default), OFF will use blanks characters to fill the space.

`NUMbers  ON | OFf`

Control how the prefix zone is rendered. ON will display the prefix command zone with the line number of the corresponding line, OFF will display all prefix command zones filled with ‘equal’ characters.

`PF  `*`pfno  command`*`  [ `*`parameters`*` ... ]`
`PF  CLEAR `*`pfno`*

Set or clear the command issued by a PF key. The *pfno* must be a number from 1 to 24, the maximal command length is 120 characters.

`PREFIX  OFf  |  ON  |  LEft  |  RIght`

Control the display and position of the prefix zone. ON or LEFT will place the prefix command zone on the left side of the screen, RIGHT will place it on the right of the screen. OFF will remove the prefix zone from the screen.

`SCALE  OFf | TOp | ABOve | BELow`

Control the display and position of the column scale on the screen. Besides giving an orientation on the column position of the edited text, the scale also indicates the tab positions currently defined for the file with a | (pipe character).

TOP will display it before the first displayed file line, ABOVE will place the scale on the line before the current line, BELOW on the line following the current line. OFF will hide the scale.

`SHIFTCONFig  CHEckall | MINimal | LIMit | TRUNCate  [shiftBy]`

Define the default behavior for the SHIFT command or the shift prefix commands (<, >, <<..<<, >>..>>) when the shifted texts reaches a line border, i.e. if non-space content would be lost by shifting the text to the left beyond the first column or to the right beyond the last column (at LRECL).

The shift commands support the following options when this situation occurs:

- CHEckall
  if text loss would happen for any of the lines to be shifted, the text will not be shifted and the shifting command is aborted.
- MINimal
  the shift-by count applied to all lines will be reduced if necessary to avoid text loss for any of the shifted lines. This ensures that the relative indentation among the lines to be shifted will be preserved.
  This is the default if no SHIFTCONFig command is given.
- LIMit
  each line will be shifted at most until the border is reached, limiting shifting for each line separately.
- TRUNCate
  the lines are shifted by the shift-by count specified, with text shifted beyond the corresponding border being truncated.

The optional second parameter specifies the default shift character count for the shift subcommand and prefix commands. This value must be in the range 1..9.

If not predefined with a `SHIFTCONFig` command, the default shift mode MINimal and the default shift count 2 will be used.

The configured shift defaults can be overridden for each shift (prefix) command by specifying the shift count and shift mode options.

`TABSet [ tabpos1 [ tabpos2 […] ] ]`

Define the tab positions for the current file. The tab position must be integer values in the range 1.255. Up to 16 tab positions can be given. If no tab positions are given, all currently defined tab positions will be removed. It is not necessary to specify a tab position for column 1, as the line start is an implicit tab position for `TABBackward`.

`WORKLrecl length`

Change the working record length for displaying and editing the file content to *length*, but limited to LRECL, values lower than 1 will interpreted as *length* = 1.

Displaying the file content and all editing operations are restricted to the first *length* bytes of a line. Anything line content beyond *length* will be not displayed and will be dropped if the line is modified in any way.

2.3.6.5 Miscellaneous commands

`CMS [ cmscommand ... ]`

Enter CMS Subset or execute a CMS command.

If invoked without a command, EE temporarily leaves the fullscreen mode and the CMS Subset level is entered on the MECAFF console. This allows entering CMS subset compatible commands until the CMS Subset command RETURN is entered, which will leave the CMS subset and restore fullscreen EE editing mode.

If a command is given, EE attempts to execute it. As an invoked program may overwrite the memory used by EE, the first token of *cmscommand* is checked and command names not on a list of explicitly allowed commands will not be executed.
Allowed CMS or CP commands are: `ACCess`, `CLOSE`, `DETACH`, `ERASE`, `LINK`, `Listfile`, `PRint`, `PUnch`, `Query`, `READcard`, `RELease`, `Rename`, `SET`, `STATEw`, `TAPE`, `Type`.

However, if an EXEC with the name of an allowed command is present, this EXEC will be executed, possibly executing programs that overwrite EE's data structures und preventing the proper continuation of EE, leading to the loss of all file changes not saved!

`CLRCMD`

Clear the command line. This command is intended for a PF-Key setting (assigned to PF11 as default).

`Help`

Show the top-level help for EE.

`RECALL`

Place the previous command in the command history. EE has a history of the last 32 commands issued on the command line. This command is intended for a PF-Key setting (assigned to PF12 as default).

2.3.7 Customizing with SYSPROF EE and PROFILE EE

The EE program reads 2 configuration files on startup before the file to be edited is read.

If found in one of the accessed disks, the `SYSPROF  EE` is processed first, then `PROFILE  EE`. For each of these files, the first file found in the search order of the minidisks is used.

However, the file SYSPROF EE is intended to be on a system disk, providing the system-wide defaults (as the FTDEFAULTS and FTTABDEFAULTS for the file types). The file `PROFILE  EE` is intended to provide the user preferences.

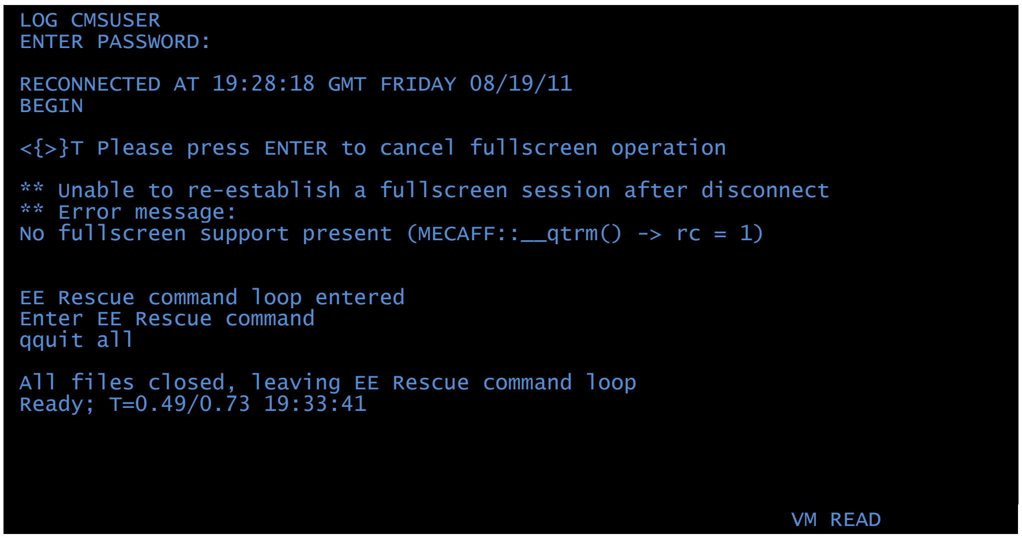
Only configuration commands (see 2.3.6.4) can be used in these files, as no file is opened when they are processed (commands requiring a file are ignored). The commands in the profiles are executed as if they were given in the command area, with the following extensions:

- A star (*) as first non-whitespace character introduces a comment line, the whole line is ignored.
- If a line ends with a backslash as last non-whitespace character, the next line will be concatenated to the line at the position of the backslash (however leading whitespace of the next line is removed). Several lines in the profile can be concatenated up to a total length of 512 characters.

2.3.8 Rescue mode

When the terminal is disconnected while editing files with EE, reconnecting to the VM with a terminal not supporting the same type of fullscreen sessions (with a standard terminal connection after using the MECAFF-version or a 3215-type terminal after using the DIAG58 version of the tools) will prevent EE from re-establishing a fullscreen mode to continue editing.

In this case, EE will enter a command line loop called "rescue mode" after leaving CP (with command BEGIN) and confirming the absence of the expected environment by pressing ENTER, for example:



```
LOG CMSUSER
ENTER PASSWORD:

RECONNECTED AT 19:28:18 GMT FRIDAY 08/19/11
BEGIN

<{>}T Please press ENTER to cancel fullscreen operation

** Unable to re-establish a fullscreen session after disconnect
** Error message:
No fullscreen support present (MECAFF::__qtrm() -> rc = 1)


EE Rescue command loop entered
Enter EE Rescue command
qquit all

All files closed, leaving EE Rescue command loop
Ready; T=0.49/0.73 19:33:41



                                                             VM READ
```

The rescue mode supports the following EE commands: `EXIt`, `FFILe`, `FILe`, `QQuit`, `Quit`, `RINGNext`, `RINGPrev`.

Additionally, the rescue mode command `RINGList` (or RL) lists all files currently in the ring.

2.4 FSLIST – Fullscreen file list

A fullscreen file list can be displayed from CMS with

```
FSLIST file-pattern
```

or

```
EE file-pattern ( FSLIST
```

where *file-pattern* is any file specification allowed for the CMS command LISTFILE (which is in fact called to get the file list). The file pattern can be given with the standard CMS convention (components separated with whitespace) or the dot-notation (see 2.2). The EE editor provides a FSLIST subcommand with the same functionality.

Browsing through the file list and leaving FSLIST is controlled with the PF-keys (summarized on the screen bottom line):

- PF01: *center the cursor's position*
  if the cursor is in the file list, PF01 will attempt to (vertically) center the character under the cursor on the screen;
  if the cursor is on the command line, the cursor will be placed in the list as near as possible to the middle of the screen.
- PF02: *EE*
  If the cursor is placed on a line in the file list: open this file with EE for editing.
- PF03: *Quit*
  Terminate FSLIST.
- PF04: *search next*
  Repeat the last search in the search direction last used.
- PF05: *Top* ; PF06: *Full page up* ; PF07: *⅔ page up*
  Scroll through the list in direction of the list begin;
  if the cursor is in the file area, PF07 attempts to vertically center the line under the cursor on the screen.
- PF08: *⅔ page down* ; PF09: *Full page down* ; PF10: *Bottom*
  Scroll through the list in direction of the list end;
  if the cursor is in the file area, PF08 attempts to vertically center the line under the cursor on the screen.
- PF12: *FSVIEW*
  If the cursor is placed on a line in the file list: open this file with FSVIEW for browsing.
- PF15: *Quit*
  Terminate FSLIST.
- PF16: *reverse and search next*
  Reverse the search direction and repeat the last search in the new search direction.

The following commands are accepted on the command prompt of FSLIST:

`Listfile` *`file-pattern`*

Change the search pattern for the file list. If a sort command was given before, the file list will sorted as specified previously.

`Sort` *`column-spec`* `[ [` *`column-spec`* `] ... ]`
`Sort OFf`

Sort the file list by the specified columns of the list or switch off sorting if `OFF` is given. A *column-spec* consists of one of the after-mentioned keywords, optionally preceded by the sort order character, with `-` for descending order and + for ascending order (and ascending order being the default), for example: `sort -ts  +type`

The following single and combined columns of the file list can be used to sort the list:

`NAme`	→ filename
`TYpe`	→ filetype
`MOde`	→ filemode
`RECFm`	→ record format
`LRecl`	→ logical record length
`Format`	→record format and record length
`RECS`	→ record count
`BLocks`	→ block count
`DAte`	→ file write date
`TIme`	→file write time
`TS`	→ file write timestamp (date and time)
`LAbel`	→disk label

`/text/`

Search forward for *`text`* in the current help page

`-/text/`

Search backward for *`text`* in the current help page

`/`

Repeat the last search in the search direction last used (also PF04)

`-/`

Reverse the search direction and repeat the last search in the new search direction (also PF16)

`BOTtom`

Scroll to the top of the list.

`TOp`

Scroll to the bottom of the list.

`Quit`

Leave FSLIST (also PF03)

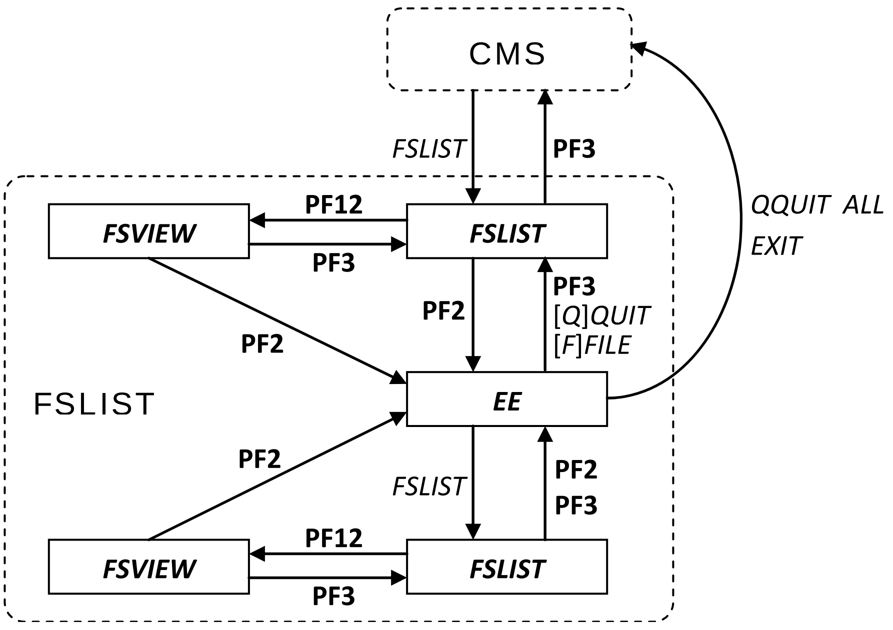
`Help`

Show the help for FSLIST.

Browsing or editing the content of a file in the list can be started by moving the cursor to the line with the filename and

- pressing PF02 to edit the file with EE,
- pressing PF12 to view the file content with FSVIEW.

The following diagram shows the transitions between the modes available from FSLIST, also showing with the commands (*CMD*) or PF keys (**PFxx**) to change from mode to mode:



2.5 FSVIEW – Fullscreen file browser

A CMS can be browsed from CMS with FSVIEW with

```
FSVIEW fn fm [ ft ]
       fn.ft[.fm]
```

or

```
EE fn fm [ ft ] ( FSVIEW
   fn.ft[.fm] ( FSVIEW
```

If a file loaded into FSVIEW contains binary characters (codes 0x00 to 0x3F or 0xFF), these characters are replaced with a dot ('.') in the file buffer.

Browsing through the file content and leaving FSVIEW is controlled with the PF-keys (summarized on the screen bottom line):

- PF01: *center the cursor's position*
  if the cursor is in the file area, PF01 will attempt to center the character under the cursor on the screen (scrolling both vertically and horizontally);
  if the cursor is on the command line, the cursor will be placed in the file area as near as possible to the middle of the screen.
- PF02: *EE*
  Open the file displayed in FSVIEW with EE for editing (FSVIEW is fact replaced with EE, so leaving EE will not return to FSVIEW).
- PF03: *Quit*
  Terminate FSVIEW.
- PF04: *search next*
  Repeat the last search in the search direction last used.
- PF05: *Top* ; PF06: *Full page up* ; PF07: *⅔ page up*
  Scroll through the file in direction of the file begin;
  if the cursor is in the file area, PF07 attempts to vertically center the line under the cursor on the screen.

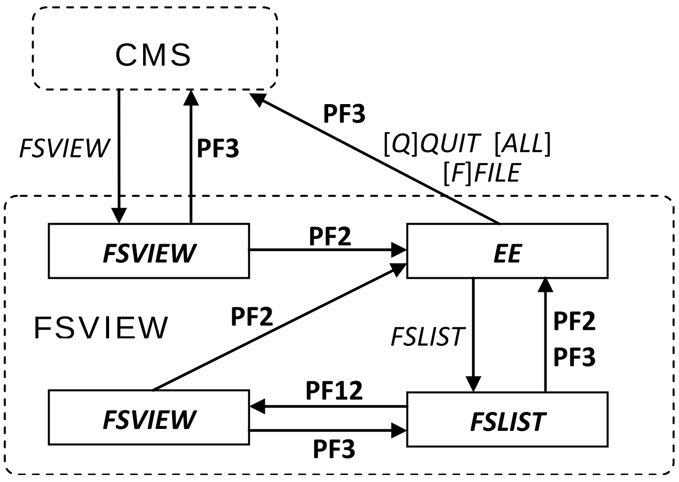
- PF08: *⅔ page down* ; PF09: *Full page down* ; PF10: *Bottom*
  Scroll through the file in direction of the file end;
  if the cursor is in the file area, PF08 attempts to vertically center the line under the cursor on the screen.
- PF11: *Scroll left by 20 columns*
  Shift the visible horizontal portion of the file to the left, i.e. move the first displayed column of the file by 20 positions to the left, up to the first column of the file;
  if the cursor is in the file area, PF11 will attempt to center the cursor's column horizontally.
- PF12: *Scroll right by 20 columns*
  Shift the visible horizontal portion of the file to the right, i.e. move the first displayed column of the file by 20 positions to the right, up to the last column of the file being displayed at the border of the screen;
  if the cursor is in the file area, PF12 will attempt to center the cursor's column horizontally.
- PF15: *Quit*
  Terminate FSVIEW.
- PF16: *reverse and search next*
  Reverse the search direction and repeat the last search in the new search direction.
- PF23: *Scroll left by 10 columns*
  Shift the visible horizontal portion of the file to the left, i.e. move the first displayed column of the file by 10 positions to the left, up to the first column of the file.
- PF24: *Scroll right by 10 columns t*
  Shift the visible horizontal portion of the file to the right, i.e. move the first displayed column of the file by 10 positions to the right, up to the last column of the file being displayed at the border of the screen.

The subcommands `TOp` and `BOTtom` scroll the file to show the first resp. the last line in the screen.

Additionally, text can be searched in the file using the subcommands `/text/` (search forward) and `-/text/` (search backward). As for FSLIST, the last search can be repeated in the last direction with the subcommand `/` (without pattern, also PF04) or with reversed search direction with subcommand `-/` (without pattern, also PF16).

The subcommand `Help` shows the help topic for FSVIEW.

The following diagram shows the transitions between the modes available from FSVIEW, also showing with the commands (*CMD*) or PF keys (**PFxx**) to change from mode to mode:



2.6 FSHELP – Fullscreen help

The fullscreen display of a standard HELP topic is invoked from CMS with

```
FSHELP topic
```

This looks for a file `topic HELPxxx` on disk U (with `xxx` one of CMD, DBG, EE, EDT, EXC or REX) and displays it. If the additional help file exists (`topic HELPxxx2 U`), it is appended to the displayed content.

FSHELP allows to navigate to other help topics either with the subcommand `Help topic` or by placing the cursor in a word of the displayed help text and pressing the PF01 key. The topics opened are stacked in a last-in-first-out manner: closing the current topic with PF03 displays the previous one and so on until closing the first (initial) topic terminates FSHELP, returning to the CMS prompt.

When invoked without a topic, FSHELP looks for a file MENU FSHELP on all accessed disks and displays this file as top level menu. If this file is not found, the help menu is build based on the help files on disk U (with filetype HELPxxx, see above), grouping the topics by the command types CMS, CP, EE, EDIT, DEBUG, EXEC. This help file is automatically saved in the file MENU FSHELP A2. If necessary (for example to include new help topics added to disk U), this menu help file can be regenerated on disk A by invoking FSHELP with only the option REBUILD:

```
FSHELP ( REBUILD
```

Browsing through the help information, navigating to other topics and leaving FSHELP is controlled with the PF-keys (summarized the screen bottom line):

- PF01: *Goto*
  Navigate to the topic identified by the word where the cursor is in the help text. If no such topic exists, an error message is displayed and the cursor stays at the current place in the help text.
- PF02: *Goto*
  (like PF01)
- PF03: *Back*
  Close the current topic and display its predecessor in the help topic stack. If the initial topic is currently displayed, FSHELP is terminated.

- PF04: *search next*
  Repeat the last search in the search direction last used.
- PF05: *Top* ; PF06: *Full page up* ; PF07: *⅔ page up*
  Browse through the current topic in direction of the help text begin.
- PF08: *⅔ page down* ; PF09: *Full page down* ; PF10: *Bottom*
  Browse through the current topic in direction of the help text end.
- PF12: *Goto*
  (like PF01)
- PF15: *Quit*
  Terminate FSHELP.
- PF16: *reverse and search next*
  Reverse the search direction and repeat the last search in the new search direction.

The following commands are supported on the subcommand program prompt of FSHELP:

`Help` *`topic`*
: Navigate to the specified help topic, pushing the current topic onto the stack.

`/text/`
: Search forward for *`text`* in the current help page

`-/text/`
: Search backward for *`text`* in the current help page

`/`
: Repeat the last search in the search direction last used

`-/`
: Reverse the search direction and repeat the last search in the new search direction

`Back`
: Close the current topic and display its predecessor in the help topic stack. If the initial topic is currently displayed, FSHELP is terminated.

`BOTtom`
: Scroll to the top of the help topic.

`TOp`
: Scroll to the bottom of the help topic.

`Quit`
: Terminate FSHELP

2.7 IND$FILE – File transfer

The command IND$FILE implements the host side of the IND$FILE 3270 file transfer protocol using the CUT-mode. It allows file transfer for VM/370:

- as upload (terminal → host) and download (host → terminal)
- for files with LRECL up to 255
- in ASCII or in binary mode

The IND$FILE command is invoked automatically by the 3270 terminal emulation when file transfer is initiated through the means of the emulation (local command or GUI dialog). It supports the command line parameters required to allow to above transfer options.

MECAFF's IND$FILE was tested against the WC3270 and VISTA TN3270 (Trial Copy) terminal emulators. IND$FILE also works with the PC3270 terminal emulator, provided the CUT-mode file transfer is activated by adding the line:
`CUTprotocol=Y`
to the `[Transfer]` section in the `.ws` profile configuration file of PC3270..

There is currently no experience with other emulators allowing file transfers based on the IND$FILE protocol.

To perform file transfers, it is required that the CMS environment is able to execute the IND$FILE command, i.e. no other CMS and no other fullscreen (EE, FSHELP etc.) is currently active and the console is not in the *More*… state.

When transferring data in ASCII mode, character mapping is an important issue as several consecutive character set mappings are involved at different levels of the complete host+terminal environment. As different character sets (and corresponding translations) may be involved when displaying a file on the terminal's screen and when transferring the same file, the transferred file contents on the PC side and the original content host side may differ.

The initial character mapping setup of IND$FILE is based on the defaults

- of the SixPack-configuration (option CODEPAGE 819/1047)
- and the open-source WC3270 terminal emulation (Bracket = charset 037)

To support differing character sets, IND$FILE allows to specify remapping tables for data transfer in ASCII mode. Remapping tables are files with the filetype IND$MAP. IND$FILE can optionally use up to 2 remapping files, which are looked for in the search order of the accessed minidisks:

- `DEFAULT IND$MAP *`
  this file gives the default mappings; using this file is useful if the terminal emulation does not allow to specify arbitrary command line options for the host IND$FILE command
- `*fn* IND$MAP *`
  with *fn* given by the command line option
  ```
      MAP fn
  ```
  for the IND$FILE command; the mappings in this file are applied after those in DEFAULT IND$MAP (if available)

Remapping is defined as additional character translation applied onto the *direct mapping* before the transfer host→terminal resp. after the transfer terminal→host. The *direct mapping* specifies the character mapping that occurs through the participating components and the characters set used there.

This direct mapping can be made visible by transferring the file EBCDIC MEMO delivered with MECAFF to the PC. This file contains all displayable EBCDIC characters spanned in a hexadecimal matrix. Transferring this file to the PC through the terminal emulation with IND$FILE (and no additional remapping of course) will show which EBCDIC code positions directly result in which characters on the PC side.

A character position showing a different visible representation on the terminal and the PC file can be considered a “wrong” mapping in this file transfer scenario. Wrong mappings can be corrected through remapping with IND$MAP file(s) specific to the involved components and charsets.

IND$MAP files have the following format:

- blank lines and lines starting with a * are ignored
- a single mapping is defined by a line with 2 hexadecimal codes
    - the first code defines the visible EBCDIC character on the host side (as displayed on the 3270 terminal) which is wrongly mapped;
      this hexadecimal code can be determined using the EBCDIC MEMO file displayed on the host side with the 3270 terminal emulation;
    - the second code defines the EBCDIC character position which results in the expected visible character on the PC side (i.e. the position of the direct mapping giving the expected character on the PC);
      this hexadecimal code can be determined by locating the wrongly mapped character in the transferred EBCDIC MEMO opened on the PC side.

If necessary, the following step can be taken to create an IND$MAP file for a given 3270 terminal emulator:

- download the file EBCDIC MEMO of the MECAFF distribution with IND$FILE in ASCII mode without a mapping file using the terminal emulation configured with the settings (character sets or the like) that will be used subsequently when ASCII data transfers will be done.
- compare both files and identify character codes wrongly mapped;
  for this, open the file EBCDIC MEMO with FSVIEW using the terminal emulation on the CMS side and the downloaded file with a text editor (for example NOTEPAD in a Win32 platform) on the outside of VM/370:
    - in a correct mapping, the displayed characters at a given byte code point (row and column of the file) are the same
    - for each byte code position having a different displayed character, find the expected character in the downloaded file and add a new line to the IND$FILE with:
        - the first hexadecimal code being the byte code of the character wrongly mapped (the code can identified from the row/column in the file EBCDIC MEMO)
        - the second hexadecimal code being the byte code of the located display-character in the downloaded file.

The file `VISTA IND$MAP` delivered with MECAFF is an example of a mapping file, correcting the charset translations for the *VISTA TN3270 Trial Copy* emulator with all default settings:

- SixPack configuration: CODEPAGE 819/1047
- VISTA TN3270:
  - *Windows (ANSI 1252)* (for ‘File Transfer Setting’)
  - *United States* (for ‘Screen Code Page’)

3 Known restrictions / problems

3.1 EE – Implementation limitations

EE currently has no checks for running out of memory, so editing very large files may lead to ABENDs and the loss of file modifications. Assuming 14M of available memory (for a 15M VM) and an edited file with LRECL 255 (the max. value supported by EE), this allows roughly for 50.000 lines (including internal overhead).

EE is implemented in C using the native C library (GCCLIB). Probably due to misunderstandings of (or wrongly using) the native C API resp. to limitations of the underlying CMS, the following problems occur in the current implementation for saving files with variable record length (RECFM V):

- Writing an empty line (line length = 0) with `CMSwriteFile` results in return code 8 and no line is written, leading to a file where all empty lines disappeared (if rc 8 is ignored).
- Trying to write a file with a given LRECL but with its widest line not reaching this LRECL, the value passed to `CMSfileOpen` as buffer length is not used as LRECL for the file written, ending with the LRECL being the length of the widest line written to the file (e.g. LRECL is to be 80 but the widest line is only 60 chars long, the LRECL of the file will be 60 instead of the intended 80).

A similar problem to empty lines with RECFM V exists for empty files (for both RECFMs), as "not writing a line into a new file between open and close" results in "no file on minidisk".

The current implementation in EE copes with these problems in the following way:

- Empty lines in the edited file with RECFM V are written with a single blank to the file. This should not be a problem for plain text files.
- Saving an empty file writes a single empty line.
- When opening a file with RECFM V, the file width used for editing is the maximum of the LRECL of the file and the LRECL value of the FTDEFAULTS command for the filetype. If no matching FTDEFAULTS is given, the terminal width (minus 7 characters for prefix zone including control positions) is taken as potential line width.

3.2 IND$FILE

As no formal specification of the IND$FILE transfer protocol was found (neither for the CUT-mode nor the DFT-mode), the MECAFF version of IND$FILE was implemented for the CUT-mode transmission based

- on the open source WC3270 terminal emulation sources,
- on the analysis of the 3270 data streams as used by the MVS version as well as the GCIC version of IND$FILE (using the `trace data` feature of WC3270)
- and some information found in the Internet, mainly IND$FILE Technical Reference (http://www3.rocketsoftware.com/bluezone/help/v50/en/bz/DISPLAY/IND$FILE/IND$FILE_Technical_Reference.htm).

So it cannot be ensured that MECAFF's IND$FILE will have a protocol conforming behavior, specifically in "non-straight-forward" situations.

Furthermore, MECAFF's IND$FILE was tested against the *WC3270*, *VISTA TN3270 (Trial Copy)* and *PC3270* terminal emulators. There is currently no experience with other emulators allowing file transfers based on the IND$FILE protocol.

Regarding WC3270, using version 3.3.10ga5 sometimes duplicates data blocks transmitted into a downloaded file (host → terminal), leading to varying (PC-)file lengths when downloading the same host file. This problem does not occur when using a current version (3.3.12ga7 or newer) of WC3270.

Limitations of MECAFF's IND$FILE in ASCII transfer mode:

- TAB-characters are currently mapped to a single blank;
- non-displayable characters (0x00..0x3F and 0xFF) are transmitted as dot-character (".");
- double-byte character sets (DBCS) are not supported.

4 Possible extensions and improvements

The MECAFF package is work in progress. The following are just ideas on what could be done, with no particular priority…

4.1 EE

- Scripting/macro capabilities (control structures and programmability capabilities for command files), possibly using BREXX (if it can be figured out on how to interface it for passing variable values, invoking REXX scripts and invoking EE commands from scripts)

4.2 FSLIST

- Make PF keys configurable
- Extend the file list to allow specifying arbitrary commands to be applied to the files. This requires to find a way to invoke CMS user area commands (like COPYFILE) from a program (EE/FSLIST) already running in user area memory.

4.3 FSVIEW

- Make PF keys configurable