MECAFF

Multiline
External
Console
And
Fullscreen
Facility

for VM/370 R6 SixPack 1.2

Version 0.9.5 (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

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 VM/380 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 (among others) the 3 basic operations needed to build fullscreen applications:
  → 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 and the settings of the console
- FS-CTL : set the properties of the console
- FSHELP : a fullscreen help (displaying the standard CMS help)
- EE : a combined fullscreen file editor (EE), filelist viewer (FSLIST) and file browser (FSVIEW), also allowing to invoke these basic functions from the CMS command line.

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 instead of using MECAFF.

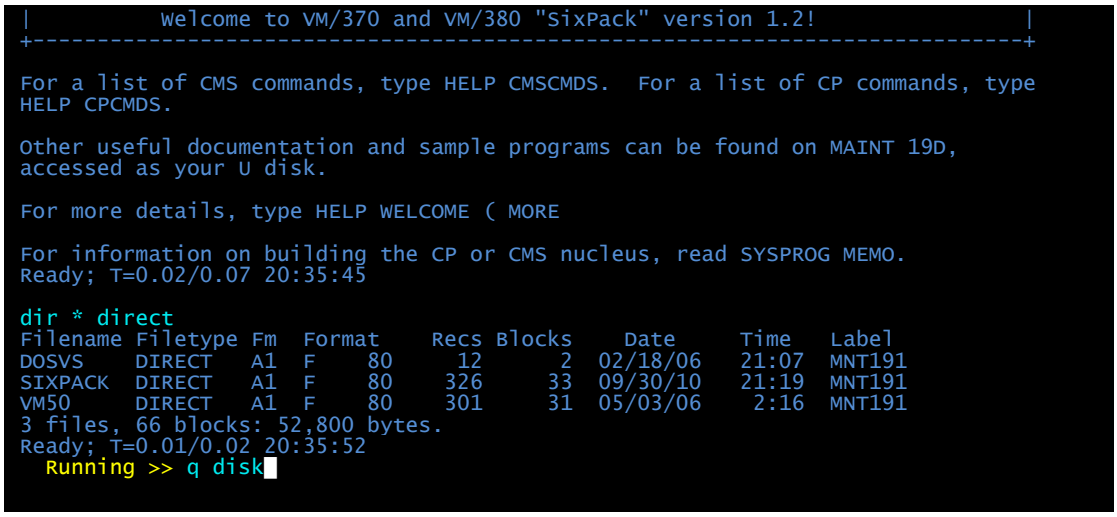
The term "multiline" in MECAFF means that several terminals can connect in parallel to the MECAFF process to start fullscreen capable sessions on the VM/370 machine.

The external MECAFF process is implemented as a stand-alone Java™ program. The CMS components (MECAFF-API and CMS 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:



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
  → 132 cols 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-3278-4 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)

For GRAF-style connections, MECAFF supports all 3270 lines provided by VM/370, as emulating a model 4 terminal allows MECAFF sessions to be connected to devices defined in DMKRIO as model 2 or as model 4.
Furthermore a LU-name may optionally be given for GRAF-style connections to bind the outgoing 3270 line to a group of lines defined in the Hercules configuration. The LU-name for a MECAFF session can be specified at 2 levels:

- The default LU-name can be predefined as parameter for the MECAFF process.
- The 3270 terminal emulator connecting to MECAFF can also define a LU-name in its connection specification.

If both are specified, the LU-name of the terminal connecting to MECAFF has priority.

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 (also: SixPack 1.1 with the statically linked MECAFF tools for CMS)
  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.
- If GRAF-style connections are to be bound to 3270 line groups via LU-names (for example to direct connections to lines configured as 3270 model 4 in DMKRIO), the corresponding group names have to be added to the device definitions in the Hercules configuration (.conf-file). The standard SixPack 1.2 configuration file already defines the groups MOD2 and MOD4.
- For CONS-style, serial terminal lines without the NOPROMPT option are needed in 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 standard 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, …).
  This scanning recognizes the specific texts issued by the VM/370-R6 SixPack Version 1.2 on

Hercules Version 3.07 resp. VM/380 SixPack Version 1.2 on Hercules 3.07:380-1.0.
If these texts from the mentioned software versions are modified or if other software versions are used (for example not VM/370 SixPack version 1.2), MECAFF must possibly be adapted for a correct state and session handling.

- 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
- `mecaff_s.aws`
  → the AWS tape file in CMS TAPE format containing only the MECAFF CMS tools as statically linked MODULEs
- `sixpack+mecaff.conf`
  → a sample Hercules configuration for SixPack 1.2; this configuration is the original file from 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/Win64 intended for a standard (fresh) SixPack 1.2 installation, using `sixpack+mecaff.conf` as configuration. This script starts the following components in parallel:
  - the MECAFF process
  - the Hercules-based VM/370 machine
  - one 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 be used as templates 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
- `-vmLUName:`*`luname`*
  Optionally specifies the LU-Name to use when opening a GRAF-style connection to the VM/370 (or VM/380) machine. The *luname* is the group name assigned to 3270 devices in the Hercules configuration. As the group name is case-sensitive, it must be specified in the MECAFF-option as spelled in the Hercules configuration file.
  This option defines the default LU-name to use if the connecting terminal does not specify a LU-name by itself. If no LU-name is specified (neither by the terminal nor the option -vmLUName), MECAFF will connect without specifying a LU-name and Hercules will assign a free 3270 device on the VM/370 machine.
  No default, i.e. don’t use a default LU-Name when opening a GRAF-style connection.
- `-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 CMS files for MECAFF are contained in 2 AWS tape files written with the CMS TAPE utility:

- The tape file `mecaff.aws` contains the common EXEC files for MECAFF and the MECAFF tools for CMS (see 4) up to the first tape mark and the MECAFF-API files with a demo program (see 5.6) up to the tape end.
  The MECAFF tools contained in this AWS tape file are MODULEs linked against the native CMS GCC runtime library GCCLIB. As these programs rely on the GCC native runtime loaded into CMS resident memory, these versions of the programs are called "dynamically linked".
- The tape file `mecaff_s.aws` contains the MECAFF tools directly linked against the objects of the GCC native runtime.
  These versions of the MECAFF tools do not depend on the GCC runtime loaded into resident memory and are therefore called "statically linked". These versions can be used if the resident GCC runtime is not available or cannot be used to avoid runtime conflicts (for example when DOS programs are used, as both use the DOSTRANS nucleus field).

The tape `mecaff.aws` has the following content:

```
  SAMPLE   EE       F2
  SYSPROF  EE       F2
  FS-QRY   MODULE   F2
  FS-CTL   MODULE   F2
  EE       MODULE   F2
  FSHELP   MODULE   F2
  FSLIST   EXEC     F2
  FSVIEW   EXEC     F2
tape-mark
  FSIO     H        F2
  BOOL     H        F2
  FSRDTEST C        F2
  FSIO     TEXT     F2
  WR3270   TEXT     F2
  CMSMIN   TEXT     F2
  FSRDTEST MODULE   F2
tape-mark
tape-mark
```

The tape `mecaff_s.aws` has the following content:

```
  FS-QRY    MODULE    F2
  FS-CTL    MODULE    F2
  EE        MODULE    F2
  FSHELP    MODULE    F2
tape-mark
  FSRDTEST MODULE     F2
tape-mark
tape-mark
```

To use the statically linked MECAFF CMS tools, the files from the tape `mecaff.aws` have to be first loaded and then overwritten with the files of the tape `mecaff_s.aws`.

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 a MECAFF tape 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
  ```

(the above commands also apply for loading the statically linked files from the tape `mecaff_s.aws` after installing the dynamically linked MECAFF files)

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 4.3.5).

Remarks:

- Deciding whether to install the statically linked MECAFF tools depends on the environment resp. the expected usage of the VM/370 system.
  If:
  - MECAFF is used with VM/370 or VM/380 in a version different from R6 SixPack 1.2
  - or loading of the GCCLIB into resident memory is not enabled in the system-wide SYSPROF EXEC
  - or any DOS-related programs is used under CMS

then:

> the statically linked version of the MECAFF tools must be installed, as the dynamically linked tools will not work at all or will provoke ABENDs in conjunction with the DOS-programs.

- If the help menu for FSHELP (see 4.6) is to be made available to all users, this help file should be generated by invoking FSHELP without parameters and then copied to disk U (or Y).
- If MECAFF version 0.9.0 was installed, the old version of FSLIST should be removed to prevent using it by inadvertence, as the old separate program is now replaced by an EXEC invoking EE and providing more functionality. Removing the old version can simply be done with:

```
erase fslist module *
```

- Under SixPack 1.2, disk A of user MAINT has a SYSPROF EXEC that hides the system wide one and prevents loading the native GCC runtime into resident memory for this user. Before the dynamically linked MECAFF tools can be used by MAINT, the local SYSPROF EXEC has to be disabled by renaming it away (and re-IPL-ing CMS or 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.
- PF3 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.
The commands bound to the PF keys can be queried with the MECAFF command FS-QRY and changed with the command FS-CTL. Besides the MECAFF internal functionality mentioned above (initially bound to the PF keys), almost arbitrary CMS or CP command texts can bound to the PF keys, sending these commands to VM when the PF key is pressed on the terminal.
As default, PF01 invokes the MECAFF tool FSHELP, displaying a menu of the help topics available on disk U.

Similarly the color settings resp. highlighting (for monochrome 3270 displays) for the different displayed screen elements of the MECAFF console (normal output from VM, input area, echoed input, …) can be queried with the command FS-QRY and modified with the command FS-CTL.

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 input data resp. state 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 tools are available, as well as a program to list the terminal characteristics.

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 in fullscreen mode, also providing an overview menu page and navigation to other help topics.

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
```

When the terminal connection is lost (the terminal is closed or the MECAFF process is terminated), the VM session is disconnected. If a MECAFF-based application is in fullscreen mode when this happens, 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 communication with MECAFF:

- this 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 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 4.3.6).

4.1 FS-QRY – Query Informations

The CMS command FS-QRY lists the characteristics of the 3270 terminal emulator connected to MECAFF, the current display attributes for visual elements of the console and the current PF settings of the MECAFF console.

The command has the following parameters to specify which information to display (optional parts of parameters are given in lower case):

VERsions

→ list the version information of the MECAFF-process and -API

```
fs-qry versions

Version information for MECAFF:
MECAFF process version : 0.9.5
MECAFF API version ... : 0.9.5

Ready; T=0.01/0.01 21:43:20
  Running >>
```

TErm

→ list the characteristics of the 3270 terminal emulator connected to MECAFF

```
fs-qry term

Characteristics of attached 3270 terminal:
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” will tell the size of the alternate screen if the terminal “Alt-Screen” is “yes” or 80 x 24 if the terminal is a model -2 not supporting an alternate screen.

ATtrs

→ list the display attributes for visual elements of the console, these elements are:

- OutNormal: output lines from VM/370
- OutEchoInput: commands entered on the MECAFF command prompt displayed in the output area

- OutFsBg: output lines from VM/370 received while the terminal is owned by the CMS program in fullscreen mode (temporary attribute used to highlight these lines while in More… state)
- ConsoleState: state indicator in front of the input prompt of the MECAFF console.
- CmdInput: the input prompt of the MECAFF console.

→ the attributes consist of an optional Highlight indicator (relevant for monochrome 3270 terminals) and the color specification (Default, Blue, Red, Pink, Green, Turquoise, Yellow, White)

```
fs-qry attrs

Settings of MECAFF console:
OutNormal ........ : Blue
OutEchoInput ..... : Turquoise Highlight
OutFsBg .......... : Pink
ConsoleState ..... : Yellow
CmdInput ......... : Turquoise

Ready; T=0.01/0.04 22:01:15
  Running >>
```

PFkeys

→ list the settings of the PF-keys of the MECAFF console

→ internal MECAFF-commands assignable to PF-keys start with an exclamation mark:

!CMDCLR	→ clear input area of the console
!TOP	→ scroll the output area to the start of the buffered lines
!BOTTOM	→ scroll the output area to the end of the buffered lines
!PAGEUP	→ scroll the output area one page towards the start of the buffered lines
!PAGEDOWN	→ scroll the output area one page towards the end of the buffered lines
!CMDPREV	→ step the input prompt back in input history
!CMDNEXT	→ step the input prompt forward in input history

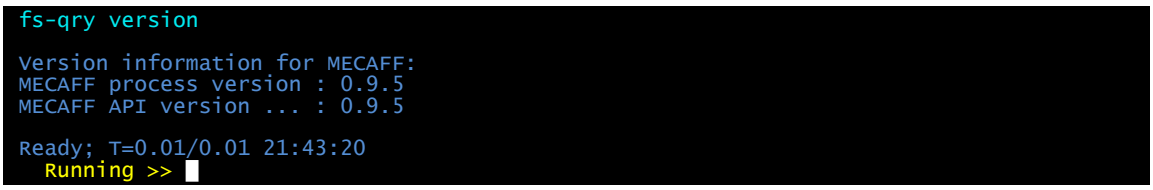
```
fs-qry pfkeys

PF-Key settings of MECAFF console:
PF01  : FSHELP
PF02 not set
PF03  : !CMDCLR
PF04 not set
PF05 not set
PF06  : !TOP
PF07  : !PAGEUP
PF08  : !PAGEDOWN
PF09  : !BOTTOM
PF10 not set
PF11  : !CMDNEXT
PF12  : !CMDPREV
PF13 not set
PF14 not set
PF15 not set
PF16 not set
PF17 not set
PF18 not set
PF19 not set
PF20 not set
PF21 not set
PF22 not set
PF23 not set
PF24 not set

Ready; T=0.02/0.06 22:08:35
  Running >>
```

VERsions

→ list the version information of the MECAFF-process and API



ALl

→ list all the information above.

STATe

→ check if the terminal is connected via MECAFF and set the return code accordingly:
RC = 0 : the terminal is connected through MECAFF and MECAFF tools can be used
RC != 0 : the terminal is connected directly to Hercules, the MECAFF tools won't work

→ no output is written to the screen.

4.2 FS-CTL

The CMS command FS-CTL allows to set the display attributes for the visuals elements of the MECAFF console as well as the functionality bound to the PF keys when the terminal is owned by the MECAFF console.

The command has the following syntax (optional parts of command tokens are given in lower case):

```
FS-CTL  ATTR elem color [ HIGHLight ] [elem color [ HIGHLight ] …]
        PF pfno [ cmd-text ]
```

Using the subcommand ATTR sets the color and optionally the highlighting for the elements of the MECAFF console, where *elem* can be:

- NORMal: output lines from VM/370
- ECHOinput: commands entered on the MECAFF command prompt displayed in the output area
- FSBG: output lines from VM/370 received while the terminal is owned by the CMS program in fullscreen mode (temporary attribute used to highlight these lines while in More… state)
- CONSolestate: state indicator in front of the input prompt of the MECAFF console.
- CMDInput: the input prompt of the MECAFF console.

The *color* can be one of: Default, Blue, Red, Pink, Green, Turquoise, Yellow, White.
Several visual elements can be set with one FS-CTL command, when specifying the same elements more than once only the last setting will be use.
Example:

```
FS-CTL ATTR  NORMAL GREEN  ECHO WHITE HIGHLIGHT  FSBG RED
```

The subcommand PF sets the command issued by MECAFF when the corresponding key is pressed while the screen is owned by the MECAFF console. The command bound to a PF key can be either an internal MECAFF command or a command string that will be sent to the VM as if it had been given on the command line.

The following internal commands can be bound to PF keys (with the PF key they are assigned per default):

!CMDCLR → clear input area of the console (PF03)
!TOP → scroll the output area to the start of the buffered lines (PF06)
!BOTTOM → scroll the output area to the end of the buffered lines (PF09)
!PAGEUP → scroll the output area one page towards the start of the buffered lines (PF08)
!PAGEDOWN → scroll the output area one page towards the end of the buffered lines (PF07)
!CMDPREV → step the input prompt back in input history (PF12)
!CMDNEXT → step the input prompt forward in input history (PF11)

A command text to be sent to VM can be up to 60 characters long, longer command strings will be truncated. If no command is given, the command currently bound to the PF key will be removed. Examples:

```
FS-CTL PF 2 FSLIST * C
```
(set PF 02 to send the command "FSLIST * C" to CMS)

```
FS-CTL PF 1 !CMDPREV
```
(set PF 01 to retrieve the previous entry of the command history into the prompt area)

```
FS-CTL PF 12
```
(remove the command bound to PF 12)

4.3 EE – Fullscreen editor

4.3.1 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, although EE currently supports only a small subset of those programs.

The editor EE is invoked from CMS with

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 4.3.4.4) for the file type (specified in one if EE's profiles, see 4.3.5).

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 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 * * *
===>
02=RingNext 03=Quit 06=SpltJ 07=PgUp 08=PgDw 10=Reset 11=ClrCmd 12=Recall
Unchanged                                                  EE V0.9.5,  1 File(s)
```

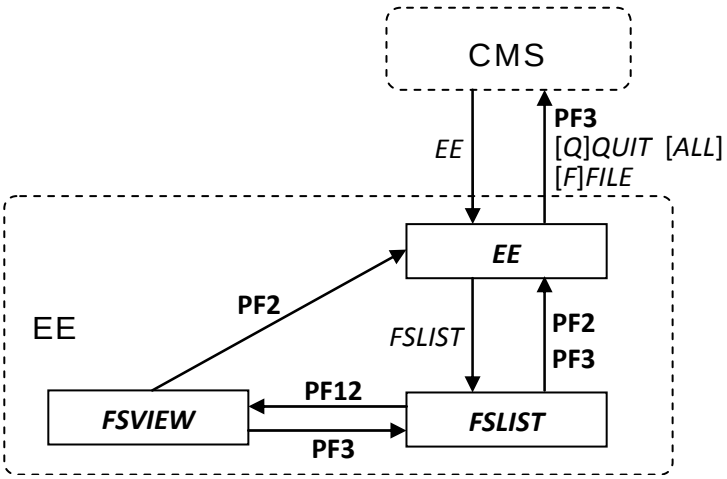
If a file loaded into EE contains binary characters (codes 0x00 to 0x3F or 0xFF), these characters are replaced with a dot ('.') in the file buffer, the file is marked as binary and EE will prevent all write operations to the original disk file. If necessary, this protection can be removed (see UNBINARY).

4.3.2 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:



4.3.3 Prefix commands

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

`/`	→	set this line to 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.
`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
`"`	→	duplicate this line
`""`	→	mark one of the boundaries of a line block to be duplicated
<[*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 4.3.4.4):
  - `?` → 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.

4.3.4 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 `/`

4.3.4.1 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 4.3.4.4), either in the profiles for EE (see 4.3.5) 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
PF10    :    PINPUT
PF11    :    CLRCMD
PF12    :    RECALL

PF13    :    TABBACKWARD
PF16    :    REVSEARCHNEXT
```

For the input-mode and the programmers-input-mode, the PF key command assignments are fixed (see the description of these modes in 4.3.4.2).

4.3.4.2 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).

`DELete [ count ]`

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

`GET [ 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$BUF. If *fm* is omitted, the filemode A is assumed. When no fileid is given, the file PUT EE$BUF A is read.

`GETD [ 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).

`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`
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.

`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.

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

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. If no *count* is given, the current line is moved by 1 line.

PUT [*count* [*fn* [*ft* [*fm*]]]]
PPUT [*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$BUF. If *fm* is omitted, the filemode A1 is assumed. When no fileid is given, the file PUT EE$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*]]]]
PPUTD [*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 4.3.4.4) 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.

4.3.4.3 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`*]]

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`*]]]
`FFILe` [*`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 4.4).

`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*]]]
SSAVe [*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.

4.3.4.4 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`*

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 `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

`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.

`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.

4.3.4.5 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).

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).

4.3.5 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 4.3.4.4) 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.

4.3.6 Rescue mode

When the terminal is disconnected while editing files with EE, reconnecting to the VM with a standard terminal connection (not using MECAFF) will prevent EE from re-establishing a fullscreen session 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 MECAFF 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.

4.4 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 EE editor provides a FSLIST subcommand with the same functionality.

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. 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)

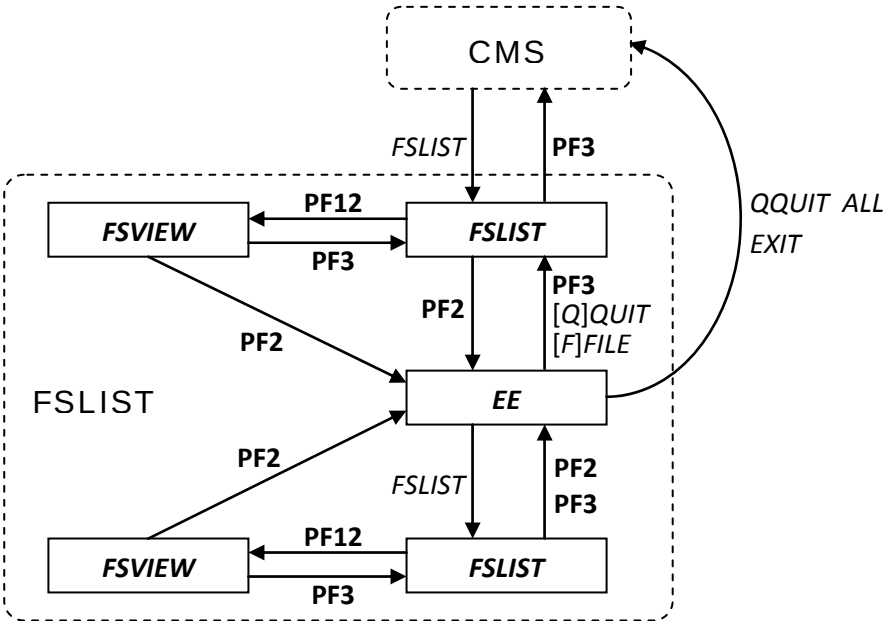
`Quit`

Leave FSLIST (also PF03)

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:



4.5 FSVIEW – Fullscreen file browser

A CMS can be browsed from CMS with FSVIEW with

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

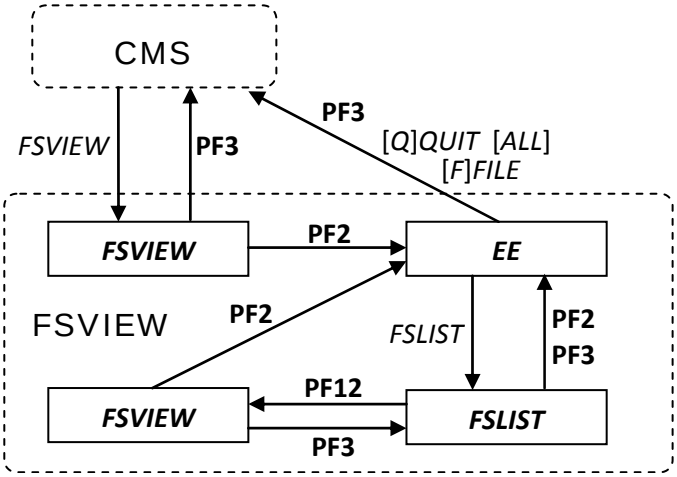
or

```
EE fn fm [ ft ] ( 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 (see the screen bottom line). FSVIEW allows to start editing the file with EE by pressing the PF02 key.
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 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:



4.6 FSHELP – Fullscreen help

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

```
FSHELP topic
```

This looks for a file *topic* HELP*xxx* on disk U (with *xxx* one of CMD, DBG, EDT, EXC or REX) and displays it. If the additional help file exists (*topic* HELP*xxx*2 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, 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.
- 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.
- PF06: *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.
- 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.

`Quit`
: Terminate FSHELP

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 basically provided to fullscreen applications:

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

Further routines allow managing the MECAFF console and the ownership of the screen after entering fullscreen mode.

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 3270 CCW 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 MECAFF-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 a read operation completes or the ownership is explicitly released by the application.

MECAFF provides several options to fullscreen applications for retrieving input from the 3270 terminal:

- Synchronous read: the program requests the read and waits indefinitely until the terminal transmits the users input. This is the standard read operation which will probably used by most fullscreen applications.
  The user input from the terminal is transmitted to the application as 3270 inbound stream. The MECAFF console regains the terminal ownership after this transmission is complete.
- Asynchronous read: the program requesting the fullscreen read is decoupled from the users effective response, either by
  - simply querying the availability of user input, but not waiting for the user to finish (and send) the input, allowing the CMS program to continue working while the MECAFF process handles the fullscreen read (and to retrieve the input at a later time by doing another fullscreen read, possibly again as asynchronous read)
  - or by specifying a timeout for the read, waiting for the input and regaining control
    - to continue computing if the user did not send the input before the timeout
    - resp. to receive the users input if was sent before the timeout.

  If user input is available and it is transmitted to the fullscreen program, the MECAFF console regains the terminal ownership immediately after the transmission.
  If input is not available (timed out read) or is not transmitted (query-only read), the terminal ownership will stay with the application for a grace period, allowing the program to keep the terminal in fullscreen mode while doing background computations. After the grace period, the MECAFF console may regain the ownership if necessary, allowing to display serial output from the VM that arrived while the terminal was in fullscreen mode or arrives after the grace period if the program did not re-acquire ownership by starting a new fullscreen cycle.

The default grace period is 3 seconds and can be changed with the MECAFF-API for subsequent asynchronous reads. Additionally, the fullscreen program can explicitly release the ownership on the terminal and give it back to the MECAFF console.

5.3.1 Standard (synchronous) Read

A standard synchronous 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

5.3.2 Extended (generalized) Read

The function `__fsrdp()` provides a more general fullscreen read operation than `__fsrd()`, allowing querying or polling for the availability of user input or waiting for user input with or without timeout.

Compared to the standard read function, `__fsrdp()` has one additional parameter specifying the fullscreen read mode resp. timeout interval. When calling `__fsrdp()`, MECAFF unlocks the terminals keyboard and processes the request as specified by the parameter `fsTimeout`.

```
int __fsrdp(
  char *outbuffer,
  int   outbufferlength,
  int  *transferCount,
  int   fsTimeout);
```

If the specified read operation requests the transmission of the user input and the input is available at completion of the call, 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 parameter `fsTimeout` controls the behavior of the fullscreen read operation. The following values can be passed in `fsTimeout` for the specified behavior, resulting in the specified return codes to identify possible outcomes of the request (with the specified constants defined in `FSIO H`):

- `FSRDP_FSIN_NOTIMEOUT`
  Standard synchronous fullscreen read, `__fsrdp()` behaves like `__fsrd()` (see 5.3.1)
- `FSRDP_FSIN_QUERYONLY` (or any negative value)
  Query if input is currently available, returning the status in the return code of `__fsrdp()`:
  → `FSRDP_RC_NO_INPUT` : no input currently available
  → `FSRDP_RC_INPUT_AVAILABLE` : a 3270 input stream is available and can be retrieved with an additional call to `__fsrd()` or `__fsrdp()`.
- `FSRDP_FSIN_QUERYDATA` (or 0)
  Poll for input, querying if input is currently available and if so, transfer it to `outbuffer`. The following return codes identify the query result:
  → `FSRDP_RC_NO_INPUT` : no input currently available
  → 0 : a 3270 input stream is available and is present in `outbuffer`.
- any value > 0
  Synchronous fullscreen read with timeout.
  The parameter `fsTimeout` specifies the maximum time interval in 1/10 seconds to wait for an user input to arrive from the terminal. If input is available before the timeout, the 3270 inbound stream is copied to `outbuffer`. The following return codes identify the outcome of the call:
  → FSRDP_RC_TIMEDOUT : fullscreen input request timed out, no input is available
  → 0 : a 3270 input stream arrived before the timeout and is present in `outbuffer`.

Furthermore, the following return codes identify technical problem conditions independently of the requested behavior for `__fsrdp()`:

- Returncode = 1
  the terminal is not owned by the application (the 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

5.3.3 Grace period setting

The grace period for subsequent calls to `__rsrdp()` can be set with the routine `__fsgp()`.

```
void __fsgp(unsigned int gracePeriod);
```

The grace period is specified in 1/10 seconds. The values passed will be bounded by MECAFF to the grace period range supported, currently 1/10 to 10 seconds (i.e. values 1 to 100 for `gracePeriod`).

5.3.4 Release fullscreen terminal ownership / cancel fullscreen mode

A program can explicitly terminate the fullscreen operation by releasing the ownership on the terminal (without having to wait for the expiration of a timeout or the grace period).

Calling the routine `__fscncl()` allows MECAFF to return immediately to console interaction with the user. Incoming console output from VM (already arrived while in fullscreen mode or coming in after the call to `__fscncl()`, for example the ready message from CMS) will clear the screen and let the MECAFF console take over control.

```
void __fscncl();
```

5.4 Querying the version of MECAFF

A program can retrieve the version information of the connected MECAFF process and the API with the following routine:

```
bool __fsqvrs(
          int *mecaffMajor, int *mecaffMinor, int *mecaffSub,
          int *apiMajor, int *apiMinor, int *apiSub);
```

The return value indicates if the terminal is connected through a MECAFF process (`true`) and the version data for the MECAFF process are meaningful or not (`false`).

5.5 Miscellaneous Operations

The MECAFF API defines some additional internal operations intended for the tools FS-QRY and FS-CTL, which are mentioned here only for completeness:

- `__qtrm2(…)` : query console information and console visual settings
- `__qtrmpf(…)` : query console PF setting
- `__strmat(…)` : set console visual attributes
- `__strmpf(…)` : set console PF key command

5.6 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.

`CMSMIN TEXT`
Minimal subset of the GCCLIB runtime required by FSIO, containing the implementation of the functions `CMSconsoleWrite()`, `CMSconsoleRead()` and `CMScommand()`.

The FSIO TEXT is compiled by GCC with the CMS option. There are 2 options when linking a MECAFF based application:

- Compile the C modules with the GCC-option LIB GCCLIB (or the CMS shortcut-option) and linking the application against the GCCLIB runtime TXTLIB.
- Compile the C modules for the standard library (passing no LIB option or one of the options LIB PDPCLIB resp. OS) and adding the module CMSMIN from the MECAFF distribution to the LOAD or INCLUDE commands when linking against the PDPCLIB.

The MECAFF tools are built using the first alternative (both the dynamic and static linked versions).

5.7 Demonstration program for the MECAFF-API

The program FSRDTEST provided with its C source demonstrates the use of the fullscreen write with `__fswr()` and fullscreen reads with `__fsrdp()`, as well as releasing the terminal ownership with `__fscncl()`.

The different modes for writing and reading in fullscreen mode are selected with PF keys, as shortly explained on the main screen of the program:

```
RTrips:     8
Mode    : synchronous
CcwMode : EraseWrite
Writing : Write screen even when no input


PF01 : toggle CCW ( W / EW )
PF02 : toggle writing if no input read

PF05 : read time-out 2 secs
PF06 : read time-out 0.5 secs
PF07 : FSIN_QUERYONLY, immediate read
PF08 : FSIN_QUERYONLY, delayed read
PF09 : FSIN_QUERYONLY, long, imm. read
PF10 : FSIN_QUERYDATA
PF12 : FSIN_NOTIMEOUT (synchronous)

PF03 : terminate program with message with MECAFF waiting
PF15 : terminate program with message, waiting on CMS side
CLEAR: terminate program without final message screen
```

Each fullscreen roundtrip (synchronous or asynchronous read) will increase the RTrips counter.

Most PF keys switch between specific usage modes of the MECAFF-API calls, the current settings are displayed in the lines *Mode*, *CcwMode* and *Writing*:

- PF01 toggles between using the CCW commands Write or EraseWrite when overwriting the screen content.
- PF02 toggles between overwriting the screen on each roundtrip or only if input was effectively available (relevant when an asynchronous `__fsrdp()` mode is used, the effect will be visible on the *RTrip* counter which will not change although fullscreen reads occur)
- PF05 and PF06 use a time out read, allowing the CMS program to receive control on user input or after a defined time delay.
- PF07, PF08 and PF09 use the query only (polling) mode of `__fsrdp()` and differ on when the input stream is retrieved if the query told it is available (immediately, after short or long computation time). In difference to the time out variants above, the program continuously consumes CPU while in one of these modes.

- PF10 uses the query for user input mode and resembles to PF07 except that no additional read operation (round-trip) is necessary to retrieve the input.
- PF12 switches back to the initial synchronous read, with the program waiting until the user sends the input.

Leaving the program has with 3 options:

- PF03 writes a final message in fullscreen mode and does a (2 seconds) time out read, allowing the user to close the program premarurely by pressing any transmission key.
- PF15 write the same final message in fullscreen mode, but does no fullscreen read; instead the program waits 4 seconds before taking of the down the screen. The user cannot end the program before this time is elapsed.
- CLEAR terminates the program immediately.

6 Known restrictions / problems

6.1 General

- When using MECAFF fullscreen programs (including FS-QRY and FS-CTL) while recording a console protocol with CP SPOOL 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.

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 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…

7.1 EE

- 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)

8 Change history

8.1 0.9.0 ⇨ 0.9.1

- EE
  - New functionality
    - Multi-file editing capability, including prefix block operations between files (source and target of copy/move operations in different files)
    - Integration of FSLIST with EE, allowing to use FLIST to open a file for editing
    - Handling of disconnect and reconnection to a new terminal (re-establish full-screen session, with fallback to a rescue session if not connected via MECAFF)
  - New commands
    - EEdit *fn* [*ft* [*fm*]]
    - EXIt
    - RINGNext , RN
    - RINGPrev, RP
    - FSLIST [*pattern*]
    - PUT, PPUT, PUTD, PPUTD [*fn* [*ft* [*fm*]]]
    - GET, GETD [*fn* [*ft* [*fm*]]]
    - DELete [*count*]
  - Extended commands
    - QQuit [ALL]
      (new optional parameter ALL to close all edited files without saving)
    - Input [*line-text*]
      (optional text after the command will insert a new line with the text instead of entering input mode)
  - New prefix commands
    - " (duplicate line)
    - "" .. "" (duplicate block of lines)
  - Bugfixes
    - DD prefix command: defining the boundaries of a DD-block in different round-trips (different screens) no longer leads to repetitive error messages forcing EE out of full-screen mode
- FSLIST
  - New functionality
    - Integration of EE with FSLIST , allowing to edit a file from FSLIST (with PF2)
    - the color settings for FSLIST (and FSVIEW) are initialized from the corresponding ATTR commands in the profiles for EE
  - New commands
    - Sort
  - Bugfixes
    - when changing the list with command Listfile, the pattern showed in the title line is adjusted accordingly instead of displaying the initial file pattern of the FLIST session
    - the header line for the file list is now shown with the HEADLine attribute (blue as default) instead of the FILe attribute (green as default), as it not part of the list
- FSVIEW
  - File viewer functionality is now available directly from CMS
  - Switch to editing the file browsed with EE (with PF2)

8.2 0.9.1 ⇨ 0.9.3

- MECAFF-Process / MECAFF-API
  - New functionality
    - GRAF-style:
      - MECAFF now emulates a minimal 3270 model 4 (80x43 geometry) instead of model 2 (with 80x24 geometry)
        - support for all 3270 model types (2 and 4) configurable in DMKRIO
        - all 3270 lines of a VM/370R6 under Hercules machine can now be used
      - support for LU-names (group names in Hercules)
        - connections from MECAFF to Hercules can optionally be bound to a group of 3270 lines
        - a default LU-name can be given on the command line for the MECAFF process
        - this default can be overruled by a LU-name of the connecting terminal
        - if no LU-name is specified, the MECAFF connection is bound to an arbitrary device assigned by Hercules (as in MECAFF versions before 0.9.3)

        → new command line parameter *–vmLUName* for the external MECAFF program
    - API:
      - additional variants for fullscreen reads, supporting
        - polling read / query availability of user input, allowing asynchronous processing
        - time out read, returning control to the program after the time interval if no user input occured
      - new API calls:

        __fsrdp() : polling/time-out fulllscreen read

        __fsgp() : set grace period for terminal ownership after __fsrdp()

        __fscncl(): cancel fullscreen operation (release terminal ownership)
      - new sample program FSRDTEST (C source and executable)
  - Bugfixes
    - API-Call __fsrd():

      fix: rc 4 (protocol error) is now returned whenever the input buffer is too small for the 3270 input stream received

      symptom: passing outbufferlength <= 0 locks up communication with the MECAFF process
    - GRAF-mode connections to VM

      fix: race and deadlock conditions removed in background MORE [→HOLDING] →RUNNING handshake, occurring mostly when outputting large amounts of lines on TN3270-emulators allowing fast AID keystrokes (which do not ensure a minimal time interval when sending consecutive user inputs, unlike wc3270/x3270)

      symptom: pressing ENTER in a fast sequence could lock up the handshake mechanism for the session, preventing further output from VM/370 on that terminal
    - GRAF-mode connections to VM

      fix: introduction of minimal time intervals between input sent to VM, both for internal handshaking input (ENTER-Aid for MORE->HOLDING or CLEAR-Aid for MORE|HOLDING->RUNNING) and for user input

      symptom: fast input to VM (e.g. CLEAR then ENTER in less than 1 ms) can lead to a quantum-physical state of the VM after FORCE-ing it, as the user seems to be both logged off and on:

      - QUERY *user* and FORCE *user* claim that this user is not logged on
      - QUERY NAMES lists the user a logged on
      - trying to log on claims the user is still logged on

- MECAFF-Console
  - New functionality
    - attribute settings (color, highlight) for visual elements of the console can now be queried (CMS command FS-QRY) and modified (new CMS command FS-CTL)
    - setting of PF-keys can now be queried (CMS command FS-QRY) and modified (new CMS command FS-CTL)
- FS-QRY
  - New functionality
    - Now allows to list the terminal characteristics (as previous versions, subcommand TERM), the display attributes of the console (subcommand ATTRS), the PF-key settings (subcommands PFKEYS) or all these informations (subcommand ALL)
    - Also allows to simply query if the terminal is connected via MECAFF and return the state in the exit code (return status 0 or 1)
- FS-CTL
  - New command
    - Set the display attributes for visual elements of the MECAFF console
    - Set the command issued through the MECAFF console when pressing a PF key is pressed, either as command sent to the VM or processed internally in the console (like paging through the output history etc.)
- EE
  - Bugfixes
    - Prefix command single-line duplicate ("): false error message (“invalid prefix command”) after executing command removed
- General
  - The MECAFF tools for CMS are now delivered in 2 versions:
    - linked against the memory resident GCCLIB as it is standard in SixPack 1.2 (called “dynamically” linked)
    - linked against the runtime files of GCCLIB loaded into resident memory (called “statically” linked)

    see section 2.2.2 for criteria on when to install which versions.

8.3 0.9.3 ⇨ 0.9.4

- MECAFF-Process
  - Bugfixes
    - Dependency of the JRE setup type
      symptom: the MECAFF-console does not work at all (hangs after showing the Hercules connection screen, ignoring any input from keyboard or output from VM/370) when the Java runtime (JRE) was installed with the *net installation* setup instead of the *full installation* setup. The net setup is a small file with about 900 KB, which downloads the remaining 11 MByte of the JRE setup, the full setup has about 16 MByte in a single file (all sizes for Win32); as the size difference suggests, the net setup seems to be a subset of the full install, at least missing the EBCDIC charset support required by MECAFF.
      fix: MECAFF no longer relies on the Java built-in text conversion, but now uses its own internal translation tables for EBCDIC⇔ASCII conversion (bracket EBCDIC charset).
- MECAFF-Console
  - New functionality
    - The PF01 key now issues the CMS command FSHELP as default, opening the (new) help menu page (see below).
- FSHELP
  - Support for navigation to a new help topic by moving the cursor to a word in the current help text and pressing the PF01 key. If available, the help topic corresponding to the word under the cursor is opened.
    The help topics are stacked in a last-in-first-out fashion: the PF03 key now closes the current help topic and returns to the previous topic. Closing the first (initial) topic terminates the FSHELP program. The program can be left from any topic on the stack with the PF15 key (generally the shift-PF03 key combination).
  - Invoking FSHELP without parameters now displays a help menu giving an overview over all help topics available on disk U. This help topic page is generated if not available on an accessible disk (or if FSHELP is invoked with the option REBUILD), grouping the help topics on disk U by command type (CMS, CP etc.). This menu page is saved to the file MENU FSHELP A2 and is then displayed as help page. This help file will subsequently be used if the menu is to be displayed.
  - New subcommands:
    - Help *topic*
      → open the help for *topic*, if available.
    - */text/*
      → search forward for *text*
    - *-/text/*
      → search backward for *text*
- FSLIST
  - New subcommands:
    - */text/*
      → search forward for *text*
    - *-/text/*
      → search backward for *text*
- FSVIEW
  - New subcommands:
    - */text/*
      → search forward for *text*
    - *-/text/*
      → search backward for *text*

8.4 0.9.4 ⇨ 0.9.5

- MECAFF-Process / MECAFF-API
  - access to version numbers (*Major*.*Minor*.*Sub*) of the connected MECAFF process and of the MECAFF-API through the new API call __fsqvrs()
  - Linking against PDPCLIB: a minimal version of the GCCLIB is now provided, allowing to use the MECAFF-API with the PDPCLIB
- FS-QRY
  - new command line parameter VERsions to output the version data of the connected MECAFF process and the MECAFF-API
- EE
  - New functionality
    - New Subcommands: TABforward, TABBackward, TABSet, FTTABDEFaults
      - Support for software tab stops:
        - TABSet (re)defines tab stops for the current file
        - FTTABDEFaults defines default tab stops for a file type
        - tab positions currently defined are marked on the scale line
      - Moving the cursor to a tab position is supported in the file area for normal edit mode, for input-mode and for programmers-input-mode:
        - TABforward (PF01) moves cursor to the next tab position
        - TABBackward (PF13) moves cursor to the previous tab position or the first column
      - If invoked from the command line, TABforward (PF01) moves the cursor to the last known cursor position (file line and column) in the file area or to the current line if the last known file line position is not visible or was deleted.
    - New subcommands: SHIFT and SHIFTCONFig
      - SHIFTCONFig CHEckall | MINimal | LIMit | TRUNCate [*shiftBy*]
        define the default shift behavior when the shifted text reaches a line border and the default shift count
      - SHIFT [*by*] Left|Right [*count* | :*line* | .*mark*] [*shiftMode*]
        shift lines around current line left or right, truncating or limiting to borders
    - New mode: programmers-input-mode
      - New command PInput, assigned per default to PF10
      - adds a single new line with the indentation of the preceding line on entering PInput resp. on each Enter-Key if the current line was modified or no other line was changed
      - all file lines on screen can be edited in addition to the current input line
      - programmers-input-mode has own fixed PF settings:
        - PF01/PF13 : TAB forward / backward
        - PF03/PF15 : leave PInput
        - PF06 : SPLTJOIN
        - PF10: move PInput-line after this line(if cursor is in the file area)
    - New subcommands: SEArchnext and R[EV]SEArchnext
      - SEArchnext (bound per default to PF04, with shortcut: /):
        - if the last LOCATE command had exactly one target specified as search pattern
        - repeat previous text search in the last search direction (down or up)

        - REVSEArchnext / RSEArchnext (bound per default to PF16, with shortcut: -/):
          - if the last LOCATE command had exactly one target specified as search pattern
          - reverse the search direction and repeat previous search in the new direction
    - New prefix commands
      - single line shift: < , >
      - block of lines shift: <<..<< , >>..>>
      - shift prefix commands can have options: [1..9][?|:|!|#]
        1..9 : shift-by count, default 1
        ?|:|!|# : shift mode, with ? = CHEckall, : = MINimal, # = LIMit ! = TRUNCate
  - Modified functionality
    - Subcommand CMS
      - when invoked without a CMS or CP command as parameter, EE leaves temporarily the fullscreen mode and the CMS Subset level is entered on the MECAFF console. This allows to enter CP/CMS subset compatible commands until the command RETURN is entered, which will leave the CMS subset and restore fullscreen EE editing.
    - Subcommand SPLTJOIN
      - Split: the new line (rest of the splitted line) now has the same indentation as the original line
      - Join: leading spaces are removed from the joined (second) line before appending to previous line at the cursor position
    - Subcommands Quit and QQuit
      - if prefix commands are present on the screen or are pending from a previous screen, these commands now act like RESET: prefix commands are removed (instead of executing them) and the current file (resp. EE) are not closed.
      - this gives the PF03 key (to which Quit is usually assigned) a generic "escape" functionality depending on the current mode:
        - leave Input resp. PInput mode
        - reset/cancel prefix operations
        - close the current file
        - close EE
- FSLIST / FSVIEW / FSHELP
  - New functionality : Repeat previous search
    - Subcommand / or PF04: repeat the last search in the same direction
    - Subcommand -/ or PF16: reverse search direction and repeat search in new direction
    - (both subcommands must be given without trailing blanks, as this would start a new search for the whitespace given)
- FSVIEW
  - New functionality : the line number of the first displayed file line on the screen and the total number of lines of the file are displayed in the top header line