

MECAFF

Multiline

External

Console

And

Fullscreen

Facility

for VM/370 R6 SixPack 1.2

Version 0.9.0 (beta)

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	Introduction.....	3
1.1	What is MECAFF?	3
1.2	Features.....	3
1.2.1	MECAFF console	3
1.2.2	Supported 3270 terminals.....	4
1.2.3	Connections from MECAFF to the VM/370 machine	5
1.3	Tested environments.....	5
2	Installation.....	6
2.1	Prerequisites.....	6
2.2	Files in the package	6
2.2.1	Running the external MECAFF process	7
2.2.2	CMS.....	8
3	The MECAFF console	9
4	The MECAFF tools for CMS.....	11
4.1	FS-QRY – Query Informations.....	11
4.2	EE – Fullscreen editor	12
4.2.1	Introduction.....	12
4.2.2	Prefix commands	13
4.2.3	Commands.....	13
4.2.4	Customizing with SYSPROF EE and PROFILE EE	19
4.3	FSHELP – Fullscreen help.....	19
4.4	FSLIST – Fullscreen file list	19
5	MECAFF-API.....	20
5.1	Query terminal information	20
5.2	Full screen write	22
5.3	Full screen read	22
5.4	API files and linking.....	23
6	Known restrictions / problems.....	24
6.1	General	24
6.2	MECAFF console	24
6.3	EE – Implementation limitations	25
7	Planned extensions and improvements	26
7.1	MECAFF console	26
7.2	EE	26

1 Introduction

1.1 What is MECAFF?

MECAFF is a non-invasive extension to VM/370 R6 providing full screen capabilities and a serial-style (instead of page-oriented) console interface with 3270 terminal emulators.

More precisely, MECAFF supports VM/370 and VM380 SixPack version 1.2 running inside a Hercules S/370 resp. S/380 emulator (the term “VM/370” will be used here for both VM/370 and VM/380).

The term “non-invasive” means that no modifications are required in the VM/370 system. The new fullscreen capability resp. the modified console interface are provided by

- an external program working between the 3270 terminals and the VM/370 host.
The 3270 terminals connect to MECAFF, which opens a connection to the VM/370 machine, providing each session with the new console (with a scrollable output line history and a command input history) and the ability to share the connected 3270 screen with (fullscreen) applications on the host.
- a small API allowing CMS programs to access the fullscreen facility of MECAFF.
This API communicates with MECAFF through the terminal connection using (hopefully) rarely used character sequences to do both the handshaking with MECAFF to share the 3270 terminal and the encoding the 3270 data streams in both directions.
The API provides 3 basic operations:
 - query terminal properties
 - send a 3270 output stream
 - receive a 3270 input stream after waiting for user input

Based on the MECAFF-API, a few CMS programs are provided with MECAFF:

- FS-QRY : display characteristics of the 3270 terminal connected to MECAFF
- FSHELP : a simple fullscreen help (displaying the standard CMS help)
- FSLIST : a simple fullscreen file list with file browser
- EE : a fullscreen file editor (basic editing functionality)

Non-invasive also means that even if all MECAFF elements are installed, the VM/370 system can still be used as if they weren’t, simply by connecting the 3270 terminal emulator directly to Hercules machine instead of using MECAFF.

The external MECAFF process is implemented as a stand-alone Java™ program. The CMS components (MECAFF-API and client programs) are mainly implemented in C with the native C runtime library GCCLIB and a small assembly routine.

1.2 Features

1.2.1 MECAFF console

The most notable difference between the standard VM/370 console (for 3270 terminals) and the MECAFF console is that – although working with a 3270 terminal – the MECAFF console simulates a line oriented terminal using a scrolling metaphor. Up to 65536 output lines are remembered, allowing to page through the output history of the current session.

MECAFF has a similar screen layout to the VM/370 console for 3270 terminals with an input zone at the bottom of the screen, the output zone filling the screen space above the input zone and a VM status indicator (where MECAFF uses mixed-case state names for the equivalent VM states: “Running”, “VM read” etc), for example:

```
|      welcome to VM/370 and VM/380 "SixPack" version 1.2!      |
+-----+
For a list of CMS commands, type HELP CMSCMDS.  For a list of CP commands, type
HELP CPCMDS.

Other useful documentation and sample programs can be found on MAINT 19D,
accessed as your U disk.

For more details, type HELP WELCOME ( MORE

For information on building the CP or CMS nucleus, read SYSPROG MEMO.
Ready; T=0.02/0.07 20:35:45

dir * direct
Filename Filetype Fm Format Recs Blocks Date Time Label
DOSVS DIRECT A1 F 80 12 2 02/18/06 21:07 MNT191
SIXPACK DIRECT A1 F 80 326 33 09/30/10 21:19 MNT191
VM50 DIRECT A1 F 80 301 31 05/03/06 2:16 MNT191
3 files, 66 blocks: 52,800 bytes.
Ready; T=0.01/0.02 20:35:52
Running >> q disk
```

However, the MECAFF console takes advantage of the 3270 terminal characteristics like colors and larger screen sizes (see 1.2.2).

Like the VM/370 console, the MECAFF console must have a “More...” status handling to give the user the opportunity to read and acknowledge chunks of output lines (instead of seeing a fast scrolling output). MECAFF takes a more *X-like approach where the output is held with the “More...” prompt after an output zone height has been written since the last user input. So commands writing out only a few lines (less than an output zone height) or programs doing question-response cycles with a few output lines between prompts will not enter the “More...” state. Releasing the “More...” state to get the next pending lines written to the screen is simply done with the ENTER key.

And of course the MECAFF console is built to share the 3270 screen with a CMS fullscreen program based on the MECAFF-API.

1.2.2 Supported 3270 terminals

MECAFF supports the following TN3270 terminals types connecting as clients with the mentioned screen geometries:

- IBM-3278-2-E, IBM-3279-2-E
→ 80 cols x 24 lines
- IBM-3278-3-E, IBM-3279-3-E
→ 80 cols x 32 lines
- IBM-3278-4-E, IBM-3279-4-E
→ 80 cols x 43 lines
- IBM-3278-5-E, IBM-3279-5-E
→ 132cols x 27 rows
- IBM-DYNAMIC
→ arbitrary screen geometry within the 14-bit buffer addressing range with at least 24 rows

and 80 columns (theoretically allowing screen sizes between 80 cols x 204 rows and 682 cols x 24 rows)

If not disabled, the terminal capabilities (alternative buffer size, support for color and extended highlighting) are determined from the terminal using a 3270 structured field query, even for standard types (allowing for example to recognize “Custom geometry” settings for the x3270 terminal emulation).

If querying the terminal capabilities is disabled used the “-nodynamic” switch (see 2.2.1), only the predefined terminal types (all mentioned above except IBM-DYNAMIC) with their standard characteristics are known.

1.2.3 Connections from MECAFF to the VM/370 machine

For each 3270 terminal connecting to the MECAFF process, a dedicated TCP/IP connection is opened to the Hercules process hosting VM/370, presenting itself either as a 3270 terminal (more precisely a minimal IBM-3278-2 emulation) or as a plain telnet client (seen by VM/370 as 3215 terminal).

Depending on the terminal type used by MECAFF, CP uses 2 different console flavors to manage console I/O, which are both mapped to the MECAFF console presentation and interaction. The device type for the console is identified by a CP QUERY VIRTUAL command either as GRAF for a 3270 terminal or CONS for a 3215 terminal. Inspired from these device names, the term “GRAF-style” is used for MECAFF connecting as 3270 and the term “CONS-style” is used for a 3215 type connection.

Which type of connection to VM/370 is to be used by MECAFF is determined by the TCP/IP port to which the 3270 terminal opens the connection to MECAFF:

- Connecting to port 3277 will use a GRAF-style connection to VM/370.
- Connecting to port 3215 will use a CONS-style connection to VM/370.

(the given port numbers can be overridden using command line parameters for MECAFF, see 2.2.1)

Both connection styles behave the same for fullscreen applications, as it is the program based on the MECAFF-API which interacts transparently with the 3270 terminal connected to MECAFF.

On the console interaction level (command → responses), MECAFF cannot completely hide the differences on how CP handles page-oriented (3270 = GRAF) and serial (3215 = CONS) terminal types (see 6.2).

1.3 Tested environments

MECAFF has been tested in the following environments:

- Windows Vista 32-Bit
Hercules 3.07 32-Bit
Java 1.6.0-13 32-Bit
VM/370 SixPack 1.2
wc3270-3.3.0
- Windows 7 Professional 64-Bit
Hercules 3.07:380-1.0 64-Bit
Java 1.6.0-26-b03 64-Bit
VM/380 SixPack 1.2
wc3270-3.3.0

- Ubuntu Linux 9.10
Hercules 3.07 64-Bit
Java 1.6.0-26 64-Bit
VM/370 SixPack 1.2
x3270-3.3.4p6

2 Installation

2.1 Prerequisites

- The external MECAFF process is implemented as Java program and requires a Java runtime Environment (JRE) Version 1.6.
- For GRAF-style, MECAFF emulates a very basic 3278-2 terminal type. So a sufficient number of 3278-2 compatible terminal lines need to be configured both in the Hercules configuration (.conf-file) and the VM/370 hardware configuration (DMKRIO). Moreover, the lines configured for Hercules may not have a group name, as MECAFF does currently not support this functionality.

However, in the default SixPack 1.2 configuration, the configuration line

```
00C0.32 3270
```

already defines 32 lines suitable for MECAFF.

- For CONS-style, serial terminal lines without the NOPROMPT option are needed in the the Hercules configuration (.conf-file) and the VM/370 hardware configuration (DMKRIO) to allow MECAFF to connect. The standard SixPack 1.2 configuration for Hercules does not provide any free lines of this type (line 0009 is available, but reserved for the integrated console). Based on the DMKRIO configuration, the devices 0004 to 0008 and line 000A can be used for CONS-style by adding the following lines to the Hercules configuration:

```
0004.5 3215
000A 3215
```

- MECAFF scans the terminal data stream coming in from CP (VM state, password prompts, ...) and Hercules (prompt string for 3215 lines) for characteristic texts to manage the own state (prompt state, session end, ...). If these texts are modified, MECAFF must possibly be adapted.
- MECAFF relies on the standard definition of the line end character (character #, see command CP TERMINAL LINEND) when sending the HT or HX commands to leave the "More..." state prematurely.

2.2 Files in the package

Unpacking the MECAFF-archive copy the following files to the current directory:

- mecaff.jar
→ the runnable jar file for the MECAFF process
- mecaff.aws
→ the AWS tape file in CMS TAPE format containing the MECAFF CMS tools and API
- sixpack+mecaff.conf
→ a sample Hercules configuration for SixPack 1.2; this configuration is the original file from the the SixPack 1.2 final release, modified to provide additional 3215 devices for CONS-style connections (and correcting directory names to lowercase for *X-OSes).

- `sixpack+mecaff.cmd`
→ startup script for Win32 intended for a standard (fresh) SixPack 1.2 installation, using `sixpack+mecaff.conf` as configuration. This script start the following components in parallel:
 - the MECAFF process
 - the Hercules-based VM/370 machine
 - a 87x32 color 3270 terminal connected to VM/370 via MECAFF using GRAF-style
- `run3270-delayed.cmd`
→ script used by `sixpack+mecaff.cmd` to start the WC3270 emulator (which is part of SixPack 1.2)
- `sixpack+mecaff.sh`
→ startup shell script for *X-OSes equivalent to `sixpack+mecaff.cmd`, but using x3270 as terminal emulation (it is assumed that x3270 is installed and available on the PATH)
- `MECAFF-Manual-version.pdf`
this file.

Although MECAFF is non-invasive, it is probably not a good idea to do the first experiments with MECAFF in the own “productive” environment. This is why the `sixpack+mecaff`-scripts allow to make the first steps with MECAFF on a fresh SixPack 1.2 installation, the only requirement being that both Hercules and a Java Runtime 1.6 are on the PATH for all users (as well as x3270 on *X). When the Hercules machine started by a `sixpack+mecaff`-script ends, the startup script automatically terminates the MECAFF process and the 3270 emulation (in fact all running 3270 emulators).

The delivered scripts can then be used as starting point to extend the own VM/370 environment.

The only “trick” to use MECAFF is to run it in parallel to the VM/370 Hercules machine and that the 3270 terminal emulators connect to this process instead of directly to the Hercules.

2.2.1 Running the external MECAFF process

The MECAFF process is implemented as Java program and needs a Java Runtime Version 1.6.

It is bundled as runnable JAR file, so it can be started with the following command line:

```
java -jar mecaff.jar [ options ]
```

The MECAFF program accepts the following case-insensitive options (parameters to options may not be separated from the option):

- `-h`
write out usage information and terminate.
- `-vmHostName:hostname`
Specifies the host on which the VM/370 (or VM/380) machine is located.
Default: localhost
- `-vmHostPort:port`
TCP/IP port to connect to on *hostname*, this is the CNLSPORT parameter of the Hercules configuration.
Default: 3270

- `-portGRAF:port`
TCP/IP listen port to open for GRAF-style connections.
Default: 3277
- `-portCONS:port`
TCP/IP listen port to open for CONS-style connections.
Default: 3215
- `-do:style`
Specifies which port(s) to open (which connection style(s) to support, possible values for *style* are:

CONS	→	open only the CONS-style port
GRAF	→	open only the GRAF-style port
BOTH	→	open both ports

 Default: BOTH
- `-noDYNAMIC`
Disable querying the terminal characteristics through a WSF-query, so only the predefined terminal types (all mentioned in 1.2.2 except IBM-DYNAMIC) with their standard characteristics are known.
Default: querying is enabled.
- `-dumpParms`
Dump communication parameters on startup.

Example:

```
java -jar mecaff.jar -portgraf:8000 -portcons:8001
```

As soon as the MECAFF process and of course the VM/370 machine are started, 3270 terminal emulations can be connected to MECAFF through one its listen ports.

However, when using CONS-style, the VM/370 startup should be completely done (i.e. `/enable all` has been executed), as having MECAFF connecting as a serial terminal before the operator's console is connected to the Hercules window (through device 0009) may lead to conflicting accesses to the device 0009, resulting at least in repeating error messages on the Hercules console.

2.2.2 CMS

The AWS tape file `mecaff.aws` is written with the CMS TAPE utility and contains the MECAFF tools for CMS (see 4) up to the first tape mark and the MECAFF-API files (see 5.4) up to the tape end.

The tape has the following content:

SAMPLE	EE	F2
SYSPROF	EE	F2
FS-QRY	MODULE	F2
EE	MODULE	F2
FSLIST	MODULE	F2
FSHELP	MODULE	F2

tape-mark

FSIO	H	F2
BOOL	H	F2
FSIO	TEXT	F2
WR3270	TEXT	F2

tape-mark
tape-mark

The files can be loaded to a private minidisk (for example CMSUSERS disk F) for local tests.

To make the tools available to all users, disk Y (19E) should be a good target for installing the MECAFF files. Loading the MECAFF-API files after the first tape mark is only necessary if own fullscreen applications will be programmed.

To install the MECAFF file on the Y disk:

- On the Hercules console enter:
`devinit 480 mecaff.aws`
- Logon as MAINT
- Access the Y disk in R/W mode:
`release y`
`access 19E y`
- Attach the tape device and load the files to disk Y:
`attach 480 to maint 181`
`tape load * * y`
`tape load * * y`
- Re-access disk Y in R/O mode:
`release y`
`access 19E y/s`

Users logging on after the files have been installed on disk Y will be able to use the MECAFF tools.

The file `SAMPLE EE` is a sample EE profile, which can be copied to `PROFILE EE A` and adapted to the own needs (see 4.2.4).

Remark: disk A of user MAINT has a `SYSPROF EXEC` hides the system wide one and prevents loading the native GCC runtime into resident memory, which is required by the MECAFF tools. Before the MECAFF tools can be used by MAINT, the local `SYSPROF EXEC` has to be disabled by renaming it away (and logging off and on again), for example:

```
rename sysprof exec a sysprof_ exec a
```

3 The MECAFF console

Like the standard VM/370 console handler for 3270 terminals, the MECAFF console subdivides the screen in an input area on the bottom of the screen (one or two lines high, depending on the screen geometry), leaving the larger upper area of the screen for the output area (see 1.2.1).

The input area has on the left a prompt indicating the current state of VM equivalent to indication on the lower right of the standard VM/370 screen layout for 3270. The input field is 130 characters long and switches to invisible text when a password prompt is recognized by MECAFF.

Input issued on the command line (except password input) is saved in a local history and can be cyclically recalled for editing and re-issuing:

- PF12 steps back in input history, the retrieved command replaces the current command line content.
- PF11 steps forward in the input history.
- PF03 clears the input area and resets the internal history pointer, so the next PF12 will again retrieve the last command entered.

Data entered on the command line is sent to VM by pressing the ENTER key. Pressing the PA1 key will enter CP (in GRAF-style, a PA1-Aid is sent to VM; in CONS-style, a `#CP` command is sent).

On the output area, the MECAFF console tries to simulate a line oriented terminal on the basically page-oriented 3270 device. Instead of the standard CP approach of filling the screen and ask the user to clear the screen when it is full, the MECAFF console emulates a scrolling behavior on the output area:

- When initially starting with a blank screen, incoming lines are written from top to bottom until the last line of the output area is written.
- As new lines are written, the current screen content is scrolled up.
- Lines moved out of the screen do not vanish, the MECAFF console remembers up to 65536 output lines, allowing to scroll back and forth using the traditional PF07 / PF08 keys for page up and down and additionally with PF06 / PF09 to jump to the top resp. bottom of the line buffer. If the output zone was paged up on the output history, entering a command will jump back to the last page of output history when the ENTER key is pressed.
- After writing out a screen height of output lines since the last user input at the command prompt, the MECAFF console will enter the “More...” state, waiting for the user to press the ENTER key to acknowledge the current screen content and to allow further output. Leaving the “More...” status is an explicit user action, there is no timeout.
- While the MECAFF console is in the “More...” status, the user has the following options:
 - Page through the output history with the PF06, PF07, PF08, PF09 keys.
 - Prepare the next command to enter, possibly using the PF03/PF11/PF12 to access the command history. However the command will not be sent to the VM as long as the MECAFF console is in (or returns to) the “More...” status.
 - Halt typing by pressing the PA2 key (sending a HT immediate command).
 - Terminate the running program by pressing the PA3 key (sending a HX immediate command).

After pressing PA2 or PA3, pressing ENTER is necessary to show the new state.

The handling of PF keys in the MECAFF console is strictly local to MECAFF, the definition of PF keys in CP (`SET PFxx...`) is meaningless for a MECAFF connected terminal.

When a program based on the MECAFF-API starts fullscreen operations, this program and the MECAFF console have to share the terminal to handle their respective I/O operations:

- If any serial output is written by the program (or the surrounding EXEC) at startup, initiating the fullscreen mode will first enter the “More...” status.
- The program has exclusive and transparent access to the 3270 terminal during the recurring fullscreen “write screen” and “receive user input” cycles.
- The program “owns” the terminal during each single cycle, including the handling of all transmission triggering keys, so MECAFFs PF-and PA-settings are meaningless while the program owns the terminal.
- MECAFF has the opportunity to regain control over the terminal only between the fullscreen cycles (i.e. after the users input in fullscreen mode has been sent to the program).
- As long as no serial output from VM occurs, the environment will stay in fullscreen mode.

- While the program owns the terminal, any output issued asynchronously by CP (like “TAPE 181 ATTACHED” or messages from other users) is buffered by MECAFF until the fullscreen cycle is completed and MECAFF may re-gain ownership on the terminal: the screen is then switched back to console mode and the buffered lines are then normally output. Any other output after a fullscreen cycle also lets MECAFF switch back to console mode.
- If any lines were written in console mode since the last fullscreen cycle, the “More...” status will be entered when the next fullscreen cycle is started by the program.

When a VM session ends on a connected terminal (either via LOGOFF, DISCONN or being FORCED), MECAFF enters the “More...” state, which is automatically released after 3 seconds, clearing the output line and command history buffers.

4 The MECAFF tools for CMS

Based on the MECAFF-API, some CMS fullscreen programs are available, as well as a program to list the terminal characteristics. The programs FSHELP and FSLIST are more experimental programs on the development path to the editor EE. These programs are basically usable, but some nice functions are surely missing.

If the user did not login to the VM with a MECAFF connected terminal, the internal communication of the MECAFF-API will simply be written out on the terminal instead of the expected fullscreen setup of the program. As the first API-call executed will always be querying the terminal type, this command sequence to MECAFF will be displayed as well as an instruction text that should be followed by simply pressing ENTER:

```
fs-qry
<{>}T Please press ENTER to cancel fullscreen operation
```

In this case, the MECAFF-API will recognize that the external MECAFF process is not present and the program will terminate with an error message and an appropriate exit code:

```
** no valid response (terminal probably not connected to a MECAFF)
Ready(00001); T=0.01/0.01 21:23:59
```

4.1 FS-QRY – Query Informations

The CMS command FS-QRY lists the characteristics of the 3270 terminal emulator connected to MECAFF.

The command has no parameters. Provided the VM session is connected to MECAFF, the output of FS-QRY lists the following data:

```
fs-qry
Terminal type .... : 'IBM-DYNAMIC'
Alt-Screen ..... : yes
Colors ..... : yes
Extended Highlight : yes
Max. Screensize .. : 87 cols x 50 rows
SessionMode ..... : 3270
Ready; T=0.01/0.02 21:14:50
Running >> █
```

The meaning of the lines issued by FS-QRY should be obvious and simply verbalize the corresponding fields of the MECAFF-API. The line “Max. Screensize” is only present if the terminal supports an alternate screen size.

4.2 EE – Fullscreen editor

4.2.1 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 fullscreen editors, although EE currently supports only a small subset of those programs and allows only editing a single file.

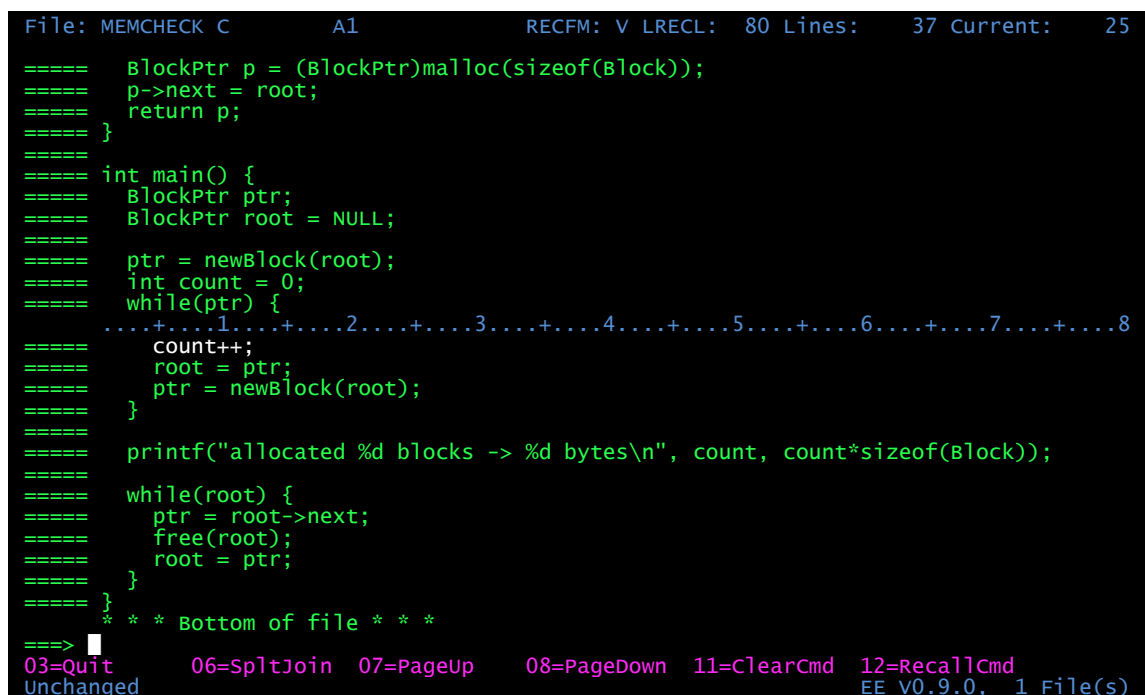
The editor EE is invoked from CMS with

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

If the CMS file *fn ft [fm]* does not exist, the file characteristics (RECFM, LRECL, case handling) are taken from the FTDEFAULTS command (see 4.2.3.3) for the file type (specified in the PROFILE EE, see 4.2.4).

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.

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



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

=====
BlockPtr p = (BlockPtr)malloc(sizeof(Block));
p->next = root;
return p;
=====
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);
        root = ptr;
    }
}

* * * Bottom of file * * *
==>
03=Quit      06=SplitJoin  07=PageUp   08=PageDown  11=ClearCmd  12=RecallCmd
Unchanged    EE V0.9.0, 1 File(s)
```

If the file loaded 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. If necessary, this protection can be removed (see UNBINARY).

4.2.2 Prefix commands

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

/	→	set this line to the current line
. <i>c</i>	→	set the line mark <i>c</i> 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.
I	→	insert an empty line after this line
D	→	delete this line
DD	→	mark one of the boundaries of a line block to be deleted
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

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.

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.

4.2.3 Commands

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

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

4.2.3.1 Editing and scrolling commands

BOttom

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

Change */string1/string2/* [*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).

Input

Enter 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.

Except for changes in the file zone of the editor, no operations are possible while in input-mode (no commands, no PF keys except for the PF03 key, 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).

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.

PGDown

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.

PGUP

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.

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.

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.

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;
- joined with the next line if the cursor is placed after the last non-whitespace character of the line, the first 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.

TOp

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

4.2.3.2 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.

`FILE [fn [ft [fm]]]`
`FFILE [fn [ft [fm]]]`

Save the file and terminate EE if the file could be 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.

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)

`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`

Terminate EE without saving the file. If the file was modified, `Quit` will not be executed. In this case, terminating EE without saving the file can be forced with `QQuit`.

`RECFM V | F`

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

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

Save the file (but do not terminate the editor). 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.

4.2.3.3 Configuration commands

`ATTR screen-object color [HIlighT]`

Define the text rendering attributes for the screen element types of the EE screen setup. Besides the color, passing `HIlighT` allows to specify the highlighted display on monochrome displays.

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 <code>INFOLines</code>
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`

Define the default record format and the logical record length to use when create a new file with the given file type, also defining the default case setting when opening a file with this file type. 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 `FTDEFault` 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`

`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, possibly 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.

```
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. 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.

4.2.3.4 Miscellaneous commands

```
CMS cmscommand ...
```

Execute a CMS command. 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 and 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).

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).

4.2.4 Customizing with SYSPROF EE and PROFILE EE

The editor EE 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 for the filetypes). The file PROFILE EE is intended to provide the user preferences.

Only configuration commands (see 4.2.3.3) 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 line can be concatenated up to a total length of 512 characters.

4.3 FSHELP – Fullscreen help

The fullscreen display of a standard HELP topic is invoked with

```
FSHELP topic
```

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

Browsing through the help information and leaving FSHELP is controlled with the PF-keys (see the screen bottom line). Although a command line is present, no commands are supported: any input is ignored.

4.4 FSLIST – Fullscreen file list

A fullscreen file list can be displayed with

```
FSLIST file-pattern
```

where *file-pattern* is any file specification allowed for the CMS command LISTFILE (which is in fact called to get the file list).

Browsing through the file list and leaving FSLIST is controlled with the PF-keys (see the screen bottom line). The following commands are accepted on the command line:

`Listfile file-pattern`

Change the search pattern for the file list.

`Quit`

Leave FSLIST.

Browsing the content of a file in the list can be achieved by moving the cursor to the line with the filename and pressing PF12. Browsing and leaving the file is also controlled with the PF-keys (see the screen bottom line).

5 MECAFF-API

The MECAFF-API consists a set of C routines defined in the file `FSIO_H` and implemented by the C module `FSIO_C` and a companion assemble file (`WR3270_ASSEMBLE`).

The following routines are provided:

1. Query terminal information
2. Full screen write
3. Full screen read

An application using the MECAFF-API should first query the terminal characteristics to verify that fullscreen operations are possible and to set up internal data.

However, this is not a strict requirement, as the full screen operations will check the MECAFF connection if the step was skipped. But in this case, the application should stick to minimal 3270 defaults (monochrome 80x24 screen) and not use the EWA command, as the presence and size of the alternate screen is unknown.

If the VM is not connected to MECAFF, the internal communication of the MECAF-API to query the terminal characteristics will simply be written out on the terminal. This command sequence to MECAFF contains a plain instruction text that should be followed by simply pressing RETURN:

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

Doing so allows the MECAFF-API to return the appropriate returncode to the application, which should handle this situation gracefully.

5.1 Query terminal information

The characteristics of the terminal connected to the VM via MECAFF can be queried with the following function:

```

int __qtrm(
    char *termName,
    int  termNameLength,
    int  *numAltRows,
    int  *numAltCols,
    bool *canAltScreenSize,
    bool *canExtHighLight,
    bool *canColors,
    int  *sessionId,
    int  *sessionMode);

```

The returncode specifies if the VM is connected to the terminal via a MECAFF process and if the other MECAFF-API routines may be used:

- The returned value 0 means that the query was successful and that fullscreen operations can be performed through the MECAFF-API.
 - Any value different from 0 means that no valid response came in for the query (no MECAFF process or some other internal protocol error).
- The MECAFF-API should not be used for fullscreen operations.

The input parameter `termNameLength` has to contain the total length of the buffer passed as `termName`. All other parameters are output parameters and filled by `__qtrm` if the returncode is 0 (if the call is successful). The values written to the output parameters have the following meaning:

- `termName`
the terminal type name, the buffer will be null-terminated and contain up to `termNameLength-1` characters.
- `numAltRows`
number of lines of the alternate screen size, if `canAltScreenSize` is true.
- `numAltCols`
number of columns of the alternate screen size, if `canAltScreenSize` is true.
- `canAltScreenSize`
If true, the terminal supports an alternative screen size, meaning that the EWA command can be used in the 3270 output stream to switch to the alternative screen.
- `canExtHighLight`
If true, the terminal supports extended highlighting (like reverse, underline, ...) through SFE (Start Field Extended) or SA (Set Attribute) orders.
- `canColors`
If true, a color terminal is connected.
- `sessionId`
this is the MECAFF internal identifier for the connection, this value can (and should) be ignored by client applications
- `sessionMode`
the connection style, with the values 3270 for GRAF-style and 3215 for CONS-style.

5.2 Full screen write

A 3270 outbound stream is sent to the terminal with the `__fswr` function:

```
int __fswr(
    char *rawdata,
    int   rawdatalength);
```

The 3270 outbound stream is passed in `rawData`, with the length of the stream passed in `rawDataLength`. The first byte of `rawData` must be one of the following 3270 CCW commands:

- W (0xF1, Write),
- EW (0xF5, EraseWrite)
- EWA (0x7E, EraseWriteAlternate)
- EUA (0x6F, EraseAllUnprotected)

The returncode specifies the outcome of the operation and the options for further processing in the client program:

- Returncode = 0
the 3270 outbound stream was successfully sent to the connected 3270 terminal. The keyboard of the terminal is locked and the application now owns the terminal until the 3270 inbound stream is requested.
- Returncode = 1
the stream could not be transmitted to the terminal: the terminal is currently owned by the MECAFF console, but the 3270 command passed in the 3270 output stream was W or EUA, expecting the terminal still to be in fullscreen mode (but the screen content of the last fullscreen operation was lost when the MECAFF console took control).
In this case, an EW or EWA command must be used to regain the ownership over the terminal and to rewrite the whole screen content.
- Returncode = 2
Fullscreen support is unavailable (not connected to a MECAFF process or some other internal protocol error).
- Returncode = 3
possibly recoverable communication problem: re-query the terminal information and retry with a EW resp. EWA fullscreen write.
- Returncode = 4
unsupported 3270 outbound stream command (3270 CCW commands WSF, RB, RM, RMA are unsupported by the MECAFF process)

5.3 Full screen read

After a successful screen write operation, the application owns the terminal until the user input from the terminal is transmitted to the application as 3270 inbound stream. The MECAFF console can regain the terminal ownership only after this transmission is complete.

A fullscreen read is initiated with a call to the function `__fsrd`. MECAFF unlocks the terminals keyboard and waits for the user to press one of the keys triggering the transmission (ENTER, PF-Keys, PA-Keys, ...). The raw 3270 data stream sent by the terminal is passed back to the application by `__fsrd`.

```
int __fsrd(
    char *outbuffer,
    int  outbufferlength,
    int  *transferCount);
```

If the read operation is successful, the 3270 inbound stream from the terminal is copied to `outbuffer` with up to `outbufferlength` bytes. The number of bytes transmitted is passed back in `transferCount`.

The returncode values have the following meaning:

- Returncode = 0
the fullscreen read operation was successful.
- Returncode = 1
the terminal is not owned by the application (there was no preceding successful fullscreen write operation or the terminal was forced out of fullscreen mode, i.e. the MECAFF console regained control due to some other output operation).
In this case, first re-gain screen ownership by doing a fullscreen write (using an EW or EWA CCW command).
- Returncode = 2
Fullscreen support is unavailable (not connected to a MECAFF process or some other internal protocol error).
- Returncode = 3
Possibly recoverable communication problem: re-query the terminal information and retry with a EW resp. EWA fullscreen write.
- Other returncodes
protocol-error, abort program

5.4 API files and linking

The MECAFF API consists of the following CMS files

```
BOOL      H
Include file with the definition of the type bool
```

```
FSIO      H
Include file for the MECAFF API calls.
```

```
FSIO      TEXT
Implementation of the MECAFF API functions.
```

```
WR3270    TEXT
Assembler adapter to call DISPW from C.
```

The `FSIO TEXT` is compiled by GCC with the CMS option, so linking a MECAFF based application must use the GCCLIB runtime `TXTLIB`.

6 Known restrictions / problems

6.1 General

- When using MECAFF fullscreen programs (including FS-QRY) while recording a console protocol with CP SOOL CONSOLE, the protocol will contain the encoded communication between the fullscreen programs and the external MECAFF process. This may be interesting, but it probably won't be what was intended, as the console protocol will be filled up with unreadable clutter.
- When the 3270 terminal is disconnected (the terminal emulator window is closed) while a fullscreen program is working, the fullscreen communication will fail after reconnecting even if the VM session is reconnected through a MECAFF-attached terminal.

The MECAFF tools (EE, FSLIST, FSHELP) currently cannot transparently proceed with a new fullscreen connection and will crash (losing changes to the edited file with EE).

6.2 MECAFF console

VM/370 (resp. CP) uses different interaction and presentation metaphors for the VM console depending on the terminal type connected to the VM. While some of the typical behavior can be normalized by MECAFF (like password prompts with CONS-style), at least the following differences between the 2 connection styles remain:

- Login
 - GRAF-style: as with 3270 terminals directly connected to VM/370, the ENTER key must first be pressed to get the initial "CP READ" prompt.
 - CONS-style: the login command may be entered without first getting a "VM read" prompt.
- Prompting for input
 - GRAF-style: the MECAFF prompt states "Running", "VM read" and "CP read" have the same meaning as for 3270 terminals directly connected, the "Enter pwd" state is synthesized from a "CP READ" VM-state with an invisible-text input field for the command prompt.

As for plain VM/370, a "Running" state can mean that a program is working (not expecting input) or that the VM is idle and would accept a CMS or CP command.
 - CONS-style: the "Running", "VM read" and "Enter pwd" states are synthesized based on the prompt string issued by Hercules (is the system expecting input?) and the last text line written by the VM (is it expecting a password?).

So the "Running" state means that a program is working and no input is currently expected (although type ahead and sending the next input is possible).
However, there is no "CP read" prompt with CONS-style.
- Standard CMS pseudo-fullscreen programs (EDIT, FLIST, ...)
 - GRAF-style: programs which use the VM/370 R6 limited support for 3270 terminal will try to do full screen writes, since MECAFF connects as a 3270 terminal.

These writes will succeed from the programs point of view, but MECAFF will attempt to serialize those outputs, which will look unusual if the 3270 terminal is 80 columns wide (as no line addressing happens and consecutive writes will not overlap/overwrite on the 80x24 display cells) and will be unreadable if the terminal has more than 80 columns (automatic wrapping will not occur where the program expected it)

- CONS-style: as MECAFF connects as a (non-3270) serial terminal, those programs will probably not try to use full screen writes and fall back to plain line oriented mode (like EDIT does)
- Wide terminals (more than 80 columns)
 - GRAF-style: lines wider than 80 characters per line are not wrapped by the terminal, but CP splits lines in 80 character chunks which are positioned separately on the 3270 screen.
So doing a TYPE on a file with up to 130 characters per line on a model-5 terminal (132 columns) will not write the lines nicely, but wrapping will seem to occur at column 80. Anything beyond the 130-th char is truncated (dropped).
MECAFF can only output the 80 character chunks as they are sent by the VM.
 - CONS-style: on serial terminals, CP does not insert line wraps at column 80, so output on a wide terminal has more similarity with the file. However lines will still be truncated after the 130-th char.
- Stopping a program from “More...”
 - GRAF-style connections: When stopping a program with the PA3-Key (sending an HX immediate command, see 3) while its output is held in “More...” status, the next command issued to CMS will ABEND, requiring to re-IPL CMS.
This is not MECAFF specific, as the same behavior occurs when connecting the 3270 terminal directly to VM/370. However, this may be a problem specific to 3270 sessions, as it did not show up when connected in CONS-style.

6.3 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.

7 Planned extensions and improvements

The following are just ideas on what could be done, with no particular priority...

7.1 MECAFF console

- Configurable attributes (color, highlight) for screen elements (output, command echo, prompt, ...), modifiable via a CMS command
- Configurable PF-key settings

7.2 EE

- Multi-file editing capability
- Additional commands: GET (insert file), PUT(D) (write a lineblock to a file), ...
- Additional prefix commands (shift right/left) and features (line count for currently single-sline commands)
- Online help
- 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)