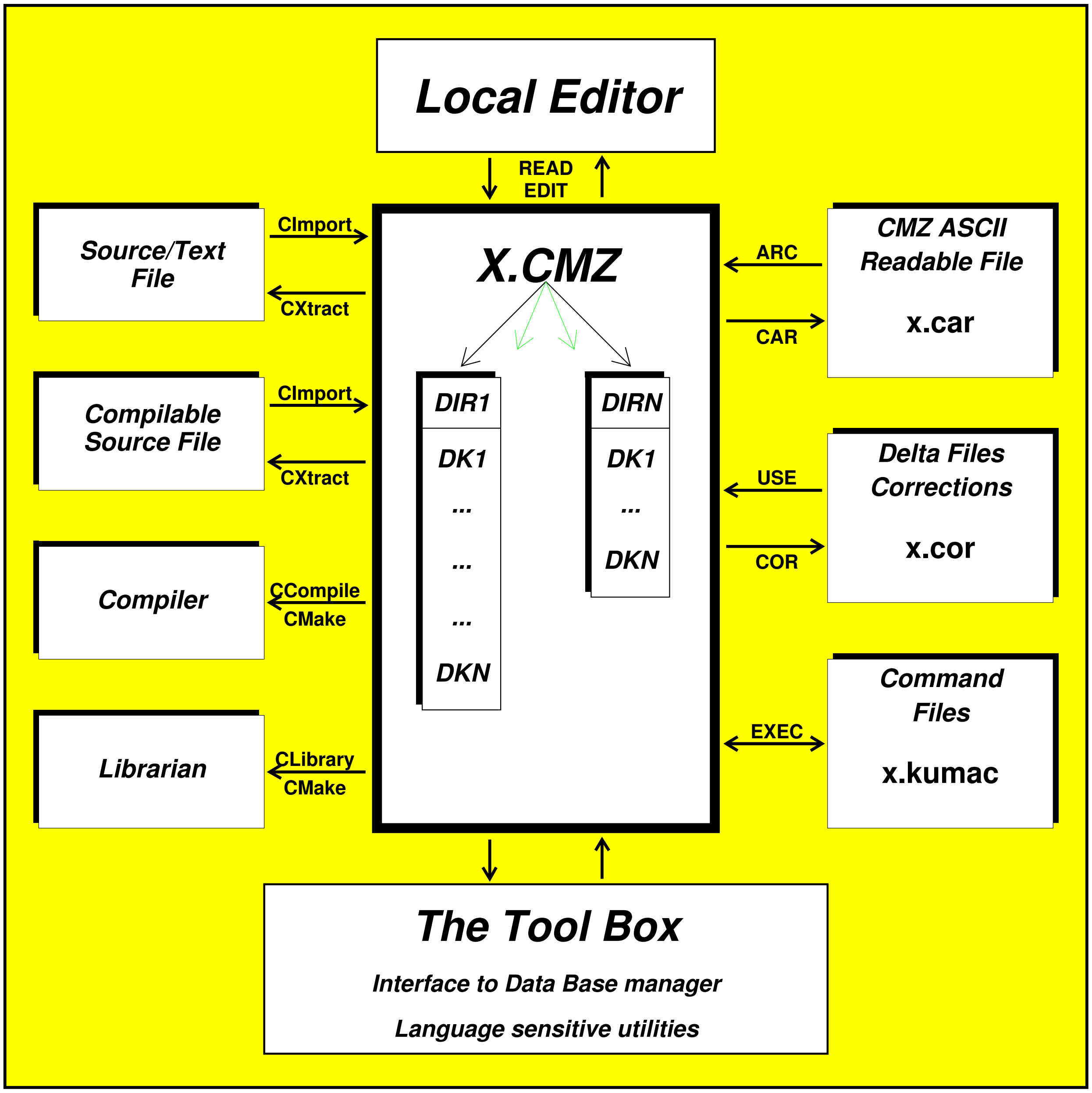
A Source Code Management System

CMZ

User's Guide & Reference Manual



Copyright Notice

Requests for information should be addressed to:

CERN Program Library Office	or	CodeME S.A.R.L.
CERN-CN Division		14 Rue de l'Eglise, Pouilly
CH-1211 Geneva 23		F-01630 St. Genis-Pouilly
Switzerland		France
e-mail: cernlib@cern.ch		e-mail: codeme@cern.ch

Version 1.49, December 19, 1996

Contents

Part I

User’s Guide

Chapter 1

Introduction

You are about to read about CMZ, an advanced, interactive, fast, self-documenting, customizable and machine-independent source code and document management system.

We call CMZ *advanced* because it offers code management facilities that go beyond simple bookkeeping and version archiving. CMZ supports the developer during the development phase of his[1] program: direct access to any routine in the local editor; check for undefined variables; automatic indentation of routines; renumbering of statement labels[2]; reordering of decks in alphabetical order; storage of related decks in directories; single command to compile a routine and update the object library; import and export of source and text files; capabilities to create procedures that automate the above mentioned operations. In the maintenance phase CMZ offers the user: an extensive versioning mechanism; generation of a correction set, by comparing different copies of the changed routines with the routines in the master library; using the correction set for generating a new object library; updating the master library by applying the correction set; automatic dependency detection and incremental build.

We call CMZ *interactive* because it provides a Unix-like shell in which you give commands. Every command gives you direct feedback. In case you enter a wrong command, argument or parameter you get an appropriate error message and in case of successful completion you get a new prompt. CMZ also provides a *batch mode* to execute a series of commands contained in a file.

CMZ is *fast* because the source code is stored in a RZ direct-access file. This makes it possible to directly access any routine for editing, copying, compiling, searching, etc. Up to 50 direct-access files (CMZ library files) can be attached to CMZ concurrently in the same session without degradation in access time. RZ is the database management part of the ZEBRA[3] [4] package.

Self-documenting means that at any time you can type `HELP` *command* to find out how *command* works, or just `HELP` to browse through the list of commands (see section 2.5). This facility is offered to you by the KUIP[5] user interface package.

Customizable means that you can redefine or rename CMZ commands. For example, you can define the Unix-

[1] In the rest of this User's Guide *he/his* can freely be replaced by *she/her*.

[2] These last three functions work only for FORTRAN code.

[3] Hence the origin of the name CMZ: Code Management system using Zebra.

[4] *ZEBRA User's Guide (3.53)*, R. Brun, M. Goossens, J. Zoll, CERN PROGRAM LIBRARY Q100, 1987.

[5] *KUIP User'sGuide (2.04)*, CERN PROGRAM LIBRARY I102, 1993.

like `ls` command to list the contents of a directory instead of the CMZ command `DIR` (see section 8.2.1). You can write entirely new commands using the KUIP MACRO facility. In a MACRO you can combine a number of commands to act as one new command (see section 6.1).

CMZ is *machine-independent*. This means that the operation of CMZ is completely independent of the underlying hardware. For example, the CMZ command to compile a routine behaves the same on any machine. The CMZ library files are stored in a machine independant format and can be transferred via binary FTP or can be NFS, AFS ,... mounted.

Currently CMZ is running on the following platforms: IBM (VM/CMS and MVS), DEC (VMS, OSF and Ultrix), Cray Unicos, Sun (SunOS and Solaris), Silicon Graphics IRIX, Gould UTX, Apollo Domain/OS, HP-UX and IBM RS6000/AIX. A PC/MSDOS version and a WINDOWS/NT on ALPHA/PC version are also available.

1.1 Notation and Conventions

In this manual, commands and other strings that must be typed in literally are shown in a typewriter font, like `PURGE`. Source code directives, file, patch, deck and other names are also in a typewriter font but surrounded by apostrophes, like '`+PATCH,AAP`'. Placeholders such as command argument names are in a slanted font, like *argument*. Square brackets ('`[ ]`') enclose optional items in command descriptions. A vertical bar ('`|`') OR's items in a list of choices, and brackets ('`( )`') group items in a list of choices.

In commands uppercase and lowercase characters are generally equivalent, so `SHELL` is the same as `shell`. Commands may be abbreviated until they become ambiguous, e.g. `sequences` may be abbreviated to `seq`, not, however, to `se` (ambiguous with `select`). In case you give an ambiguous command a list of possible commands will be shown.

1.2 Organization of the CMZ Documentation

The CMZ documentation consists of two parts: the User's Guide and the Reference Manual. The User's Guide gives an in-depth description of most of the CMZ commands. The behavior of, and the context in which every command should be used is described, as well as side-effects and features. The User's Guide contains the following chapters:

Chapter 2 explains what you need to know to enter and exit CMZ. It tries to give you an idea of CMZ's capabilities by showing you a sample session.

Chapter 3 describes the structure of the CMZ library file[6] as well as the structure of the CMZ Ascii readable file and the source code directives.

Chapter 4 describes the CMZ commands most often used in the code development phase.

Chapter 5 contains a description of the commands most often used during the code maintenance phase.

Chapter 6 describes the macro facility.

[6]From here on we will call a CMZ library file just a CMZ file.

The Reference Manual is a copy of the on-line CMZ Reference Manual. The Reference Manual gives of every command a terse description followed by some examples showing how to use the command.

CMZ uses the same source code directives (see section 3.3) as the batch oriented PATCHY source code manager[7] developed at CERN. This makes it trivial to convert an existing PATCHY card image file into a CMZ file. In Appendix A.3 the differences between the CMZ and PATCHY directives are documented.

An extensive study of how to use CMZ in a distributed, heterogeneous environment of a large High Energy Physics collaboration was made by members of the H1 experiment in DESY, Hamburg. Their conclusions and guidelines can be useful to other people and collaborations and are therefore included in Appendix B of this manual.

[7] *PATCHY Reference Manual (4.09)*, H. Klein, J. Zoll, CERN PROGRAM LIBRARY.

Chapter 2

Getting Started

2.1 Entering CMZ

Invoking CMZ is the only machine dependent part of CMZ because the passing of command line arguments differs from machine to machine.

2.1.1 UNIX Systems

To invoke CMZ just type `cmz` at the shell.

Here are the allowed CMZ arguments:

'`-n`'	Do not read your cmzlogon file '`cmzlogon.kumac`' (see section 2.3).
'`-l` *logon*'	Read logon commands from the file '*logon*`.kumac`'. The '`-l`' and '`-n`' options are mutually exclusive.
'`-r [`*restore*`]`'	Restore the environment of the previous CMZ session. By default the file '`cmzsave.dat`' will be read. To change the name of the restore file see 4.12. This option is mutually exclusive with all other options.
'`-b` *batchjob*'	Run CMZ in batch mode. The commands CMZ has to execute are read from the file '*batchjob*`.kumac`'. The '`-b`' option implies also the '`-n`' option.
'`-install` *file* `[`*opts*`]`'	Execute the install macro in the file '*file*`.car`' or '*file*`.cmz`' (see section 5.7).
'`-x` *file*`[`*mname* `[`*opts*`]]`'	Execute the default flogon macro (or the macro mname with option opts) in the file '*file*`.car`' or '*file*`.cmz`' (see section 5.8).

Whenever you have to specify a filename in CMZ you can use the standard Unix filename syntax, like '`/user/rdm/test.f`'. The '`~`' as in '`~bonapart/idsim.cmz`' and in '`~/delphi/delgra.car`' are interpreted in the usual Unix way. Environment variables at the beginning of a filename are substituted by their associated value, e.g. '`$HOME/pipo.f`' will become '`/u/xx/rdm/pipo.f`' when '`$HOME=/u/xx/rdm`'.

2.1.2 VAX & ALPHA VMS

To invoke CMZ just type `cmz` at the VMS prompt.

Here are the allowed CMZ arguments:

‘`/NOLOG`’	Do not read your cmzlogon file ‘`CMZLOGON.KUMAC`’ (see section 2.3).
‘`/LOGON=`*logon*’	Read logon commands from the file ‘*logon*`.KUMAC`’. The ‘`/LOGON`’ and ‘`/NOLOG`’ options are mutually exclusive.
‘`/RESTORE[=`*restore*`]`’	Restore the environment of the previous CMZ session. By default the file ‘`CMZSAVE.DAT`’ will be read. To change the name of the restore file see section 4.12. This option is mutually exclusive with all other options.
‘`/BATCH=`*batchjob*’	Run CMZ in batch mode. The commands CMZ has to execute are read from the file ‘*batchjob*`.KUMAC`’. The ‘`/BATCH`’ option implies also the ‘`/NOLOG`’ option.
‘`/INSTALL=`*file* `[,`*opts*`]`’	Execute the install macro in the file ‘*file*`.car`’ or ‘*file*`.cmz`’ (see section 5.7). Note that options must be separated by a comma.
‘`/X=`*file* `[,`*mname*`[,`*opts*`]]`’	Execute the default flogon macro (or the macro mname with option opts) in the file ‘*file*`.car`’ or ‘*file*`.cmz`’ (see section 5.8). Note that options must be separated by a comma.

Whenever you have to specify a filename in CMZ you can use the standard VAX VMS filename syntax, like ‘`DISK$DELPHI:[RDM]TEST.FOR`’.

2.1.3 IBM VM/CMS

To invoke CMZ just type `cmz` at the CMS prompt.

Here are the allowed CMZ arguments:

‘`(NOLog`’	Do not read your cmzlogon file ‘`CMZLOGON KUMAC`’ (see section 2.3).
‘`(LOGon=`*logon*’	Read logon commands from the file ‘*logon* `KUMAC`’. The ‘`(LOGon`’ and ‘`(NOLog`’ options are mutually exclusive.
‘`(REStore[=`*restore*`]`’	Restore the environment of the previous CMZ session. By default the file ‘`CMZSAVE DAT`’ will be read. To change the name of the restore file see section 4.12. This option is mutually exclusive with all other options.
‘`(BATch=`*batchjob*’	Run CMZ in batch mode. The commands CMZ has to execute are read from the file ‘*batchjob* `KUMAC`’. The ‘`(BATCH`’ option implies also the ‘`(NOLOG`’ option.
‘`(INStall=`*file* `[,`*opts*`]`’	Execute the install macro in the file ‘*file*`.car`’ or ‘*file*`.cmz`’ (see section 5.7). Note that options must be separated by a comma.
‘`(X=`*file* `[,`*mname*`[,`*opts*`]]`’	Execute the default flogon macro (or the macro mname with option opts) in the file ‘*file*`.car`’ or ‘*file*`.cmz`’ (see section 5.8). Note that options must be separated by a comma.

Whenever you have to specify a filename in CMZ you can use the standard VM/CMS filename syntax except that the blanks in the filename should be replaced by dots. The default disk is your A disk. For example, ‘`AAP CARDS A`’ should be specified in CMZ as ‘`AAP.CARDS`’ and ‘`PIPO CMZ D`’ should be specified as ‘`PIPO.CMZ.D`’.

```
*
*-- logon file for project AAP
*
ALIAS/CREATE LS    DIR   -C
*
AUTHOR   Louis Daniels
*
FILE     AAP
SELECT   VAX
SEQ      AAPCDES VAX
SET      AAP.OLB -L
*
MESSAGE  Welcome to the AAP development system
```

Figure 2.1: *Example of a* `cmzlogon` *file.*

2.2 Exiting CMZ

To terminate CMZ, type `EXIT`. This command releases in a correct way all files connected to the current CMZ job and writes the files '`lastcmz.kumac`', containing the last 50 commands issued during the just terminated session, and '`cmzsave.dat`' (see section 4.12) which contains the complete environment (connected files, compile options, sequence definitions, aliases, etc.) of the just terminated session.

One can also suspend a CMZ session. Suspending means creating an subshell and stopping CMZ temporarily, allowing you to resume CMZ later, after exiting the subshell, with the same files, same compile options, same sequence definitions, and so on.

To suspend CMZ, type `SHELL`. `SHELL` actually creates a subshell that communicates directly with the terminal and CMZ waits until you exit the subshell. Before creating the inferior shell CMZ tells you how to exit this shell (since this may be machine dependent). For example, `C-d` (i.e., CONTROL-D) or `exit` are typical commands to exit a subshell in Unix.

On windowing systems there is no need to suspend CMZ. Typically you keep CMZ always running in one window, while other windows can be used for what ever else you need to do.

2.3 The '`cmzlogon.kumac`' File

When you start CMZ it tries to read the file '`cmzlogon.kumac`'[1] in your current working directory[2]. This logon file may contain commands which are typically typed at the beginning of each CMZ session. Initially this file could contain something like:

```
MESSAGE   Welcome to the AAP development system
```

[1] '`kumac`' stands for **KU**IP **mac**ro, in the rest of this User's Guide we omit the extension '`.kumac`' from files containing KUIP macro's.

[2] If the operating system supports the notion of directories (Unix, VMS), otherwise it just tries to read the available '`cmzlogon`' file in the user's default file area.

A more intelligent 'cmzlogon' file could look like the one in Figure 2.1. This logon file first defines some aliases (see section 8.2.1), next it sets the name of the author (see section 4.2 and 7.1.1), connects the CMZ file 'aap.cmz' (see section 4.4 and 7.2.2), selects the VAX control option (see section 4.7 and 7.1.2), loads the sequences defined in directories AAPCDES and VAX (see section 4.8 and 7.1.3), sets the default library to 'aap.olb' (see section 4.12 and 7.1.4) and says hello to the user (see section 8.1.5).

If there is no 'cmzlogon' file in your current working directory then CMZ tries to read a default logon file. The place where your default logon file should reside is machine dependent. For the following systems the location of the default logon file is:

Apollo Domain/OS	'~/user_data/cmzlogon.kumac'.
UNIX systems	'~/.cmzlogon.kumac'.
VAX & ALPHA/VMS	You have to define (in your 'LOGIN.COM' file) the logical name 'CMZ$LOGON' to point to the desired default logon file.
IBM VM/CMS	'CMZDEF KUMAC A'.

In case your default logon file does not exist CMZ will inform you about this fact before proceeding.

2.4 The 'cmzsys.kumac' File

Before reading the 'cmzlogon.kumac' file, CMZ tries to read the optional file 'cmzsys.kumac'. Typically this file is written by the group manager to provide a similar environment to all developers in the same project.

On UNIX and Apollo systems, if the environment variable '$CMZSYS' pointing to the desired file is not defined, CMZ looks for the '$HOME/cmzsys.kumac' file.

On VMS systems, CMZ looks first for the logical name 'CMZ$SYS' and, if it is not defined, for the 'SYS$LOGIN:CMZSYS.KUMAC' file.

On IBM VM/CMS system CMZ, looks for the 'CMZSYS KUMAC *' file.

2.5 On-line Help

CMZ features on-line help for every command. To find out how a command works simply type HELP *command*. This gives a full description of the command's functionality, its arguments and options. The reference manual is a copy of the on-line help. To get a one line reminder of a command type USAGE *command*.

2.6 A Sample Session

As an introduction to CMZ, let us consider the following example. Start by invoking CMZ:

```
$ cmz
**** CMZ Code Manager version  1.45/17  (01/03/94) ****
```

```
CMZ [1] AUTHOR  Louis Daniel
CMZ [2] help CREATE
CMZ [3] help MKDIR
CMZ [4] help DIR
CMZ [5] CREATE  analysis
analysis[6]  MKDIR  code
analysis[7]  CD  code
analysis/code[8]  DIR
Current Working Directory = //ANALYSIS/CODE
   00_PATCH;1
Total:      1 Decks     1 Cycles
analysis/code[9] help CIMPORT
analysis/code[10] CIMPORT  analysis.f
Number of DECKS   read   :  4
Number of RECORDS written:  428
analysis/code[11] DIR
Current Working Directory = //ANALYSIS/CODE
   00_PATCH;1   HWABEG;1     ANALYS;1     HWAEND;1     FILLHIST;1
Total:      5 Decks     5 Cycles
analysis/code[12] help VERSION
analysis/code[13] VERSION  -S  1.00/00
VERSION: Directory $VERSION created
VERSION: Creating new TITLE deck
analysis/code[14] help HOST_ED
analysis/code[15] HOST_ED 'vi &'
analysis/code[16] EDIT new
EDIT: New deck new
analysis/code[17] DIR
Current Working Directory = //ANALYSIS/CODE
   00_patch;1   hwabeg;1     analys;1     hwaend;1     fillhist;1
   NEW;1
Total:      6 Decks     6 Cycles
analysis/code[18] ALPHA_ORDER
analysis/code[19] DIR  -A
Current Working Directory = //ANALYSIS/CODE
  DECKNAME             CYCLE   VERSION   DATE/TIME  BYTES  LINES
  00_PATCH                1*   1.00/00 940119/1145     68      2
  ANALYS                  1*   1.00/00 920402/ 119   7472    226
  FILLHIST                1*   1.00/00 920402/ 124    952     36
  HWABEG                  1*   1.00/00 920402/ 118   3032    116
  HWAEND                  1*   1.00/00 920402/ 123   1592     54
  NEW                     1            940119/1151    112    153
Total:      6 Decks     6 Cycles      437 Lines
analysis/code[20] help SET
analysis/code[21] SET  *.f -F F77
analysis/code[22] CCOMPILE
Number of DECKS   written:  5
Number of RECORDS written:  596
analysis/code[23] help SHELL
analysis/code[24] SHELL
$
```

We are back to our shell. We can check that eleven new files have been created in our working directory:

```
analysis.cmz   analys.f    analys.o   fillhist.f   fillhist.o   hwabeg.f
hwabeg.o       hwaend.f    hwaend.o   new.f        new.o

$ exit

analysis/code[23] EXIT

All connected CMZ files are now released

**** CMZ normal termination ****
```

2.7 The CMZ Prompt

If you have read through the examples of the previous section you have probably noticed that the CMZ prompt changed after every command. The number in the prompt is the *command number*. This command number can be used to re-execute commands via a Unix C-shell-like history mechanism. Starting a command with an exclamation mark followed by:

- An unsigned integer n will re-execute the n^{th} command entered since the beginning of the session (e.g. `!3`).
- A negative number $-n$ will re-execute the command identified by the current command number minus n (e.g. `!-11`).
- Another exclamation mark will re-execute the previous command (e.g. `!!`).
- Anything else (i.e. a non-numeric string) will re-execute the latest command entered which starts with the specified string (e.g. `!EDIT`).
- Nothing else will show the list of recallable commands (e.g. `!`).

For a more sophisticated command line recall and editing mode see the command `RECALL_STYLE` (section 8.3.9).

The part of the prompt before the command number reflects the current file and working directory.

Chapter 3

The CMZ File Structure

3.1 Structure of a CMZ Library File

CMZ views the whole pool of text to be managed as a set of nodes in a hierarchical data structure (Figure 3.1). The structure of the CMZ library file is created and maintained by the ZEBRA RZ package. A CMZ library file has in general the filename extension '.cmz'.

The leaf nodes, known as *decks* in CMZ terminology, correspond to the original material (compilable units, document chapters, etc.) and constitute the basic units of information addressed in a CMZ library file.

We show in Figure 3.1 the complete structure of a sample file `EXAM1`: at the first level the material used to identify the file constitutes a deck whose name is '`TITLE`'; at this level you also find the directories (also named *patches* in the CMZ terminology), there are 5 in this example: '`CDES`', '`EEN`', '`TWEE`', '`$VERSION`' and '`$KUMAC`'. Version and history information are contained in the directory '`$VERSION`' while command procedures (i.e. macro scripts) to extract the material or set up the working environment are contained in the directory '`$KUMACS`'.

Each deck is identified by a unique *pathname*. The full pathname of a deck consists of four elements[1]: the CYCLE number, the DECK name, the PATCH name and the FILE name. The full pathname is constructed following the Unix conventions (with a little touch of VMS) for naming directories and files, e.g.: `//FILE/PATCH/DECK;2`

The highest level component of the pathname is called the *root*, denoted by '`//`'. The CMZ filename is the *top directory*. The top directory is not part of the pathname stored into the CMZ file, i.e. renaming the filename changes the top directory name.

The CYCLE number allows for the existence of multiple versions of the same decks. In Figure 3.1, the deck '`NOOT`' in patch '`EEN` 'appears with cycle number `1` while deck `AAP` in the same patch appears with cycle number `3`. When attaching another CMZ file, this file will be added to the root as another top directory, thus allowing the simultaneous availability of files which can contain decks and patches with identical names. Up to **50** CMZ files may be attached simultaneously.

The maximum length of a directory (patch) name is **16** characters and the maximum length of a deck name is

[1]Although RZ allows for unlimited nesting of directories, CMZ currently allows only one level of directories (i.e. patches). In future updates, we foresee to remove this restriction.

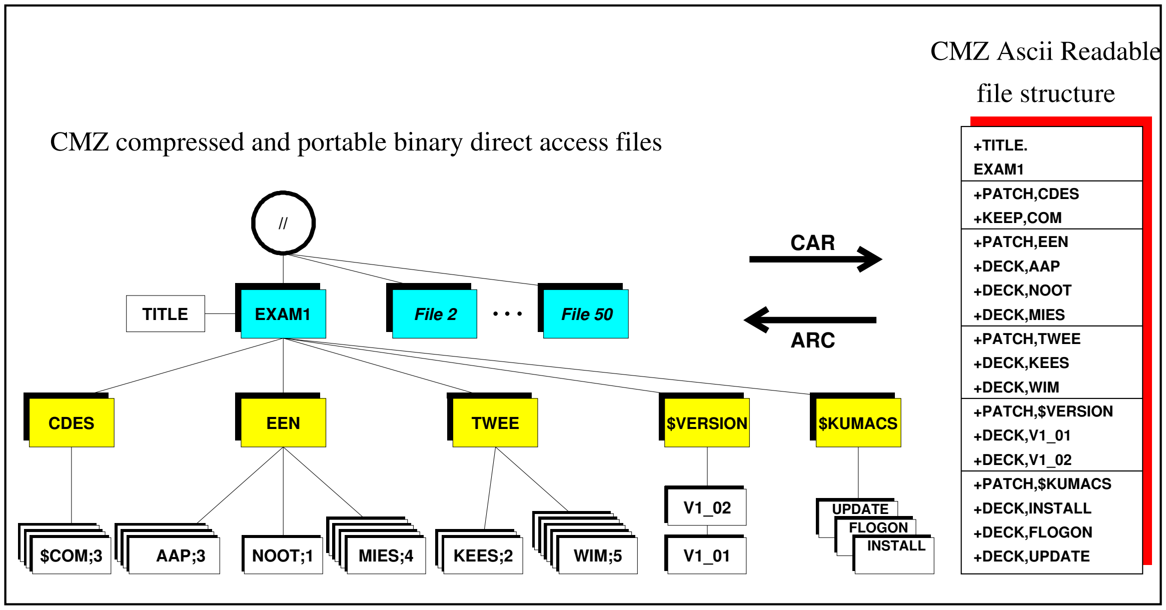


Figure 3.1: *The CMZ file with its hierarchical database structure can be mapped onto the CMZ Ascii Readable file via the command CAR and viceversa via the command ARC.*

32 characters. Longer names are truncated to the allowed maximum lengths.

For maximum access speed to any version of a deck CMZ stores full copies of old versions of decks. To compensate for the increased disk usage caused by this scheme, CMZ saves space by using compression algorithms[2].

Because of data compression, the CMZ database is stored on disk as a binary file. As different vendors support different kinds of binary files, this can result in an incompatibility situation that usually requires byte swapping client-server protocols to access a file across a heterogeneous network. The CMZ program, however, handles itself the data conversions for all supported platforms. This means that a CMZ library file can simply be copied from one platform to another without the user having to perform a conversion and that it can also be directly accessed across a heterogeneous network via NFS, AFS, etc. (Figure 3.2).

The CMZ file structure can be mapped (and its content transcripted) onto a special file called CMZ Ascii Readable file (shortly, CAR file) (see Figure 3.1 and 3.2).

3.2 Structure of a CMZ Ascii Readable File

A CMZ Ascii Readable, CAR, file is an Ascii transcription of a CMZ library file. In this section we will describe the general structure of a CAR file. A CAR file has in general the filename extension '`.car`'. A compressed and portable direct access binary file is typically about 3 to 5 times smaller than the corresponding CAR. A CAR file can be produced from a library file using the command `CAR` (see section 7.4.2). Viceversa, a library file can be produced from a CAR file using the command `ARC` (see section 7.3.1).

[2] Jean-loup Gailly, Mark Adler, Peter Jannesen and Haruhiko Okumura. Gzip algorithm from Free Software Foundation

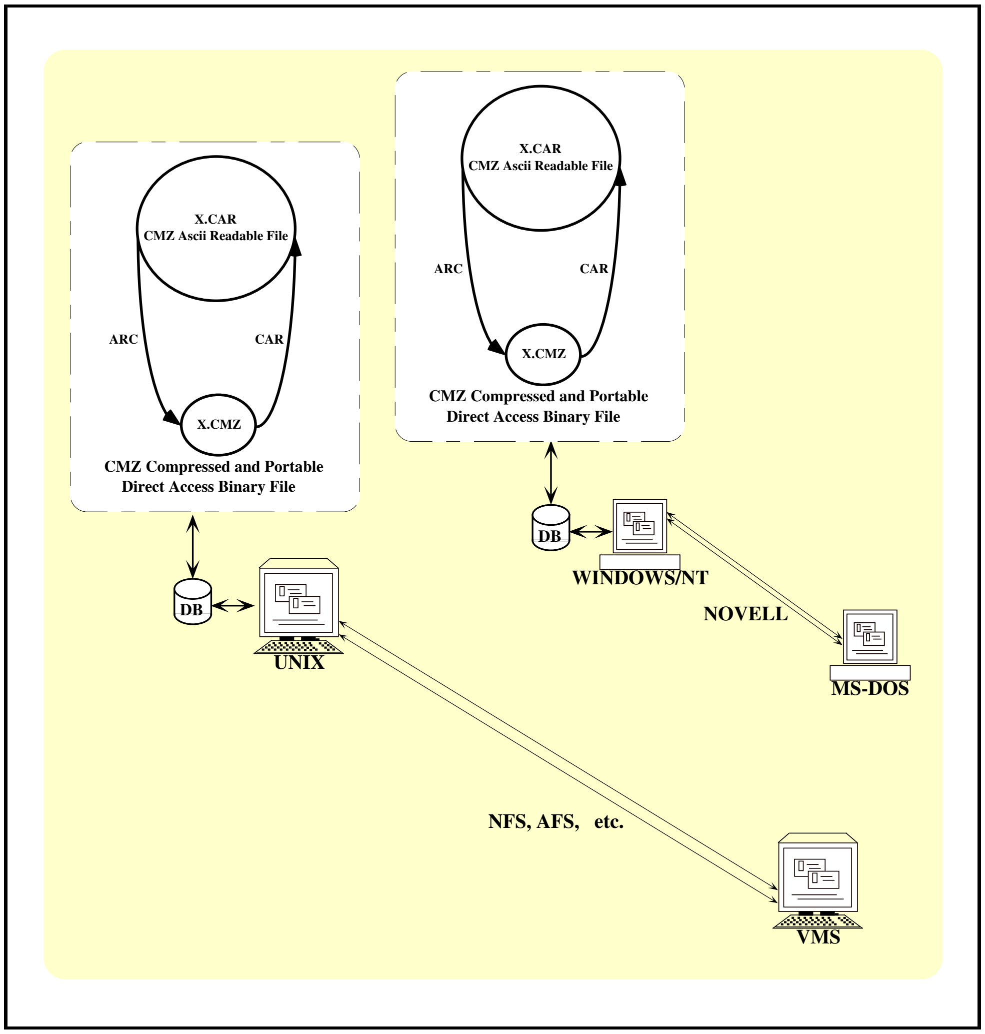


Figure 3.2: *The CMZ Compressed and Portable Direct Access Binary file 'x.cmz' residing on a machine with a given file management system can be accessed without previous conversion from a machine with a different file system. Two different situations are shown (link of Unix and VMS, Windows/NT and MS-DOS) that are easily handled with CMZ.*

A CAR file is divided into divisions called *patches*. A patch may be further sub-divided into *decks*. The beginning of a new deck is marked by the directive '+DECK,*dname*.'; the directive '+PATCH,*pname*.' marks the beginning of the new patch '*pname*'. '*Pname*' and '*dname*' are the identifiers of patches and decks. Decks consist of a number of lines each not longer than **255** columns. Patch identifiers must be unique in the CAR file. Deck identifiers must be unique in the patch.

In a CAR file code is grouped into decks, usually with one subroutine per deck in case of Fortran, one file per deck in case of C/C++, one chapter in case of documents, etc. To reduce confusion it is advised to use the subroutine/procedure/file names also as deck names. Decks which logically belong together are grouped into a patch.

A sample CAR file is shown in Figure 3.3. In this CAR file we see the six patches 'DCDES', 'INIT', 'WORK', 'CWORK', '$VERSION' and '$KUMACS'. Three of them contain source code written in Fortran and one of them code written in C. Patch 'DCDES' for the *sequence definitions* (see section 3.3.4), patch 'INIT' for the initialization routines, patch 'WORK' and 'CWORK' for the processing and output routines, patch '$VERSION' to store version and history information and patch '$KUMACS' for command procedures to extract the material or set up the working environment. Patch 'INIT' has two subroutines in two decks, deck 'START' contains the initialization part of the program and deck 'FOPEN' contains a machine independent file open and read routine (the machine dependence is taken care of via the '+IF' directives). Patch 'WORK' has two subroutines in two decks. Every patch may contain up to **1000** decks.

The patch 'DCDES', in the example, is the *sequence patch*, it contains the definitions of all those sequences of code providing the Common-Dimension-Equivalence (or any other) statements for use everywhere in the program. In this example, for Fortran source code you assign one labelled common block to one sequence name:

```
+KEEP,DCONST.
      INTEGER       K1,K2
      COMMON/DCONST/K1,K2
```

Any deck in the program which needs at least one variable of /DCONST/ calls for this COMMON statement with the directive '+SEQ,DCONST'. This makes it easy to change the contents of a common block, which will happen often in the development phase of a program.

In the case of C language code, the deck in the sequence patch contains the type definitions and global data. These sequences will be expanded in include files.

Machine dependent (or version, e.g. debug version dependent) code can also be inserted in the code itself (including KEEP definitions) using the '+IF' directive. This mechanism can be used to select different variants of code. For example:

```
<general code>
+IF,VAX.
<VAX dependent code>
+ELSEIF,APOLLO.
<Apollo dependent code>
+ENDIF.
<general code>
```

The machine dependent code that will be used depends on which option was set with the SELECT (see section 7.1.2) command.

```
+TITLE
DEMO    1.34/00  900806  17.04  CMZ demonstration file
+PATCH,$KUMACS.
+DECK,INSTALL.
      macro  install  machine=x
       ...
      return
+PATCH,$VERSION.
+DECK,V1_34.
       ...
+PATCH,DCDES.
+KEEP,DCONST.
      INTEGER       K1,K2
      COMMON/DCONST/K1,K2
+KEEP,DSTORE.
      PARAMETER (NWORDS=50000)
      INTEGER NWORDS,LPOINT,LHITS
      REAL    WS
      COMMON/DSTORE/WS(NWORDS),LPOINT,LHITS
+PATCH,INIT.
+DECK,START.
      SUBROUTINE START
+SEQ,DCONST.
      K1=1
      K2=2
       ...
      END
+DECK,FOPEN.
      SUBROUTINE FOPEN
+SEQ,DSTORE.
+IF,IBM.
      OPEN(1,FILE='/FNAME FTYPE',STATUS='OLD')
+ELSE.
      OPEN(1,FILE='FNAME.FTYPE',STATUS='OLD')
+ENDIF.
      READ(1,1000)WS
       ...
      END
+PATCH,WORK.
+DECK,COMPUT.
      SUBROUTINE COMPUT
+SEQ,DCONST,DSTORE.
      DIMENSION SUM(NWORDS)
      SUM(1)=WS(1)
      DO 10 I=2,NWORDS
         SUM(I)=SUM(I-1)+WS(I)
10    CONTINUE
      END
+DECK,RESULT.
      SUBROUTINE RESULT
+SEQ,DCONST,DSTORE.
      PRINT *,'WS(K1)=',WS(K1),'WS(K2)=',WS(K2)
      END
+PATCH,CWORK,T=C.
+DECK,CUPAD,IF=UNIX.
#include <string.h>
void cupad_(char *name ,int len_name)
{
  char fname[256];
  strncpy( fname, name, len_name );
  fname[len_name] = '\0';

  ku_pad(fname,1);
}
```

Figure 3.3: *Example layout of a CAR file.*

3.3 CMZ Source Code Directives

The next sections describe the CMZ source code directives and their allowed parameters and options.

3.3.1 Format of Source Code Directives

The source code directives are flagged by a '+' in column 1 and identified by the first 3 letters of the directive in columns 2–4, trailing characters of the directive are ignored, the directive is separated from the first parameter by a comma. Some examples:

```
+PATCH, AAP, IF=SUN.
+KEEP, AAPCOM.
+SEQ, AAPCOM, NOOTCOM, MIESCOM.
+IF, VAX, SUN.
+ELSEIF, DECS.
+ELSE.
+ENDIF.
+DECK, NOOT.
```

All source code directives may be in upper-case or lower-case.

Lines with a '+' in column 1 but unknown directives are treated as ordinary lines. Blanks are significant in columns 1–4.

CMZ processes the information on a directive up to and excluding the '.' or column **255**, whichever comes earlier, ignoring blanks.

The columns following the terminating '.' up to column 255 are called the comment field of the directive. For example:

```
+IF,APOLLO.  Following code is for the Apollo only
```

In some circumstances *multiple parameters* are accepted, e.g.:

```
+SEQ, AAPCOM, NOOTCOM, MIESCOM.
```

The directives `+PATCH`, `+DECK`, `+KEEP` and `+SEQ` may be made conditional on the status of some options (set by the `SELECT` command, see section 4.7 and 7.1.2) by adding *IF parameters*. These IF parameters must be the *last* parameters on the line. For example:

```
+PATCH, AAP, IF=APOLLO.
```

The logical operators :	',' (comma) or '|' (pipe)	for OR
	'&' (ampersand)	for AND
	'-' (minus)	for NOT

are allowed in conjunction with the IF parameters, which are written in general as follows:

```
..., IF=p1|(p2 & p3)
```

The operators are analysed from the left to the right. Parenthesis are required to break the left to right analysis. For example:

' ..., IF= p1 | p2 & p3 '	is equivalent to	' ..., IF= (p1 | p2) & p3 '
	but different from	' ..., IF=, p1 | (p2 & p3) '

The p*i* are options, possibly preceded by '`-`' to indicate the logical NOT. The truth-value of a particular name 'p*i*=*pname*' is 'true' if the '*pname*' compile option is set by the `SELECT` command and 'false' otherwise; for 'p*i*=–*pname*' the value is 'true' if the flag '*pname*' is **not** set by `SELECT`. A compile option may consist of up to **32** alphanumeric characters.

Associated material belonging to a rejected directive is skipped. For example:

```
+PATCH, AAP, IF=VAX.
<patch material>
```

the material of patch '`AAP`' will only be used if the compile option '`VAX`' has been set with the `SELECT` command.

3.3.2 The '`+TITLE`', '`+PATCH`' and '`+DECK`' Directives

The directive

```
+TITLE.
```

starts a CAR file. The *title deck* must contain at least one line. Its first line being the CAR title consisting of the CAR file identifier. Leading blanks or an initial '`*`', '`C`' or '`c`' followed by blanks are ignored. The identifier should be followed by a version number (for the version format see section 5.1), followed by the date, time and a comment. For example:

```
+TITLE.
PAW      2.05/01  28/03/94  08.42.11  PROGRAM LIBRARY PAW  = Q121
```

The '`+TITLE`' directive may be omitted. In that case the first line of the CAR file will be treated as the title. A correctly formatted title deck is normally made by giving the command `EDIT title` in the top directory of a CMZ file (see section 4.9).

The directive

```
+PATCH, pname [,T=...]  [,IF=...].
```

starts patch '*pname*'.The patch identifier '*pname*' may consist of up to 16 alphanumeric characters. To denote that all decks in a patch are modules in a given language L the option '`T=`*L*' should be added to the '`+PATCH`' directive, where L has to be defined via the `SET` command. A '`+PATCH`' directive is made by the command `MKDIR` (see section 4.15). The actual '`+PATCH`' directive is stored in the deck '`00_PATCH`' and this deck should be edited to change the '`+PATCH`' directive.

The directive

```
        +DECK, dname [,T=...]  [,IF=...].
```

starts deck '*dname*'. The deck identifier '*dname*' may consist of up to 32 alphanumeric characters and must be unique in the patch. By adding option 'T=*L*' to the '+DECK' directive a deck can be denoted to contain material in language L. A '+DECK' directive is normally made by the command EDIT (see section 4.9) and can be changed while being in the editor.

The whole patch or deck is skipped if the result of the 'IF=...' is 'false'.

3.3.3 The Conditional Directives '+IF', '+ELSEIF', '+ELSE', '+ENDIF'

The general IF construction

```
       +IF, ...
          statements
       +ELSEIF, ...
          statements
       +ELSE.
          statements
       +ENDIF.
```

controls a section of conditional material, which is skipped if the IF-result is 'false' or is accepted if 'true'. In section 3.3.1 the logical operations are explained.

A '+SEQ' directive, i.e. sequence calls within conditional material can be part of an IF block construction. The conditional directives can be nested and indented. To indent, place the symbol '_' (underscore) between the '+' in column 1 and the directive 'IF,' , 'ELSEIF,', 'ELSE' or 'ENDIF'.

An example showing the use of the directive of the '+IF' group is shown in Figure 3.4.

3.3.4 The '+KEEP' and '+SEQ' Directives

Sequences are strings of zero, one or more lines, which once loaded by the SEQUENCE command can be called up any number of times for inclusion in any deck. They have to be *defined* in CMZ as sequence material and *recognized* as such by CMZ at the moment of building the compilable source file. To allow CMZ to *recognize* sequences, they have to be preceded a +KEEP directive.

For instance, in Fortran, two distinct types of usage are made of sequences: they are used to provide Common-Dimension-Equivalence (CDE) statements for the decks of a program and they are used to provide optional code for places where it is demanded.

The directive

```
+DECK,CREAD.
*CMZ :  1.44/11 29/07/93  11.50.46  by  Nancy Drew
*-- Author :    Nancy Drew   30/09/88
      SUBROUTINE CREAD(EDECK)
*
+SEQ,CMZPAR,CMLUN,CMLIST,CREADO.
*
      CHARACTER*(*)      EDECK
      CHARACTER*(LGLINE) KLINE
*
      IF (NDECK.EQ.0) THEN
+IF,NEWLIB.
         CALL ITOFT(LUNSCR,FILNAM,IERR)
+ELSEIF,IBMMVS.
         FILNAM = 'CMZ.BROWSE.LIST'
+ELSE.
         I1 = 1
         IF (EDECK(1:1).EQ.'$') I1 = 2
+_IF,APOLLO,CRAY,(UNIX & -MSDOS).
         FILNAM = '/tmp/'//EDECK(I1:LENOCC(EDECK))//'.TMP'
         CALL CUTOL(FILNAM)
+_ELSE.
         FILNAM = EDECK(I1:LENOCC(EDECK))//'.TMP'
+_ENDIF.
*
+ENDIF.
*
+IF,-IBM.
         CALL CMOPEN(LUNSCR,FILNAM,' ',IERR)
+ELSE.
         CALL CMOPEN(LUNSCR,FILNAM,'N',IERR)
+ENDIF.
         IF (IERR.NE.0) RETURN
      ENDIF
*
      CALL WDECK(LUNSCR,EDECK)
*
      END
```

Figure 3.4: *Example of the usage of the '*`+IF`*' directive.*

```
+KEEP, sname [,IF...].
```

starts the definition of a sequence with identifier '*sname*'. The maximum length of a sequence identifier is **32** characters. The sequence material is terminated by the next CMZ directive other than '`+SEQ`' (see paragraph below) or '`+IF`' (see paragraph above). Thus the definition of a sequence '*sname*' may include other sequences, even sequences which are not yet defined; *quoted* sequences are not called until '*sname*' itself is called. Multi-level inclusions are allowed (with a maximum of **10** levels). Also the sequence definition may contain optional sequence material. Sequences may be *called* into a program with the directive

```
+SEQ, sname [,s2, ...]  [,IF=...].
```

When writing the source code, if several sequences are called, substitution occurs in the order indicated on the line. If a sequence is called which has not been loaded by the `SEQUENCES` command (see section 4.8 and 7.1.3), a 'missing sequences' diagnostic is given.

A deck may contain several sequence definitions or each sequence definition can be stored in a separate *keep deck*. A keep deck is a deck with a '+KEEP' directive in place of a '`+DECK`' directive and is edited by typing `EDIT $`*sname* (for more see section 4.9). After saving and exiting the editor, keep decks will be created for each sequence definition. However, prior to doing this you have to check that the sequence name(s) are unique. The following will cause the creation of one deck with two cycles:

```
+KEEP,AAP,IF=APOLLO.
.....
+KEEP,AAP,IF=HPUX.
.....
```

This can be rewritten like:

```
+KEEP,AAP.                                   +KEEP,AAP1.
+IF,APOLLO.                                    Apollo specific code
  Apollo specific code          or           +KEEP,AAP2.
+ELSEIF,HPUX.                                  HPUX specific code
  HPUX specific code                         +KEEP,AAP.
+ENDIF.                                      +SEQ,AAP1,IF=APOLLO.
                                             +SEQ,AAP2,IF=HPUX.
```

Of course it is possible to have several sequence definitions in one deck. In this case it stays possible to have `IF` selections on sequences with identical names. However, the `cmake` command will work most efficiently when keep decks are used, because in that case the modification date and time of every individual sequence definition is available and only those decks referencing the changed sequences will be recompiled.

After editing a deck containing sequence definitions, CMZ will automatically (re)load all definitions into memory.

3.4 Fortran and C/C++ Code Specific Sequences

3.4.1 The DATEQQ, TIMEQQ, VERSQQ, AUTHQQ and QFTITLCH Sequences

The following sequence names are *reserved* by CMZ for use in Fortran source code:

```
+SEQ, DATEQQ.     is replaced by
␣␣␣␣␣␣IDATQQ=yymmdd
+SEQ, TIMEQQ.     is replaced by
␣␣␣␣␣␣ITIMQQ=hhmm
```

These Fortran statements make the date and time of the CMZ run available to the program.

```
+SEQ, QFTITLCH,N=n.  is replaced by the Fortran continuation line
␣␣␣␣␣+␣<first n characters of the CMZ title>
```

thereby making the file identifier and title, as stored in the '`TITLE`' deck (see section 3.3.2) in the top-directory of the CMZ file, available to the program at execution time. The first `n`<63 significant characters of the title are transmitted; if the parameter '`N=n`' is absent, `n=8` will be used. A leading '`C␣`' will be stripped off from the CMZ title.

```
+SEQ, VERSQQ.     is replaced by
␣␣␣␣␣␣VERSQQ='vv.rr/ll'
␣␣␣␣␣␣IVERSQ=vvrrll
```

thus making the version number, as stored in the '`TITLE`' deck, available to the program at execution time. '`VERSQQ`' should be defined as a '`CHARACTER*8`' variable in the routine that calls '`+SEQ,VERSQQ`'. '`IVERSQ`' contains the same version number in integer form.

```
+SEQ, AUTHQQ.     is replaced by
␣␣␣␣␣␣AUTHQQ='Author␣Name'
```

thus making the author name, as defined via the '`AUTHOR`' command, available to the program at execution time. '`AUTHQQ`' should be defined as a '`CHARACTER*32`' variable in the routine that calls '`+SEQ,AUTHQQ`'.

Equivalently, the following sequence names are *reserved* by CMZ for use in source code written in C/C++:

```
+SEQ, DATEQQ.     is replaced by
#define IDATQQ=yymmdd
+SEQ, TIMEQQ.     is replaced by
#define ITIMQQ=hhmm
```

These C/C++ statements make the date and time of the CMZ run available to the program. Analogously, for the version number and the author name:

`+SEQ, VERSQQ.` is replaced by

```
#define VERSQQ "vv.rr/ll"
#define IVERSQ vvrrll
```

`+SEQ, AUTHQQ.` is replaced by

```
#define AUTHQQ "Author Name"
```

3.4.2 The VIDQQ Sequence.

The '`+SEQ, VIDQQ`' sequence is expanded to a line consisting of the following three fields.

1. The name of the package as written on the TITLE deck.
2. The version number and the date/time when the file was versioned.
3. The date/time when the extract file was generated.

The `VIDQQ` sequence as implemented in CMZ generates a static variable for Fortran or C/C++ programs. This static variable contains the magic characters recognized by the Unix command 'what'.

Example:

```
(hpcmz) [173] what cmz
cmz:
    CMZ      1.45/17  01/03/94  10.01.23  C: 01/03/94  10.02.29
    ZEBRA    3.72 /00 06/09/93  16.00.00  C: 08/12/93  11.09.26
    KUIP     2.04/08  16/02/94  14.16.39  C: 01/03/94  10.00.47
(hpcmz) [174] what paw
paw:
    PAW      2.04/13  28/02/94  18.50.06  C: 28/02/94  18.52.30
    yaccpar   1.9 (Berkeley) 02/21/93
    COMIS     1.17/10  13/01/94  12.16.3  C: 16/02/94  16.29.48
    HBOOK    4.21/09  28/02/94  18.55.39  C: 28/02/94  18.57.36
    KUIP     2.04/09  01/03/94  11.27.08  C: 01/03/94  14.10.01
    ZEBRA    3.72 /00 06/09/93  16.00.00  C: 08/12/93  11.09.26
    MINUIT   2.05/01  23/07/91  13.00.00  C: 09/12/93  08.44.57
    HPLOT    5.18/11  01/03/94  09.10.55  C: 01/03/94  09.11.49
    HIGZ     1.20/11  28/02/94  11.26.52  C: 28/02/94  11.44.08
```

Chapter 4

Using CMZ in the Code Development Phase

In this chapter we will explain how to use CMZ most effectively during the code development phase. The order in which commands are described in the next sections of this chapter, will give you a feeling of when, where and how to use a specific command. Figure 4.1 gives an overview of the commands and the files they act upon. Only the commands that need more explanation are described in this chapter. See the Reference Manual for a description of all commands.

4.1 CMZ Command Syntax

The command syntax of CMZ is Unix-like. This means that options always start with a '`-`' followed by a single character. For some commands the order of the arguments is significant. While for others it is not. To find out the correct syntax of a command, see the Reference Manual or the on-line help.

4.1.1 CMZ Pathnames

A full pathname is constructed by glueing together with slashes, '`/`', the root, '`//`', the filename, the patch name, the deck name and, preceded by a '`;`', the cycle number. Omitting the root and file name makes a pathname relative to the current file. Omitting also the patch name results in a pathname relative to the current file and patch. Omitting the '`;`' and the cycle number from the pathname defaults to the highest cycle of the specified deck. When in a sub-directory (patch), the parent directory (i.e. the top-directory (file)) can be specified by '`..`'. Some examples:

`//top1/patch3/deck2;4`	full pathname of the 4$^{\text{th}}$ cycle of '`deck2`' in '`patch3`' in file '`top1`'
`/patch3/deck2;4`	pathname of the same deck as above relative to file '`top1`', i.e. you must already be somewhere in file '`top1`'
`deck2;4`	pathname of same deck as above relative to '`patch3`', i.e. you must already be in sub-directory '`patch3`'
`../patch4/deck5`	pathname of highest cycle of '`deck5`' in the sister directory '`patch4`' of the current working directory

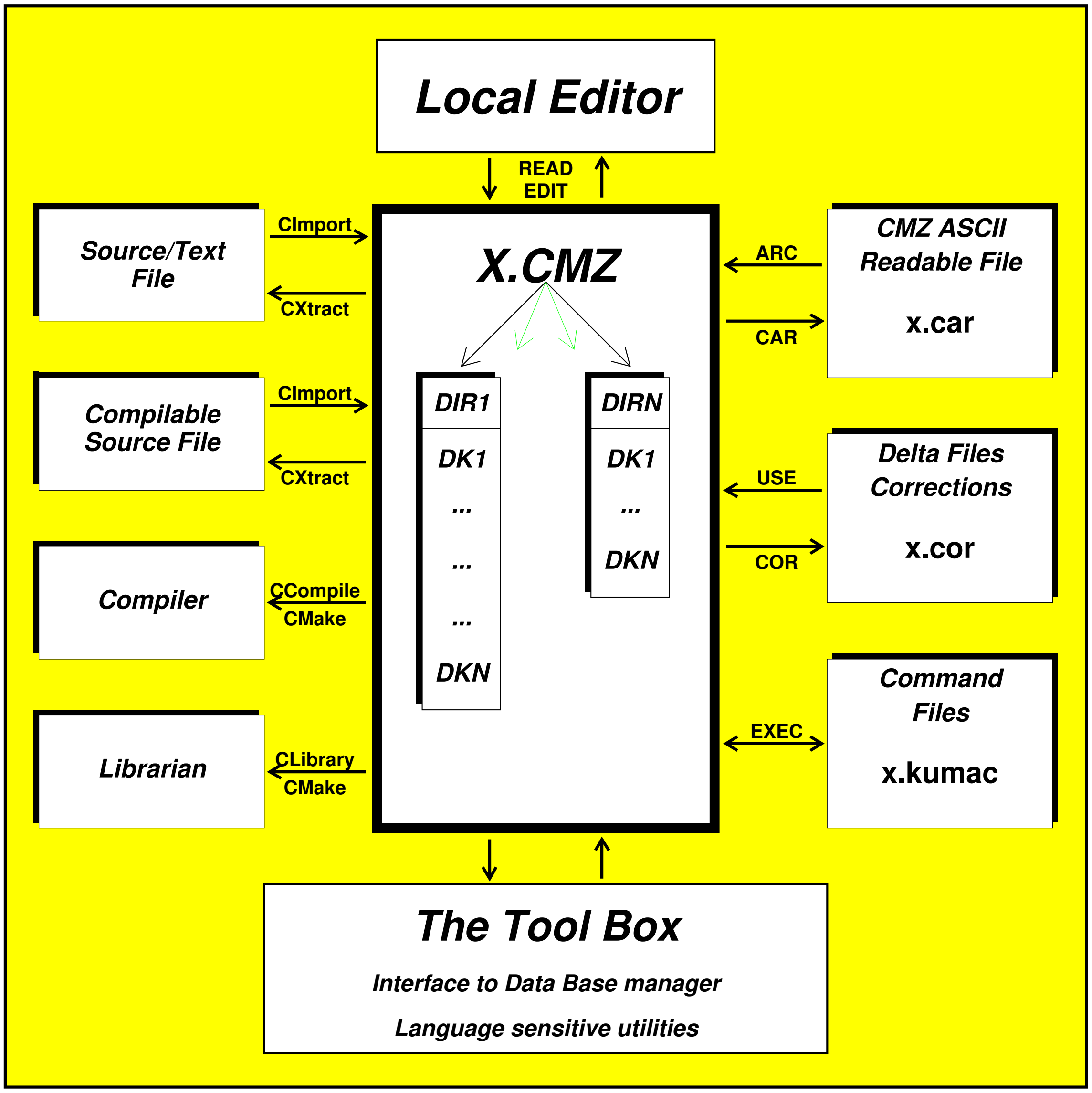


Figure 4.1: *Overview of CMZ commands and the files they act upon.*

4.1.2 The Decklist

A number of commands allow as an argument a *DECKLIST*. The *DECKLIST* argument allows you to specify *one or more* sets of decks. A set of decks is specified by giving the first and last deck of the set separated by a '`.`' (the first and last deck are included in the set), e.g. '`deck1.deckN`'. If '`deck1`' is the first deck in the CMZ current working directory (CWD) than '`deck1`' may be omitted. If '`deckN`' happens to be the last deck in the CWD then '`deckN`' may be omitted. If a set contains only a single deck name then wildcarding is permitted (for the wildcarding syntax see section 7.5.11). The first deck of the set may be given in the general form '`//top/patch1/deck1`', which sets the CWD for the entire set to '`//top/patch1`'. A set which includes all decks in the CWD may be specified by '`*`'. A set only contains the highest cycles of the specified decks unless the set consists of a single deck, in which case the cycle number may be added to the deck name. Instead of decks you may also specify sets of patches this way. Some examples:

`//top/patch2/deck1`	'`deck1`' in '`patch2`' in file '`top`'
`deck1`	'`deck1`' in the CWD
`deck1.`	from '`deck1`' included to the last deck in the CWD
`.deck2`	from the first deck to '`deck2`' included in the CWD
`deck3.deck6`	from '`deck3`' to '`deck6`' included in the CWD
`*`	all decks in the CWD or all decks in all patches if the CWD is a top directory
`//top/patch2.patch4`	from '`patch2`' to '`patch4`' included in file '`top`'
`//top1 //top2`	all decks in all patches in the files '`top1`' and '`top2`'
`//`	all decks in all patches in all connected files

4.1.3 The Date

If a command expects a *DATE* you can give the date/time in the following format: '`dd/mm/yy.hh.mm`'. The date and time can be specified separately by '`dd/mm/yy`' and '`hh.mm`', respectively. Leading zeroes in the date and time may be omitted. The date may also be specified by the following reserved words: '`TODAY`', '`YESTERDAY`', '`MONTH`' and '`LASTMONTH`'.

Some valid date specifications are:

`2/6/88.17.15`	2nd of June 1988 quarter past 5 pm.
`9.15`	today quarter past 9 am.
`MONTH`	from the 1st of the current month

4.2 Making Yourself Known to CMZ

Before doing anything in CMZ you should make yourself known to CMZ. Not just as a matter of politeness, but because CMZ's history mechanism keeps track of changes by putting your name in the history records of every deck you edit.

`AUTHOR` [*username*]

With this command you tell CMZ your name. The *username* argument may contain blanks. If you omit the argument, `AUTHOR` will tell you the current author name. The most convenient place for this command is in the '`cmzlogon`' file.

4.3 Creating a CMZ Library File

Before any material can be stored, a CMZ library file must be created.

`CREATE` [-C] *cmz_file* [*nrecs*]

`CREATE` creates a CMZ file and sets the current working directory to the top of this new file. The file will get the extension '`.cmz`' if you do not specify an extension. By default a newly created CMZ file can contain up to 16 Mb of data (i.e. 32000 records of 512 bytes). If you think you are going to store more than that into the file you can override the default by setting *nrecs* to the amount of (512 bytes) records you need. If the option `-C` is set, decks entered into the file will be compressed using the gzip algoritm. Typical compression factors range from 3 to 6.

After creating the CMZ file you can store data into it using the commands `ARC`, `CIMPORT`, `FTOC` or `TTOC`. Instead of reading data from a file you also can create patches (directories) and decks interactively with the commands `MKDIR` and `EDIT`.

4.4 Connecting and Releasing a CMZ Library File

`FILE` *cmz_file* `[-R] [-X [`*macro* `[`*args...*`]]]`
`RELEASE` *cmz_file* `| *`

Use `FILE` to connect an already existing CMZ file. The current working directory will be set to the top of the newly connected file. The default file extension is '`.cmz`' and need not be specified. If you give the optional parameter `-R` the file will be connected in *readonly* mode. Use the *readonly* mode when you want to connect CMZ files for which you don't have write permission, e.g. CMZ files from other users. If you do not specify the `-R` option and CMZ detects that the CMZ file is write protected, it will automatically connect the file in *readonly* mode. Up to **50** CMZ files may be connected at the same time.

If option `-X` is specified a macro, stored in the CMZ file, will automatically be executed once the file has been connected. By default the macro in deck `/$kumacs/flogon` will be executed. In deck `flogon` one typically stores the file dependent equivalent of the `cmzlogon.kumac` file. For more on macro execution see command `EXEC` (section 7.5.5).

A CMZ file can be disconnected from CMZ with the `RELEASE` command. All connected files can be released at once by giving the argument `*`. When leaving CMZ with `EXIT` all connected files will be released automatically.

4.5 Importing and Exporting Code and Text Files

A CMZ file may contain source code (FORTRAN, C and others) as well as plain text. The material that can be imported by CMZ can be either in CAR, FORTRAN, C, plain text or any other format.

CMZ can output the contents of the CMZ file in CAR, pure FORTRAN, pure C or any compilable language (i.e. compile-ready, with sequences and conditional material resolved) or plain text format.

4.5.1 Importing and Exporting CAR Files

`ARC` *car_file* `[-C[C]][-A][-S] [`*decklist*`]`
`CAR` *car_file* `[-H[nn][L]][-A][-V` *vers_num*`][-W] [-B | $USE |` *decklist*`]`

The command `ARC` is used to import a CAR file into the CMZ file. With the *decklist* argument (see section 4.1) you specify the set of patches and/or decks you want to import from *car_file*. Omitting *decklist* results in the import of the whole CAR file. Patches will become directories in the CMZ file with the decks as the elements. In every directory a deck '`00_PATCH`' will be created containing the '`+PATCH`' directive. If the CAR file does not contain history records the following default history records will be added to every deck[1] after the '`+DECK`' directive:

```
*CMZ :          01/06/88  18.02.57  by  Louis Daniels
*-- Author :
```

If you set option `-A` in `ARC` then also the author field of the history records will be filled in:

```
*CMZ :          01/06/88  18.02.57  by  Louis Daniels
*-- Author :    Louis Daniels   01/06/88
```

If the CAR file already contains history records (from a previous CMZ run) no new history records will be added. For more information on the history mechanism see section 4.11. If you give the `-C` option `ARC` will strip off anything appearing in the columns 73–80 from all statements that are not comment lines [2]. When you set option `-CC` the comments in columns 73–80 are stripped off from all lines.

Use command `CAR` to export the patches and/or decks specified either with *decklist* or option `-B` in CAR format. The selected patches and/or decks will be written onto the file *car_file*. In case the file *car_file* already exists it will be renamed to *car_file*`.bak` (thereby overwriting an already existing *car_file*`.bak`). If you use option `-B` all deck names stored in the *decklist buffer* (see section 7.5.2) will be used. When `$USE` is given as decklist then all the decks referenced in the USE bank (see section 5.3) are processed. If *decklist* or option `-B` are omitted all patches and decks in the CWD will be exported. To write only the decks belonging to version *vers_num* (for the format of *vers_num* see section 5.1) to the CAR file use option `-V` *vers_num*. If you set option `-H` then the decks will be written onto the CAR file without their history records. This might be useful when the CAR file will be shipped to end-users who are not interested in the history comments. Stripping off the history records can also save a considerable amount of space (assuming you elaborately documented all your changes in the history records). If you use option `-H` and a deck is missing its '`*-- Author :`' card, you will get a message that this card is missing. No history records will be stripped off in that case. If option `-HL` is set only the last CMZ comments are written to the CAR file. If option -Hnn, where nn is an integer less than 100, is set then the last nn CMZ comments with version numbers are written to the CAR file.

When you give the `-A` option in `CAR`, all cycles of all decks will be written onto the CAR file. This is mainly useful when you want to export the CMZ file without losing the old versions of decks. A CAR file created with the `-A` option can be read back into CMZ by `ARC` just like any other CAR file. The options `-H` and `-A` may be concatenated into `-AH` or `-HA`, they must, however, precede the *decklist*.

[1] Except to the '`00_PATCH`' decks which will only get the '`*CMZ :`' record after the '`+PATCH`' directive.

[2] Option `-C` or `-CC` are Fortran77 features where comment lines are lines starting with a '`*`', '`C`' or a '`c`'.

When the `-W` option is given, the command is equivalent to the command `CTOY` (see section 7.4.3) which converts a CMZ file into a CAR file compatible with Patchy.

4.5.2 Importing and Exporting source files

```
CIMPORT [-T language][-K][-[I]S][-A] files
CXTRACT [-H[L]][-U][-R] (-P | [-P][-V vers_num] -B | [-K] $USE | -F car_file |
        [-P][-N][-V vers_num] decklist)
```

With command `CIMPORT` you can import one or more source files into a CMZ file. Each file will be stored in a single deck. The deck name will consist of the first 32 characters of the source filename. The deck will be created in the CWD. The default history records will be added to the new deck (see section 4.5.1). The effect of option `-A` is identical as for the command `ARC`. Decks created by the `CIMPORT` command will get on the '`+DECK`' directive the option '`T=`*language*' if the option `-T` is specified or '`T="current_language"`' if the option `-T` is not specified. This helps the export commands to recognize the type of the decks. If the option `-K` is set, a '`+KEEP`' header line is created instead of a '`+DECK`' directive.

If you set option `-S` then CMZ will search for matches between the sequence definitions currently stored in memory (see section 4.8) and identical lines in the source file. When a match is found, the corresponding lines will be replaced by a '`+SEQ,`*sname*' line, where *sname* is the name of the sequence in memory. When option `-IS` is set only include lines are replaced.

The command `CXTRACT` is used to export the compile-ready code. All '`+SEQ`' sequences and '`+IF`' conditionals will be resolved and the code will be written on the default extract (source) file (see section 4.12). For a detailed description of the options see section 4.13.

4.5.3 Importing pure FORTRAN Code

```
FTOC fortran_file [-S][-C[C]][-A] [routinelist]
```

With command `FTOC` you can import pure FORTRAN source code. With the optional argument *routinelist* you can specify the routines to be imported. The syntax of *routinelist* is the same as the *decklist* syntax (see section 4.1.2) except that no pathname may be specified. If *routinelist* is omitted the whole *fortran_file* will be read in. Each routine will be stored in a single deck. The deck name will be the same as the routine name (truncated to 32 characters). If there is a blank blockdata, it will be stored in the deck '`BLOCKDAT`'. The decks will be created in the CWD. To each deck the default history records will be added (see the previous section). The effects of the options `-C`, `-CC` and `-A` are identical as for the command `ARC`.

If you set option `-S` then CMZ will search for matches between the sequence definitions currently stored in memory (see section 4.8) and identical lines in the FORTRAN file. When a match is found, the corresponding lines will be replaced by a '`+SEQ,`*sname*' line, where *sname* is the name of the sequence in memory.

4.5.4 Importing and Exporting Plain Text Files

```
TTOC [-W][-A] text_files
CTOT [-S][-U][-R][-H][-D][-V vers_num] (-P | -B | [-K] $USE | [-N] [decklist])
```

With the command TTOC you can import plain text files into the CMZ file. The deck name will consist of the first 32 characters of the text filename (minus the filename extension, e.g. if the filename is 'exam.tex' then the deck name will become 'exam'). The deck will be created in the CWD. The default history records will be added to the new deck (see section 4.5.1). The effect of option -A is identical as for the command ARC. By default any line longer than 255 columns will be truncated. To prevent any truncation specify option -W to enable line-wrapping. Decks created by the TTOC command will get the option 'T=TEXT' on the '+DECK' directive. This makes it possible to output all text decks in a single go (CTOT -D *, see below).

The command CTOT is used to export decks to plain text files. Of each deck the '+DECK' card and the history records will be stripped off, unless option -H is set in which case only the '+DECK' card will be stripped off. The text file will have the deck name extended by the file extension *.ext* as set with the SET command (see section 4.12). The default extension is '.txt'. The decks to be exported can be specified either with *decklist* or by the options -B, -P or $USE. If option -P is given then all patches and decks selected by the PILOT (for creating a pilot option list see section 4.7) command will be scanned for the type specifiers 'T=TEXT' or 'T=DATA' The decks found will be written to the default text file. If you set option -B all deck names stored in the *decklist buffer* (see section 7.5.2) will be exported. If you specify option $USE all decks referenced in the USE bank (see section 5.3) will be exported. Option -D tells CTOT to write only patches or decks of type 'T=TEXT' or 'T=DATA' to the default text file. If option -V *vers_num* is set then only the decks belonging to version *vers_num* will be written to the default text file (for the format of *vers_num* see section 5.1). If option -N is set the decks will be taken from the USE bank (the decks must be new decks not yet in the CMZ file). By default the corrections stored in the USE bank will be taken into account. If you don't want any correction cards to be used set option -U. The options -N and -U and $USE and -U are mutually exclusive. If option -R is specified all decks are removed from the USE bank after processing. When option -S is given the '+IF' directives will be interpreted according to the active control options. The -S option is especially useful if you have a deck which contains, for example, a machine dependent installation script. After setting the desired control option (with SELECT) the example of Figure 4.2 could be extracted with the command CTOT -S INSTALL (for more elaborate installation decks and automatic installation procedures see command EXEC in sections 5.7 and 7.5.5). The options -S, -U, -H, -R and -N may be concatenated, they must, however, precede options -B, -P, $USE or the *decklist*.

4.6 Wandering through the CMZ Library File

Once you've created a CMZ file and imported some material you can move from directory to directory and list the directory contents with the following commands:

CDIR [*pathname*]
DIRECTORY [-L|-A] [*pathname*]

Use command CDIR to move from directory to directory. Specify with *pathname* a new CWD (see for the pathname format section 4.1.1). To go to another directory in the same file it is enough to give the directory name without a preceding '/' or '../'. If *pathname* is omitted then CDIR prints the full pathname of the CWD. Every time the CWD changes the prompt changes to reflect the new CWD.

With command DIR you get a listing of the contents of a directory. DIR first lists all the sub-directories of the CWD (because of the structure of the CMZ file only the top-directory can have sub-directories[3]) followed by the decks stored in the CWD. If there are more than **15** sub-directories they will be listed in rows. If there are less sub-directories or if option -L or -A is set the sub-directories will be listed in a single column. If you give as first argument option -L then DIR will produce a more verbose listing of the directory. Of each deck the cycle number, the date/time of creation and the size in bytes will be shown. Option -A behaves like option -L

[3]Although RZ allows for unlimited nesting of directories, CMZ currently allows only one level of directories (i.e. patches). In future updates, we foresee to withdraw this restriction.

```
+DECK,INSTALL,T=TEXT.
*CMZ :  1.00/00 20/03/94  14.51.53  by  Bob Smith
*-- Author :    Bob Smith   20/03/94
*****************************************************************
*                                                               *
                  macro  install
*                                                               *
*****************************************************************

   set F77 -lan

+IF,APOLLO.
   set fixit.ftn -f
+ELSEIF,CRAY.
   set 'cft77 -o off -b $compfile.o $compfile' -c
   set fixit.f -f
+ENDIF.

   sel . $MACHINE fixit
   cfor /

+IF,APOLLO.
   set stripit.ftn -f
+ELSEIF,CRAY.
   set stripit.f -f
+ENDIF.

   sel . $MACHINE stripit
   cfor /

+IF,APOLLO.
   shell /com/bind -b fixit fixit.bin
   shell /bin/rm fixit.bin

   shell /com/bind -b stripit stripit.bin
   shell /bin/rm stripit.bin
+ELSEIF,CRAY.
   shell segldr -o fixit fixit.o
   shell /bin/rm fixit.f fixit.o

   shell segldr -o stripit stripit.o
   shell /bin/rm stripit.f stripit.o
+ENDIF.

return
```

Figure 4.2: *Example of a machine dependent install script.*

except that it also adds the version number , the number of lines and the compression factor of each deck, If *pathname* is omitted then the contents of the CWD will be listed. *Pathname* may be either a directory or a deck. If *pathname* is a directory then `DIR` lists the contents of that directory. If *pathname* is a deck, the highest cycle of that deck will be listed. Wildcarding (see section 7.5.11) is permitted (if wildcarding is used, the *pathname* is assumed to be a deck). The symbol `...` (ellipses) may be used in *pathname* to indicate that any number of directory names (including zero) may appear in its place. So, `DIR //file1/...` will list the contents of all directories in '`file1`' and `DIR /.../ff?*` will list all decks of which the name begins with '`ff`'.

By setting *pathname* equal to `//` you can obtain a list of all currently connected CMZ files.

4.7 Selecting Control Options

`SELECT [.] | [-][`*comp_opt*`]`

If there are any control options they must be set before any source code will be exported. The CMZ `IF` directives are solved according to the selected control options.

Use `SELECT` to set the control options. With *comp_opt* you can add a control option and with `-`*comp_opt* you can remove a control option from the option list. More than one option may be set or removed at a time. If no arguments are given, a list of all active control options will be shown. Use argument `.` to clear all options from CMZ's memory.

Use command `CLIST` to find out which control options are used in the different decks of your CMZ file (look for the 'CONTROL OPTIONS' index, see section 4.14 and 7.4.1). Ideally, the CMZ file should contain a deck which documents all different control options.

4.8 Loading Sequences

Before any decks containing '`+SEQ`' directives can be exported as pure code the sequence definitions must be loaded.

`SEQUENCES [.]|[-E]|[-R]|[-O]|[-F` *file*`]|[-P` *patch*`]| [-V` *vers1* `[-VREF` *vref*`]][-U][`*seq_patch*`]`

Use `SEQUENCES` to load sequence definitions into memory. `SEQUENCES` scans all decks in the sub-directory *seq_patch* for sequence definitions (denoted by the '`+KEEP`' directives). More than one *seq_patch* may be specified at a time. By default the corrections stored in the USE bank (see section 5.3) will be taken into account. If you set option `-U` then the contents of the USE bank will be ignored. If no arguments are given, `SEQUENCES` gives you a list of all sequence definitions currently stored in memory. If you give instead of *seq_patch* option `-E` then the full definitions of all sequences currently in memory will be shown in the local browser or pager. Option `-R` has the same effect as option `-E` except that all '`+SEQ`' directives will be resolved. Any missing sequence definitions will be detected. You can write all sequence definitions from memory to a file by using option `-F` *file*. The '`+KEEP`' directive will be replaced by '`*KEEP`' and all '`+SEQ`' directives will be resolved. Instead of to a file, the sequence definitions can also be written with `-P` *patch* to a patch. In that case the '`+SEQ`' directives will not be resolved. Use argument "." to clear all sequence definitions from CMZ's memory. When there are a lot a sequences in memory (read from a lot of different sequence patches) it is often easier to overwrite the changed definitions than to clear the sequence bank and reload all the definitions. To overwrite existing definitions specify the `-O` option. If `-O` is not specified, already defined sequences will not be replaced. Use option '`-V` *vers1*' to load sequences belonging to a certain version. If the option '`-VREF` *vref*' is given, all sequences which have changed between the reference version *vref* and the *vers1* version are marked internally for a later use in the code extraction step via the command '`cmake -V v1 -VD v2`'.

NOTE

Always set the required control options with `SELECT` (see previous section) before loading any sequence definitions. The control options are necessary to correctly interpret the '`+IF`' directives or any `IF=` parameter on '`+KEEP`' and '`+SEQ`' directive lines.

If the same sequence is defined more than once, the first definition will be kept, unless `-O` is specified.

To find out in which sub-directories sequence definitions are, use `CLIST` and look for the 'SEQUENCES' index (see section 4.14). Ideally, the CMZ file should contain a deck which documents all patches containing sequence definitions (see Figure 3.3). The command `SEQUENCES` with '`//`' as argument loads in memory all sequences referred in all connected files.

4.9 Editing Decks

One of the important features of CMZ is the possibility to have direct access to any deck in the local editor of your choice.

`EDIT [-V` *vers_num*`] (-B |` *decklist*`)`

Use `EDIT` to read into the local editor the decks specified by *decklist* or option `-B` (to change the default local editor to the editor of your choice see section 4.12). If you use option `-B` all deck names stored in the *decklist buffer* (see section 7.5.2) will be edited. If you set option `-V` only those decks belonging to version *vers_num* will be edited. All decks to be edited are written onto the temporary edit file '`cmedt.edt`' (you may change this default file name, see section 4.12) which is automatically read by the editor of your choice (see section 4.12).

WARNING

Never delete the '`+DECK,`*dname*' and '`+PATCH,`*pname*' directives. In case you accidently removed the (first) '`+DECK`' or '`+PATCH`' directive, you will be warned about this fact. If a '`+DECK`' directive is removed, the saved deck will be stored under the name '`BLANKDEK`'. To correct your mistake add to '`BLANKDEK`' a new '`+DECK`' card and delete the '`BLANKDEK`' (for deleting a deck see section 4.16). If a '`+PATCH`' card was removed the saved deck will be stored in the CWD. If the deck used to be in the CWD no further action is required. If the deck used to belong to another directory, however, you can correct your mistake by copying the deck to its original directory (for copying see section 4.15) and by deleting the deck from the CWD. The '*dname*' and '*pname*' arguments may, however, be changed to rename or reposition a deck into another directory. A better way of renaming or repositioning a deck is with the `COPY` command.

Also never delete the '`*-- Author :`' history record.

During the edit session CMZ is suspended until the editor is exited. When the edit file has been saved, CMZ will create a new cycle only for the decks that have been changed in the edit file. Complete decks will be stored in the new cycles and not only the changes (i.e. *delta's*) to the original. This is done to obtain maximum access speed to any cycle of a deck.

Multiple edit sessions can take place simultaneously if the kuip edit server, `kuesvr`, is running (see 7.1.5).

If `EDIT` is invoked with, as argument, a non existing deck name, '*new_deck*', then the edit file will contain the directive '`+DECK,`*new_deck*.' followed by the default history records:

```
*CMZ :          01/06/88  18.02.57  by  Louis Daniels
*-- Author :    Louis Daniels   01/06/88
```

which in turn may be followed by a *deck header template* (see command `SET`, section 4.12).

To create a keep deck, type `EDIT $`*sname*, where *sname* is the name of the sequence. When *sname* happens to be equal to one of the KUIP system functions (see 8.1.8) the sequence name has to be given between quotes, `' '`.

To create a title deck type `EDIT title` in the top directory. CMZ will create in this case a correctly formatted title deck. If you wish, you may add some more lines to the title deck. However, don't change the format of the version number and the date and time on the line after the '`+TITLE`' directive (behind the time you may change the line as you please).

NOTE

When creating a new deck, you can specify in the command all the informations you want on the '`+DECK`' directive. Example:

```
CMZ > edit 'newdk,T=C++,IF=PROJECT.   Comments'
```

4.10 Browsing a Deck

`READ [-V` *vers_num*`][-S][-H] (-B |` *decklist*`)`

The `READ` command provides another way to view the contents of a deck. Contrary to the `EDIT` command `READ` shows the *decklist* decks in the local pager or browser (in read-only mode). On Unix systems the current host pager can be defined via the environment variable `CMZPAGER`'. If `CMZPAGER` is not set, the value set by the Kuip command `HOST_PAGER` is used. On VMS systems `TYPE/PAGE` is used unless the logical name `CMZ$PAGER` is defined. The command options are the same as the ones for `EDIT`.

On machines running the X window system and on HP/Apollo computers the browsing of the decks happens asynchronously, i.e. CMZ can execute other commands while at the same time the decks stay on the screen, in their own windows. To make use of this facility on X systems you have to set the environment variable '`CMZPAGER`' to:

```
setenv CMZPAGER xterm     (csh)
export CMZPAGER=xterm     (ksh)
```

In this case CMZ will invoke `xterm -e view` to view the decks. A more explicit `xterm` command can be specified, like:

```
      export CMZPAGER='xterm -geometry 80x48 -bg black -fg yellow -e view'
```

In this case you should not forget the '-e view'. On HP/Apollo systems the asynchronous mode is default and no environment variables need to be specified.

4.11 The History Mechanism

CMZ's history mechanism records for every deck the date-time of modification and the name of the person who made the modification (for setting the author name see section 4.2). The history record will be put, automatically, directly below the '+DECK' card and has always the following form:

```
*CMZ :          01/06/88  18.02.57  by  Louis Daniels
```

Every time a new cycle of a deck is generated a new history record is written. The top-most history record, not yet version-stamped, will be overwritten by the new record, to prevent an unlimited growth of the history list. It is also possible to add your own comments to the history list. When in the editor add just below the '+DECK' directive your comments describing the modifications made to the deck. Each comment line should either start with a '*', a 'C' or a 'c'[4] and the first comment line should contain at least **10** non-blank alphanumeric characters. CMZ considers a comment line containing less than 10 characters useless and removes the comment line. In case you added comments a new history record will automatically be added just above your comments and below the '+DECK' directive. This history record, dating your comments, will not be overwritten the next time a new cycle of the deck is generated. To prevent history lists from becoming too long you should add comment lines only after you have made major changes to a deck. For example add some comment lines after the '+DECK' card of deck 'MIES':

```
+DECK,MIES.
* I added just this comment to note a change
* the comment may be more than one line long
*CMZ :          01/06/88  18.02.57  by  Louis Daniels
```

after saving the deck and editing it again the history records look like:

```
+DECK,MIES.
*CMZ :          08/01/90  03.05.27  by  Fons Rademakers
* I added just this comment to note a change
* the comment may be more than one line long
*CMZ :          01/06/88  18.02.57  by  Louis Daniels
```

The history records can be listed by the command:

LMODS [[H.][-]*author* [*from_date* | -V[I] *vers1*] [*to_date* | *vers2*]]

Command LMODS, without any arguments, lists the top-most history record of all decks in the CWD. If the CWD is the top-directory then also the top-most history record of all decks in the sub-directories will be listed. If argument *author* is specified then only the top-most history record of the decks last changed by *author* will be listed (if *author* is specified then the *from_date* and *to_date* arguments may be omitted). If you give ME as *author* then your name as set by the AUTHOR command (see section 4.2) will be used. To get a list of all authors that recently have made changes you have to set argument *author* to AUTHORS. If *from_date* is specified (in 'DATE' format, see section 4.1) then only the top-most history record of the decks modified from *from_date* onwards

[4]Or between /* ... */ pairs when in C language mode (see section 4.12 on how to change to C mode).

will be listed (you cannot omit argument *author* in this case, but you can replace it by a `*`). If you give *to_date* (also in '`DATE`' format) then only the top-most history record of the decks modified up to and including *to_date* will be listed (*from_date* can be given as `*`). To find out which deck has been changed last in the CMZ file give `LAST` as *from_date*. To list the complete history list of the decks, replace *author* by option `H` (note that this option is not preceded by the usual '`-`'). When only the full history list, introduced by user *author*, has to be listed precede *author* by `H.` (don't leave any space between the . and *author*). If *author* is preceded by a '`-`' then this will be interpreted as all users except *author*. If instead of *from_date* `-V` *vers1* is specified then all decks modified after version *vers1* will be listed (for the format of *vers1* see section 5.1). If *vers2* is also given then only the decks modified between *vers1* and *vers2* will be listed. To include *vers1* and *vers2* in the range use option `-VI`.

4.12 Changing Defaults

```
SET [filename (-LAN|-E|-F[E]|-L|-S|-D|-T[D]|-X[A][C][D[1][2]][L][P]|-C)]
HOST_EDITOR [editor]
HOST_SHELL [shell]
FILECASE [option]
```

Use `SET` to change the default filenames for the editor, extract (FORTRAN, C, text, etc.), object library, execution, template or save files. Note that the Filecase may be changed via the Kuip command FILECASE (see section 8.3.11). Also the default compiler directive and the language mode (FORTRAN , C or anything else) can be changed by `SET`. `SET` without arguments shows you the current defaults.

With option `-LAN` you define the output file names and the compiler directives for several languages (default is F77). When changing language mode the name of the compile file and the compiler directive change accordingly (see tables 4.1, 4.2 and 4.3).

When you issue the `EDIT` command the deck or decks to be edited are written on the edit file. By default this is the file '`cmedt.edt`'. You can change the name of this default edit file with: `SET` *new_edit_file* `-E`. For Unix systems a useful setting might be, e.g. `SET /tmp/cmedt.edt.$pid`, where `$pid` is a KUIP function that returns the current process id (see 8.1.8).

By using option `-F` you can change the name of the extracted source file (depending on the language mode), i.e. the file on which the compile-ready code will be written. The commands `CXTRACT`, `CCOMPILE` `CLIB` and `CMAKE` will write to this file (see next section). If the name of the source file contains a `*` then every deck will be written on a new file with the `*` being replaced by the deck name. For example:

`SET new_*.c -F C`	Write every deck on a new file '`new_`*deck*`.c`'.
`SET ftn/*.f -F F77`	Write every deck on a new file '*deck*`.f`' in the directory '`ftn`' (Unix case).
`SET [.ftn]*.for -F F77`	Idem as above but for VMS case.

This feature is especially useful for systems where routines need to be compiled separately to create an object library. When the source file name contains the string '`$cmzdir`' it will be replaced by the CMZ current working directory. For example: when the CWD is '`//project/dir1`' then '`src/$CMZDIR_*.C`' will be replaced by '`src/DIR1_*.C`'.
When the option `-FE` is set, the extracted code will contain special CMZ comments. These special comments allow CMZ to later re-import via the `SYNCHRONIZE` command the decks that were changed outside of the CMZ environment. This feature allows one to edit the extract files directly, while still being able to use the CMZ build commands.

With the option `-COM` you can define the comment character(s) corresponding to the language. Comment character cannot be redefined for C and F77 languages. CMZ needs to know this comment character when writing the CMZ history lines on the extract file. If CMZ does not know which comment character to use, the CMZ history lines are not written on the extract file avoiding a compilation error. For example:

`SET % -COM Latex`	set % as comment character for Latex language
`SET /* */ -COM C++`	set /* and */ as comment characters for C++ language

With the option `-INC` you can select the include option. When this option is active, the sequences are written onto include files instead of being developed inside the extract file. For example:

```
SET '#include "directory_path/$SEQ.exth"'-INC C
SET '\include {directory_path/$SEQ.exth}'-INC LATEX
```

`SET . -INC C`	disable the include option for C language mode

The string $SEQ is replaced by the name of the sequence when writing the deck onto the extract file. If the extension '.exth' is omitted, the default extension '.h' is taken. If the directory path is not set, the include file will be written in the same directory as the one of the extract files.

NOTE

For C/C++ language, the include option is set to: `SET '$SEQ.h' -INC C/C++`
It means that the include files will be generated automatically and written in the same directory as the one of the C/C++ extract files. Use `'SET . -INC C/C++'` to disable the writing of include files. In that case sequences will be expanded inside the extract files.

The object library name can be changed by using option `-L`. This file will be created or updated by the commands `CLIB`, `LIBRARY` and `CMAKE`. When the object library name contains the string '`$cmzfile`' it will be replaced by the CMZ filename corresponding to the CWD. For example: when the CWD is '`//project/dir1`' then '`$CMZFILE.OLB`' will be replaced by '`PROJECT.OLB`'. This feature is convenient when you are working with more than one CMZ file. Changing to another CMZ file automatically changes the object library name in which the compiled decks will be stored.

The commands `CCOMPILE`, `CLIB` and `CMAKE` prepare also an execute file which contains the necessary commands to compile the source file(s) and update the object library. With option `-X` it is possible to change the execute filename. Option `-X` may be followed by the following modifiers:

`A`	The execute file will be automatically executed by the commands `CCOMPILE`, `CLIB` and `CMAKE`.
`C`	A comment directive will be added before each compiler directive, this is useful for quiet compilers, i.e. compilers that don't tell you which routine they are compiling.
`D[1][2]`	The object file(s) will be deleted once they have been replaced in the object library. In case of `D1` the compiler source file(s) will be deleted. In case of `D2` the compiler source file(s) and the object file(s) will be deleted.
`L`	The librarian will be invoked after all calls to the compiler have been made. This is more efficient in case a large number of source files have to be compiled since the compiler will not be paged out of memory by the librarian every time an object file has to be replaced in the object library.
`P`	The option P prevents the setting of the PATH in the execution file.

```
      SUBROUTINE $DECK
************************************************************************
*                                                                      *
* Name : $DECK                                                       $#*
*                                                                      *
*                                                                      *
* IN:                                                                  *
*                                                                      *
* OUT:                                                                 *
*                                                                      *
* written by:  $AUTHOR, $DATE $TIME                                  $#*
*                                                                      *
************************************************************************
```

Figure 4.3: *Example of a deck header template.*

On exiting CMZ saves its current status (connected files, control options, sequence definitions, aliases, current working directory, etc.) on a file '`cmzsave.dat`'. This file will be read when CMZ is invoked with the restore option (see section 2.1). The restore option is especially useful on systems where you have to exit and enter CMZ often. The name of the save file can be changed with: `SET` *new_save_file* `-S`. To turn off the writing of the save file give as *filename* a '`.`'.

A template file can be specified by the options `-T` or `-TD`. A template file contains a deck header template. The deck header template will be inserted by `EDIT` every time a new deck is created. The template will appear just below the default history records. The strings '`$DECK`', '`$AUTHOR`', '`$DATE`' and '`$TIME`' are reserved and will be replaced by the deck name, the author name (as set by the `AUTHOR` command), the date and the time, respectively. The string '`$#`' acts as a `TAB` character. If option `-TD` is used then *filename* is considered to be a CMZ pathname. This gives you the possibility to store the template together with your code in the CMZ file. The default template file extension is '`.cmt`'. To unset a template filename give as *filename* a '`-`'. An example of a Fortran deck header is shown in Figure 4.3.

	Edit file	FORTRAN file	C file	Library file
Apollo	`cmedt.edt`	`cmfor.ftn`	`*.c`	`$cmzfile.lib`
Unix	`cmedt.edt`	`cmfor.f`	`*.c`	`$cmzfile.a`
VMS VAX/ALPHA	`CMEDT.EDT`	`CMFOR.FOR`	`*.C`	`$CMZFILE.OLB`
IBM VM/CMS	`CMEDT.EDT`	`CMFOR.FORTRAN`	`*.C`	`$CMZFILE.TXTLIB`
WINDOWS/NT	`cmedt.edt`	`*.f`	`*.C`	`$cmzfile.lib`
MSDOS	`cmedt.edt`	`*.f`	`*.C`	`$cmzfile.a`

	Text file	Execute file	Template file
Apollo	`*.txt`	`cmexec.exec`	-
Unix	`*.txt`	`cmexec.exec`	-
VMS VAX/ALPHA	`*.TXT`	`CMEXEC.EXEC`	-
IBM VM/CMS	`*.TXT`	`CMEXEC.EXEC`	-
MSDOS	`*.txt`	`cmexec.bat`	-
WINDOWS/NT	`*.txt`	`cmexec.bat`	-

Table 4.1: *Default filenames.*

The syntax for the filename argument, for all supported machines, is described in the sections on filenames in chapter 2. At startup the default filenames are as in table 4.1.

	FORTRAN Compiler Directive
Apollo	`ftn $compfile -b $compfile.bin -zero -save -indexl -pic -dbs`
HP-UX	`f77 -K +ppu -g -c $compfile`
Gould UTX	`fort -f -g -w -c $compfile`
Cray Unicos	`cft77 -ezh -o off -b $compfile.o $compfile`
Ultrix	`f77 -static -G 3 -g -c $compfile`
SunOS	`f77 -g -c $compfile`
SOLARIS	`/opt/SUNWspro/bin/f77 -g -c $compfile`
IRIX	`f77 -static -g -nocpp -c $compfile`
CONVEX	`fc -fi -g -c $compfile`
IBM 6000	`xlf -qextname -qrndsngl -qcharlen=32767 -g -c $compfile`
Alliant	`fortran -c -nc -save -Og -w -g $compfile`
VAX VMS	`$ FORT/DEB/NOOPT/OBJ=$COMPFILE.OBJ $COMPFILE`
ALPHA VMS	`$ FORT/SEPARATE/DEB/NOOPT/OBJ=$COMPFILE.OBJ $COMPFILE`
IBM VM/CMS	`'VFORT $COMPFILE (NOPRINT'`
IBM MVS	`CO $COMPFILE CPGM FVS ELOAD`
MSDOS	`F77 -C $compfile`
WINDOWS/NT	`f77 -nologo $compfile -nolink`

Table 4.2: *Default FORTRAN compiler directives.*

With option `-C` command `SET_FILE` can also be used to change the default compiler directive. Give the compiler directive between quotes. The string '`$compfile`' will be replaced by the name of the file to be compiled. If '`$compfile`' is followed by a file extension, like '`.bin`' or '`.o`', then it will be replaced by the name of the file to be compiled minus the compile file extension. See tables 4.2 and 4.3 for the default FORTRAN and C compiler directives.

You can change the default editor that will be invoked by `EDIT` with the command `HOST_EDITOR`. Giving `HOST_EDITOR` without argument will show you the currently set editor. At startup the default editors are as in table 4.4. On systems running the X window system you can define the editor via `HOST_EDITOR 'xterm -e vi'`. In that case you get the editor in a new window and you can still read and scroll the output of the CMZ session in your CMZ window.

The default shell used by the `SHELL` command can be changed with the command `HOST_SHELL`. Giving `HOST_SHELL` without argument will show you the currently set shell. At startup the default shell on Unix systems is the Bourne shell (`/bin/sh`).

4.13 Compiling and Archiving of Decks

One of the main features of CMZ is that you can activate from the CMZ shell, in a totally transparent way, the local FORTRAN or C or other compiler and the librarian.

```
CXTRACT [-H[L]][-U][-R] (-P | [-P][-V vers_num] -B | [-K] $USE | -F car_file |
        [-P][-N][-V vers_num] decklist)
CCOMPILE [-H[L]][-U][-R] (-P | [-P][-V vers_num] -B | [-K] $USE | -F car_file |
         [-P][-N][-V vers_num] decklist)
CLIB [-H[L]][-U][-R] (-P | [-P][-V vers_num] -B | [-K] $USE | -F car_file |
     [-P][-N][-V vers_num] decklist)
LIBRARY [-D][object_file]
CMAKE [-N] [-A] [-O] [-P] [-I history_filename] [-V [vers1]][-VD vers2]
```

	C Compiler Directive
Apollo	`/com/cc $compfile -b $compfile.bin -pic -dbs`
HP-UX	`cc -g -c $compfile`
Gould UTX	`cc -g -c $compfile`
Cray Unicos	`cc -c $compfile`
Ultrix	`cc -g -c $compfile`
SunOS	`cc -g -c $compfile`
SOLARIS	`/opt/SUNWspro/bin/cc -g -c $compfile`
IRIX	`cc -g -c $compfile`
CONVEX	`cc -fi -g -c $compfile`
IBM 6000	`cc -g -c $compfile`
Alliant	`gcc -c -g -w $compfile`
VAX VMS	`$ CC/DEB/NOOPT/OBJ=$COMPFILE.OBJ $COMPFILE`
ALPHA VMS	`$ CC/DECC/PREFIX=ALL/DEB/NOOPT/OBJ=$COMPFILE.OBJ $COMPFILE`
IBM VM/CMS	`‘CC $COMPFILE ALIAS($COMPFILE)‘`
IBM MVS	`CO $COMPFILE CPGM C ELOAD`
MSDOS	`gcc -c -O2 $compfile`
WINDOWS/NT	`claxp -c -nologo -DWIN32 -DALPHA=1 -D_ALPHA_ $compfile`

Table 4.3: *Default C compiler directives.*

	Editor
Apollo	DM System Editor
Unix	vi
VAX VMS	edit
IBM VM/CMS	xedit

Table 4.4: *Default editors.*

With the commands `CXTRACT`, `CCOMPILE` and `CLIB` it is possible to extract the compile-ready FORTRAN or C or other source code (depending on the language mode), or to extract the source code and invoke the local compiler, or to extract the source code, invoke the local compiler followed by the local librarian. All three commands resolve the ‘`+SEQ`’ and ‘`+IF`’ directives and write the code to the default extract file (see the previous section for the default filenames and how to change them). Next, the commands `CCOMPILE` and `CLIB` invoke the desired compiler with the compiler directive as specified by `SET` with the `-C` option (see previous section). The command `CLIB` continues by invoking the local librarian and archiving the compiled file into the default object library (see the previous section for the default object library name and how to change it).

NOTE

Before issuing either `CXTRACT`, `CCOMPILE` or `CLIB` you must first set the control options and load the sequence definitions (see the sections 4.7 and 4.8, respectively).

These three commands feature an identical set of options. The decks on which they operate are specified either with *decklist*, option `-B`, `-F` or `$USE`. If you use option `-B` all decks stored in the *decklist buffer* (see section 7.5.2) will be processed. If you specify option `$USE` all decks referenced in the USE bank (see section 5.3) will be processed. With option `-F` a *car_file* containing decks or normal code can be processed (see also command `AXTRACT`, section 4.13.3). If option `-R` is specified all decks are removed from the USE bank after processing (see section 5.5 for an example of the use of the `-R` option together with the `-F` and `$USE` options). If you set option `-V` only those decks belonging to version *vers_num* will be written to the compile file. By default the corrections stored in the USE bank will be taken into account. If you don’t want any correction cards to be used set option `-U`. If option `-N` is set the decks will be taken from the USE bank (the decks must be new decks not yet in the CMZ file). The options `-N` and `-U` and `$USE` and `-U` are mutually exclusive. In case you don't want the history records in the compile file you have to set option `-H`. If you set option `-HL` then only the last CMZ history records will be in the compile file. The options `-H[L]`, `-U`, `-R` and `-N` may be concatenated, they must, however, precede options `-B`, `-P`, `$USE` or the *decklist*.

Concerning the `-N` option

In case you are doing `CLIB *`[5] from the top-level of the CMZ file any new decks in patch '`CRA*`' (which has been created in memory by the command `USE`, see 5.3) will automatically be added to the default object library. However, in case you do `CLIB //top` the new decks in patch '`CRA*`' will not automatically be added to the default object library. The reason behind this is that in case you have a large number of CMZ files you might want to build your object library not all at once but in a number of steps, without having the new decks in patch '`CRA*`' added to the library at every step. Now to get at least once the new decks in the library you have to specify, for example, as last step `CLIB -N CRA*`.

NOTE

When in FORTRAN mode the commands `CXTRACT`, `CCOMPILE`, `CLIB` and `CMAKE` will not output any non-FORTRAN decks. Sometimes it can happen that you are convinced that a certain deck contains legal FORTRAN code, while nothing is written on the default FORTRAN file. What can be wrong? There are two possibilities:

1. The deck is not compile selected. Check the '`IF=`' parameters on the '`+DECK`' and '`+PATCH`' directives against the currently active control options.
2. The '`SUBROUTINE`' or '`FUNCTION`' statement does not start in column 7, but in the columns 1–6 (illegal syntax).

So, before panicking check your code carefully.

In X language mode the commands `CXTRACT`, `CCOMPILE`, `CLIB` and `CMAKE` will output only decks that have the option '`T=X`' on the '`+DECK`' directive (or all decks that are in a patch which has option '`T=X`'on the '`+PATCH`' directive).

Command `LIBRARY` can be used to archive the *object_file* into the default object library. If *object_file* is not specified, `LIBRARY` will archive the (binary) default compile file.

Use command `LIBRARY -D` to remove an *object_module* from the default object library. This command is useful to remove the object modules of obsolete decks (i.e. decks that have been deleted from the CMZ file) from the object library.

`CMAKE` is a powerful command with which you can automatically recompile all decks that are out of date compared to the date and time written on the 'FILE.history' file (the history file is written by the command 'LMODS -O'). If this file does not exist and the option '-I' is not specified `CMAKE` recompile decks that are out of date compared to the default library. `CMAKE` compares the time of last change written in the history file (or of the date and time of the default library if the history file does not exist) with the time of last change of all the decks in the CWD and its sub-directories. Decks that are more recent than the time of the last change will be written to the default extract file, recompiled and replaced in the library. All decks that need to be recompiled will also be stored in the decklist buffer (see section 7.5.2).

[5] Or `CXTRACT *` or `CCOMPILE *`.

If option `-O` is set the decks are compared to their default object files instead of the default library (in this case the librarian will not be called). If option `-N` is set the decks that need to be recompiled will only be listed. This is a very useful option since the decklist buffer is always filled and subsequently can be used by, for example, the command `UNDEFINED -B` (see section 7.6.8) to check all decks, that need to be recompiled, for undefined variables. Other useful sequences are:

```
CMAKE -N   followed by CLIB -B
CMAKE -NO followed by CCOMPILE -B
```

If the option '`-V vers1`' is given all decks having a version number greater than '*vers1*' or no version set are recompiled. If 'vers1' is not specified, the version number written on the history file (see section 4.13.2) is taken. If the option '`-V vers1 -VD vers2`' is given all decks having changed between '*vers1*' and '*vers2*' are recompiled. The resulting code belongs to the version 'vers2'.

If option `-A` is given then all connected CMZ files will be searched for decks that need to be recompiled.

Whenever a sequence definition has been changed all decks that *use* the changed sequence must be recompiled. This will be done, automatically, by `CMAKE`. To do this `CMAKE` checks the date and time of a deck with the date and times of all sequences used by that deck. If one sequence definition is changed more recently than the deck, the deck will be recompiled. The date and time of a sequence defintion is stored in the keep deck. When a number of sequence definitions are stored in one deck, they will all get the same date and time. Due to this fact it may happen that, although you changed only one sequence, many more decks will be recompiled than you expected.

4.13.1 Include Files

When the include option is selected via the command SET and option -INC (see section 4.12), the sequences are written onto include files instead of being developed inside the extract file. For example:

```
SET   src/*.c           -F     C
SET   '#include "$SEQ"' -INC   C
```

The CMZ directive '`+SEQ,PROJ.`' will be replaced by the include statement '`#include "proj.h"`' and the file '`proj.h`' will be written in the directory '`src`', if it does not exist or if the sequence `PROJ` has been modified.

The directive '`T=NOINCLUDE`' on the `+SEQ` card inhibits the writing of the include files for the sequences mentionned on the '`+SEQ`' card and allows the expansion of these sequences in the extract file.

4.13.2 The History File

The History file '`FILE.history`' where `FILE` is the name of the connected cmz file contains the date and time of the last deck changed in the file `FILE`. To create this file use the command `LMODS -O` (see section 7.1.11).
The command `CMAKE` (see section 7.1.10) reads the History file and recompiles the decks that are out of date compared to the date and time written on the '`FILE.history`' file. If this file does not exist and the option '-I' is not specified then the decks that are out of date compared to the default library are recompiled.

Creation of the 'libproject.a' library

```
CMZ [1] file    project
CMZ [2] set     libproject.a  -L
CMZ [3] sel     PRO  MACH
CMZ [4] seq     //
CMZ [5] clib    *
CMZ [6] lmods   -O
CMZ [7] exit
```

Updating the 'libproject.a' library

```
CMZ [1] file    project
CMZ [2] set     libproject.a  -L
CMZ [3] sel     PRO  MACH
CMZ [4] use     proj1.cor
CMZ [5] use     proj2.cor
CMZ [6] seq     //
CMZ [7] cmake
CMZ [8] lmods   -O
CMZ [9] exit
```

4.13.3 Creating source Code Directly from a CAR File

`AXTRACT` *car_file* `[-C[C]][-H][-U] [`*decklist*`]`

The command `AXTRACT` can be used to extract the compile-ready source code directly from a CAR file, without creating a CMZ file. All '`+SEQ`' and '`+IF`' conditionals will be resolved and the code will be written to the default source file. Before doing a `AXTRACT` make sure you have set all necessary control options with the `SELECT` command (see section 4.7). Any sequence definitions will be loaded automatically while reading the *car_file*.

If you give the `-C` option `AXTRACT` will strip off any comments in the columns 73–80 from statements which are not comment lines. When you set option `-CC`, however, the comments in columns 73–80 are stripped off from all lines. By default the corrections stored in the USE bank will be taken into account. If you don't want any correction cards to be used set option `-U`. In case you don't want the history records in the FORTRAN file you have to set option `-H`. The options `-C`, `-CC`, `-U` and `-H` may be concatenated, they must, however, precede the *decklist*.

4.14 Making a Listing of a CMZ Library File

`CLIST` *list_file* `[`*nn*`] [-A][-S][-H][-I][-E][-V` *vers_num*`] [-B |` *decklist*`]`

Use `CLIST` to generate a line-by-line listing of a CMZ file or directory. The listing shows the line count for each line and the deck and patch name. The line count corresponds to the line numbers on the action directives generated by `COR` (see section 5.2) and can be used to check where the corrections will be applied. At the end of the listing a table of contents, a sorted index of all patches, decks, sequences and control options is given. By default the CWD directory will be listed. In case the CWD is the top-directory the whole file will be listed.

A specific deck or set of decks and patches can be listed by using the optional parameters *decklist* or option `-B`. If you use option `-B` all decks stored in the *decklist buffer* (see section 7.5.2) will be listed. If option `-V` is given then all decks belonging to version *vers_num* will be listed.

By default FORTRAN carriage control characters will be used to format the listing. If you set option `-A` the listing will be formatted with ASCII carriage control characters. In case you don't want the history records in the listing you have to set option `-H`. Set option `-I` in case you are only interested in an index. If option `-E` is given then the listing will be shown directly in the local browser or pager. The default number of lines per page is **56**. You can change the number of lines per page by giving a new number *nn*. If the option `-S` is given every deck starts a new page.

4.15 Copying Decks

`COPY [-C] (-B |` *source*`)` *target*
`CPTREE` *old_topdir new_topdir*
`MKDIR` *new_dir*

Decks can be copied from one directory to an other directory with the `COPY` command. The other directory may be in the same or in a different CMZ file. Use `COPY` to copy *source* deck(s) or the deck(s) in the decklist buffer (option `-B`) to *target*. The *source* may contain wildcards[6] (see section 7.5.11) or be in the form of a decklist. In either case the *target* must be a directory. If the current working directory is the *target* than it may be specified with a '`.`'. Since a `RENAME` command does not exist, `COPY` should be used for that purpose. First copy a deck to a new location then delete the original. If *target* is not the same name as *source* the deck name on the '`+DECK`' directive will automatically be changed to *target*. `COPY` will not clear the version field in the top-most history record of a deck and it will automatically lock decks belonging to a version. This default behaviour is useful when decks are copied from a master CMZ file to a private CMZ file for further development. When later comparing the copied deck with the original deck using `COR` (see section 5.2) one does not get a correction line for the history record due to the missing version number. If the option `-C` is given, the date and time of the copied decks are set to the current one.

With the command `CPTREE` it is possible to copy one *complete* CMZ file (*old_topdir*) to a new CMZ file (*new_topdir*). With *complete* we mean every cycle of every deck of every directory. This command is useful to compact a fragmented CMZ file. After a lot of edit and purge cycles a CMZ file will get fragmented, i.e. it will contain holes, some of which are too small to be filled with new decks [7]. To get rid of these holes use `CPTREE` to make a new compact CMZ file. Before using `CPTREE` the CMZ file *new_topdir* should have been created with `MAKE` (see section 4.3).

Use `MKDIR` to create a new directory. This new directory is empty except for a '`00_PATCH`' deck which contains the '`+PATCH`' directive. To add, for example, an '`IF=`' parameter to the '`+PATCH`' directive you have to edit the '`00_PATCH`' deck. To move a newly created directory to its desired place in the list of directories use the `MOVE` command (see section 7.2.6).

4.16 Deleting Decks

`DELETE [-V` *vers_num* `| -O | -I] (-B |` *decklist*`)`

Decks can be deleted with the `DELETE` command. The decks to be deleted can be specified either by a decklist (see section 4.1.2) or by option `-B`. If option `-B` is used then the decks are taken from the decklist buffer (see section 7.5.2).
If option `-V` *vers_num* is given then the decks belonging to version *vers_num* that are not associated with any other version will be deleted. For example, deck '`com;3`' belongs to version 2.01/03, deck '`com;4`' belongs to version 2.01/04 and deck '`pipo;3`' belongs also to version 2.01/03 (deck '`pipo;3`' is the highest cycle of the '`pipo`' decks). If we now do `DELETE -V 2.01/03 *` we will only delete deck '`com;3`'. Deck '`pipo;3`' will not be deleted since it belongs also to version 2.01/04 (since their is not a deck '`pipo`' with a cycle number higher than 4 belonging explicitly to version 2.01/04).
Option `-O` should be used to mark decks as obsolete. Of these decks only the history records will be saved, the body of the deck will be deleted. This option is useful to keep a correct version structure. Say, for example, we don't need anymore routine '`NOOT`' in deck '`noot;4`'. Deck '`noot;3`', however, belongs to version 2.01/04. If

[6]The '`00_PATCH`' deck cannot be copied using wildcards.

[7]When the CMZ file gets too fragmented a warning is issued.

we would just delete deck ‘noot;4’ deck ‘noot;3’ would become the highest cycle and would, therefore, also belong to all higher versions of our program. To prevent this from happening deck ‘noot;4’ should be marked as obsolete.

4.17 The $VERSION, $CMZFILE and $CWD Functions

The KUIP system functions described in section 8.1.8 can be used inside a macro. Executing the macro shown in Figure 4.4 gives the following output:

```
CMZ [12] exec test
 HELLO: Today 03/04/94 11.27.28
        You are running CMZ on HPUX/UNIX

 You are using the following CERNLIB files

 Current File is: KUIP
 Current Directory is: //KUIP/CMOTIF

 PAW  : 2.05/01     HBOOK: 4.22/00
 HPLOT: 5.18/11     HIGZ : 1.21/00
 KUIP : 2.05/00     SIGMA: 1.10/03

 You are using the new version of PAW

 Current Working Directory = //HBOOK
 Changing directory to //KUIP/CMOTIF
 Current Working Directory = //KUIP/CMOTIF
```

```
macro TEST
message  HELLO: Today $DATE $TIME
message '       You are running CMZ on $MACHINE/$OS'
message
message You are using the following CERNLIB files
message
cernlib = /cern/new/cmz
file [cernlib]/paw.cmz -r ;file [cernlib]/hbook -r
file [cernlib]/hplot -r
file [cernlib]/higz  -r ;file [cernlib]/sigma  -r
file [cernlib]/kuip  -r
cd cmotif
message Current File is: $CMZFILE()
message Current Directory is: $CWD()
message
curdir = $CWD()
message PAW  : $VERSION(PAW)   HBOOK: $VERSION(HBOOK)
message HPLOT: $VERSION(HPLOT) HIGZ : $VERSION(HIGZ)
message KUIP : $VERSION(KUIP)  SIGMA: $VERSION(SIGMA)
message
if $SUBSTRING($VERSION(PAW),2,4)=2.05 then
   message You are using the new version of PAW
   message
endif
cd //hbook   ;  cd
message Changing directory to [curdir]
cd [curdir]  ;  cd
release *
return
```

Figure 4.4: *Example of a macro with functions.*

Chapter 5

Using CMZ in the Code Maintenance Phase

The requirements for a code management system during the code maintenance phase are quite different from the requirements during the code development phase. During code development the emphasis is on speed, interactivity, fast access to the local editor, compiler and librarian. During the code maintenance phase the emphasis is on version management, generation and usage of correction sets and *batch* functionality. In this chapter we will describe the tools that CMZ offers to make your life in the code maintenance phase easier.

5.1 Keeping Track of Versions

`VERSION [+ | +0 | +00 | -S[A]` *vers_num*`][-E][-F` *filename*`] | [-L]`

Use command `VERSION` to stamp a *version number* on the highest cycle of all decks in a CMZ file. Once decks have been marked with a version number they can be handled by all commands featuring the `-V` option[1]. A version number consists of three parts: a version (x), a release (y) and a level (z) part. These three parts are combined in the following, fixed, way to form a version number: `[x]x.yy/zz`. For example:

```
 1.01/02
13.06/11
```

The version number will be placed in the top most history record of a deck, e.g.:

```
*CMZ :  1.01/02 01/06/88  18.02.57  by  Louis Daniels
        ^^^^^^^
```

To set an initial version number on a CMZ file use option `-S` followed by a *vers_num*. After changing some decks (i.e. after creating new cycles) you can use option `+` to create a new level by increasing the level part of the version number by one. With option `+0` the release part can be increased by one and the level part reset to `00` and with option `+00` the version part can be increased by one and the release and level parts reset to `00`. If option `-L` is set a list of all versions stored in the CMZ file will be given. Without any option the current highest version number is given.

[1]Currently `CXTRACT`, `CCOMPILE`, `CLIB`, `CAR`, `CTOT`, `EDIT`, `DELETE`, `COR`, `DIFF`, `SEQUENCES` and `LMODS`.

In case the options `-S` *vers_num*, `+`, `+0` or `+00` are given, `VERSION` does a global `KEEP` (see section 7.5.8) on the whole CMZ file to lock the highest cycle of all decks. The new version number is written in the decks that do not yet have a version number. A new deck with the name '`V[x]x_yy`', corresponding to the new version number is created in directory '`$VERSION`' (if directory '`$VERSION`' does not yet exist CMZ will create one). This new deck is opened in the editor and you are encouraged to add behind the appropriate '`*::> VERSION`' line comments concerning the new version[2]. If option `-E` is set the deck `V[x]x_yy` is saved without appearing in the window editor. If option `-F` is set the text contained in '`FILENAME`' is added in the deck `V[x]x_yy`.

NOTE

The current version number is extracted from the '`TITLE`' deck and the new version number is written back into it. So never change the version number in the '`TITLE`' deck manually since this could disturb the version management.

5.2 Generating Correction Sets

`COR [-O][-H]` *original(s)* `[`*descendant*`] [-F` *cor_file*`]`
or
`COR [-H] -V` *vers_num1* `(`*descendant* `| -VD` *vers_num2*`) [-F` *cor_file*`]`

Once programs reach a certain size and become fairly stable it is no longer efficient to distribute after every change the whole CAR or, when binary file transfer is possible, the CMZ file. In that case it is much more efficient to distribute only a correction (*delta*) set. A CMZ correction set consists of a number of '`+ADB`', `+ADD`', '`+REP`' and '`+DEL`' directives as described in the next section.

Use command `COR` to generate a correction set. `COR` creates a correction set by comparing an *original* with a *descendant* deck, patch or file (note that *original* and *descendant* are either both a deck, a patch or a file). The correction set will be stored in a new deck in the directory '`$CORR`' (if directory '`$CORR`' does not yet exist CMZ will create one) with the name '`CORR`*n*', with $n = 1, 2, 3, \ldots$. When option `-F` has been specified the correction set will instead be written to an external file *cor_file*. Set option `-O` when `COR` has to compare only the decks *available* in the *descendant* file with the decks in the *original* file (see note 1 below). Use option `-V` when you want to compare version *vers_num1* of a deck, patch or file with its *descendant* (see note 2 below). If instead option `-VD` *vers_num2* is given then *vers_num2* is compared against *vers_num1*. Options `-H` and `-O` must precede all other options and arguments.

There are two suggested ways of making changes to a CMZ file so that it will be easy to generate a correction set later.

1. Make a new CMZ file and copy only the decks that you need to modify from the original to the new CMZ file (using `COPY`, see section 4.15 and 7.2.9), instead of making a complete copy of the original. Make sure that before you copy the decks you create identical sub-directories as in the original CMZ file (copy also the '`OO_PATCH`' decks from the original CMZ file in case they are not the default '`OO_PATCH`' decks, i.e. if they contain an '`IF=`' parameter). This scheme is not only space efficient but also allows for faster generation of the correction set. Generate the correction set with `COR -O //file_org //file_new`.
2. Another way to proceed (only when the original CMZ file has been locked by the `VERSION` command, see previous section) is to edit the decks directly in the original CMZ file and use option `-V` to generate

[2] A new deck will only be created after changing the release part of the version number (options `+0` and `+00`). When a new level is created (option `+`) the comments will be added to the deck of the corresponding release.

the correction set. Use `COR -V` *highest_vers_num* `//file`. In case you made a new version before you created the correction set you can generate the correction set with `COR -V` *one_highest_vers_num* `-VD` *highest_vers_num*.

5.2.1 The Directives '+ADB', '+ADD', '+REP', '+DEL'

These directives are generated by the command `COR` and used by the commands `USE` (see section 5.3 and 7.1.13) and `UPDATE` (see section 5.4 and 7.1.14) You never should create these directives yourself. They are only described here for completeness' sake.

The directive

```
+ADB, pname, dname, cnumber.
```

or

```
+ADD, pname, dname, cnumber.
```

cause the material following them to be *added before* or *added after* the specified line '*cnumber*' in the deck '*dname* in the patch '*pname*' (where the '`+DECK`' directive is always line 0).

The material following any of these directives is terminated by the first CMZ directive other than '`+SEQ`'. Thus action material can *quote* sequences. If the material contains other CMZ directives, like '`+DECK`', the '`+`' is replaced by a '`&`' to prevent the termination of the material. The '`&`' will be changed into a '`+`' during the interpretation of the action material. For example

```
+ADD, pname, dname, *.
&DECK, dname.
<rest of deck material>
```

will add a new deck '*dname*' to patch '*pname*'.

The directive

```
+REP, pname, dname, c1-c2.
```

causes the lines '*c1*' to '*c2*' inclusive to be deleted and replaced by the material following the directive. For single line replacement the parameter is written '*cnumber*'.

The directive

```
    +DEL, pname, dname, c1-c2 [,c3-c4, ...].
```

causes deletion of lines '*c1*' to '*c2*', '*c3*' to '*c4*', ... (inclusive). An entire deck is deleted by the directive

```
    +DEL, pname, dname, *.
```

and an entire patch by the directive

```
    +DEL, pname, ****, 0.
```

As stated before, you don't have to generate these directives yourself, CMZ will automatically generate them for you.

5.3 Using Correction Sets

```
USE [.] | [[-R][-E] deck_list] | [[-O][-I] cor_list] |
    [-P corr_dir] | [-F filename]
```

Before being able to do anything with a correction set you have to load it into CMZ's memory (into the so called USE bank). Do this with the `USE` command. Once the correction set(s) have been loaded into memory they will be automatically used by the commands `CXTRACT`, `CCOMPILE`, `CLIB`, `CTOT`, `ARC`, `CMAKE`, `SEQUENCES` and `UNDEFINED`.

To load corrections into memory give `USE` followed by one or more correction files. To be able to load a correction file generated ignoring CMZ history records you have to set the `-O` option. In this case CMZ will take the '`*-- Author :`' card as card 0, instead of the '`+DECK`' card when the corrections will be applied. Correction sets that are stored in a CMZ file can be loaded by specifying option `-P` followed by the directory name that contains the correction sets (normally this will be the '`$CORR`' directory). In this latter case option `-O` may not be specified. To list the patches and decks for which corrections are stored in the USE bank give `USE` without any arguments. A deck or a list of decks may be removed from the USE bank by specifying the `-R` option. Use argument `.` to wipe all corrections from memory.

When more than one correction set has been loaded into memory they will be automatically merged. When there are overlapping corrections (a clash) you will be warned. In that case check very carefully the result of the merging and if it is not correct you have to fix and re-load the correction sets before proceeding. Use option `-I` to suppress any warning messages concerning overlapping corrections, the first correction will be used. To view the result of the merging, in the local pager or browser, specify option `-E`, possibly followed by *deck_list*. The result of the merging can be written to a single new correction file by giving the option `-F` followed by a filename. Correction sets loaded with the `-O` option cannot be merged with other correction sets, i.e. it is forbidden to first load a correction set with the `-O` option and next load another set without the `-O` option, or vice versa.

5.4 Applying the Correction Sets

```
UPDATE [-E] ($USE | [-REF //reffil] decklist)
```

Once the correction set(s) have been loaded into memory you can update your CMZ file to reflect the original CMZ file. Use command UPDATE to update the decks specified by *decklist* or by $USE. When $USE has been specified then all decks referenced in the USE bank will be updated. UPDATE will create for every deck that has been changed a new cycle containing the updated version. It will also add the letter 'U' to '*CMZ :' in the top most history record to tell you that the deck has been updated using UPDATE, e.g.:

```
*CMZU:          15/03/89  13.09.16  by  Louis Daniels
```

If you specify option -E before the *decklist* the result of the update will be shown in the editor. If you like the result you can save the edit file and the deck will be updated. If, however, you don't like the result of UPDATE then just don't save the edit file and no updating will be done.

5.5 Using the -R, -F and $USE Options of CXTRACT, CCOMPILE and CLIB

In the maintenance phase it typically happens that you have to change or fix routines from an existing program. As an experienced CMZ user you create your own CMZ file in which you store the new or modified routines while leaving the master CMZ files untouched (see section 5.2). Once you've finished fixing the routines you create a correction set (see command COR) which can be shipped to other users. When other people on the project follow the same method you get after a while a number of correction sets. Now, to get a version of the program including your changes and the correction sets of other people there are two methods that can be followed:

1. You always apply the correction sets to your master CMZ files by using the UPDATE command. In this way your master files always reflect the most recent situation. To test your changed routines you only have to do

   ```
   CCOMPILE //new
   ```

   where '//new' is the CMZ file containing your changed routines. To make an executable of the new program you link the resulting object file from CCOMPILE with the object libraries.

2. You do not update your master CMZ files, but use the correction sets directly. Now you have to do the following to test your changed routines

   ```
   USE cor1.cra cor2.cra cor3.cra
   CCOMPILE -U -R //new $USE
   ```

   With the USE command you store the corrections in memory (in the USE bank). Next you compile your changed routines and all the routines referenced in the USE bank. The option -U tells CCOMPILE not to apply the corrections in the USE bank on the routines found in file '//new'. This ensures that you get your modified routines without any accidental changes introduced via the USE bank. Option -R tells CCOMPILE to remove all routines found in file '//new' from the USE bank before recompiling all routines referenced in the USE bank (option $USE). This prevents multiple inclusion of the same routine on the compile file.

An user not involved in major development of the program code may not want to create an own CMZ file but just make a CAR file[3] containing his modified routines. This mode of working is not advised (you lose CMZ's history mechanism and utility functions) but it can be useful to get people compiling their own version of a program with minimal knowledge of CMZ (like in large collaborations where the average physicist only needs

[3]The CAR file should, however, reflect the patch/deck structure of the master CMZ files.

to change a few routines of the general simulation or analysis programs). Also, in this case, there are two different ways of working depending on if the master CMZ files are updated or not:

1. In case the master CMZ files have been updated you just do

   ```
   CCOMPILE -F new.car
   ```

   where '`new.car`' is the CAR file containing your changed routines. To make an executable of the new program you link the resulting object file from `CCOMPILE` with the object libraries.

2. In case the master CMZ files have not updated you have to do

   ```
   USE cor1.cra cor2.cra cor3.cra
   CCOMPILE -R -F new.car $USE
   ```

   With the `USE` command you store the corrections in memory (in the USE bank). Next you compile your changed routines and all the routines referenced in the USE bank. The option `-F` tells `CCOMPILE` to read the routines from file '`new.car`'. Option `-R` tells `CCOMPILE` to remove all routines found in file '`new.car`' from the USE bank before recompiling all routines referenced in the USE bank (option `$USE`). This prevents multiple inclusion of the same routine on the compile file.

5.6 Running CMZ in Batch Mode

To run CMZ in batch mode you have to create a CMZ command file. A CMZ command file contains a list of CMZ commands, just like in a '`cmzlogon`' file (see section 2.3). Remember that CMZ does not read any '`cmzlogon`' file when running in batch mode, so you have to define your aliases (see 8.2.1) and author name (see command `AUTHOR`, section 4.2) in the batchjob command file. Commands that invoke the local browser, pager or editor are not allowed in batch mode[4].When running CMZ in batch mode with for example the argument '`-b batchfile`', the trace is written on the '`batchfile.log`' file in the working directory. For invoking CMZ in batch mode see section 2.1.

5.7 Automatic Installation of CMZ Files

When invoking CMZ with the argument '`-install`'[5] the macro '`/$KUMACS/INSTALL`', contained in the specified CAR or CMZ file, will be executed. This makes it possible to automatically create an executable program or object library without having to go into CMZ and type CMZ commands. See Figure 5.1 for an example install macro. This macro performs the same task as the one shown in Figure 4.2. The linking of the programs is done in macro '`BIND`' which must also reside in the patch '`$KUMACS`' (see Figure 5.2). To execute the install macro and create the programs '`fixit`' and '`stripit`', simply type:

```
cmz -install fixit
```

where '`fixit`' is either the file '`fixit.cmz`' or '`fixit.car`'. When '`fixit.cmz`' does not exist CMZ will create it from the CAR file, so there is no need to create the CMZ file by hand. The macro '`INSTALL`' can also have options. These options can be specified after the filename and will be passed directly to the install macro

[4]Not allowed in batch mode are: `EDIT`, `SEQUENCES -E`, `CLIST -E`, `UPDATE -E`, `INDENT`, `LABEL`, `COR` without `-F` option, `FIND -E` or `-R` and `VERSION`.

[5]Or '`/INSTALL`' or '`(INSTALL`' depending on which machine you are running (see section 2.1).

(see section 2.1). In these install macros we can make good use of the KUIP system functions '$MACHINE' and '$OS'(see section 8.1.8 for more information about these system functions).

5.8 Automatic set-up

When invoking CMZ with the argument '-x'[6] the macro '/$KUMACS/FLOGON', contained in the specified CAR or CMZ file, will be executed. To execute the set-up macro simply type:

```
      cmz -x fixit
    or
      cmz -x fixit setup
```

where 'fixit' is either the file 'fixit.cmz' or 'fixit.car'. When 'fixit.cmz' does not exist CMZ will create it from the CAR file, so there is no need to create the CMZ file by hand. The default macro 'FLOGON' (or the macro 'setup') can also have options. These options can be specified after the filename and will be passed directly to the set-up macro (see section 2.1).

5.9 Importing and Exporting CMZ Library Files

In addition to the CAR and ARC procedures, described in section 4.5.1, CMZ files can also be transported via networks or be NFS mounted in a heterogeneous environment. CMZ libraries are standard ZEBRA/RZ direct-access files with a record length of 512 bytes.

For example, a user on CERNVM can transfer a CMZ file from the CERN VAX Cluster to CERNVM in the following way:

```
      FTP VXCERN
      FTP > BIN F 512
      FTP > GET DISK$DELPHI:[VISITOR]DELANA.CMZ DELANA.CMZ
      FTP > QUIT
```

[6] Or '/X' or '(X' depending on which machine you are running (see section 2.1).

```
+DECK,INSTALL,T=TEXT.
*CMZ :  1.44/19 14/10/93  10.25.57  by  M. Baker
*-- Author :    F. Smith   24/04/91
*****************************************************************
*                                                               *
                  macro  install machine=x
*                                                               *
*****************************************************************

if ([machine]=x) then
  NAME = $MACHINE
else
  NAME = $upper([machine])
endif

case [NAME] in

    (APOLLO) set fixit.ftn -f
             sel . $MACHINE fixit
             ccompile *

             set stripit.ftn -f
             sel . $MACHINE stripit
             ccompile *

    (CRAY)
             set 'cft77 -o off -b $compfile.o $compfile' -c
             set fixit.f -f
             sel . $MACHINE fixit
             ccompile *

             set stripit.f -f
             sel . $MACHINE stripit
             ccompile *


    (?,HELP,X,*)
       mess To install DEMO on APOLLO: cmz -install demo APOLLO
       mess To install DEMO on CRAY  : cmz -install demo CRAY
       exitm

endcase

exec bind name==[NAME]

mess ' '
mess 'Installation of fixit and stripit finished'

return
```

Figure 5.1: *Example of an install macro.*

```
+DECK,BIND,T=TEXT.
*CMZ :  1.44/19 14/10/93  10.25.57  by  M. Baker
*-- Author :    F. Smith   24/04/91
******************************************************************
*                                                                *
                  macro  bind  name=x
*                                                                *
******************************************************************

case [name] in

    (APOLLO) shell /com/bind -b fixit fixit.bin
             shell /bin/rm fixit.bin

             shell /com/bind -b stripit stripit.bin
             shell /bin/rm stripit.bin

    (CRAY)   shell segldr -o fixit fixit.o
             shell /bin/rm fixit.o

             shell segldr -o stripit stripit.o
             shell /bin/rm stripit.o

endcase

return
```

Figure 5.2: *The* ‘`BIND`’ *macro as called by* ‘`INSTALL`’.

Chapter 6

Macros

The macro facilities provided by CMZ are part of the Kuip processor.

6.1 Macro definition

A macro is a set of command lines stored in a deck or in an external file, which can be created and modified with any text editor. The command `EXEC` (see section 7.5.5) envokes the macro and allows for two ways of specifying the macro name:

```
EXEC  mname [parameter-list]
EXEC  mname#macro [parameter-list]
```

The first form executes the first macro contained in '*mname*' while the second form selects the macro named '*macro*'. Macros may be stored in a deck or in a normal text file.

If '*mname*' is a full CMZ pathname the macro in that deck will be executed. IF '*mname*' is not a full pathname it will be searched for in the deck '*mname*' in the CWD. If it cannot be found in the CWD it will be searched for in the CMZ directory $KUMACS. If '*mname*' cannot be found in the CMZ file, it will be searched in the file '*mname*.kumac'.

In addition to all available CMZ commands the special "macro statements" in table 6.1 are valid only inside macros (except for `EXEC`) which is valid both inside and outside).

Note that the statement keywords are fixed. Aliasing such as "`ALIAS/CREATE jump GOTO`" is not allowed.

A `.kumac` file or a deck can contain several macros. An individual macro has the form

```
MACRO macro-name [ parameter-list ]
    statements
RETURN [ expression ]
```

Macro Statements	
STATEMENT	DESCRIPTION
`MACRO mname [ var1=val1 ... ]`	define macro `mname`
`RETURN [ value ]`	end of macro definition
`ENDKUMAC`	end of macro file
`EXEC mname [ val1 ... ]`	execute macro `mname`
`EXITM [ value ]`	return to calling macro
`STOPM`	return to command line prompt
`name = expression`	assign variable value
`READ var [ prompt ]`	prompt for variable value
`SHIFT`	shift numbered macro variables
`GOTO label`	continue execution at `label`
`label:`	`GOTO` target label (must terminate with a colon)
`IF expr GOTO label`	continue at `label` if `expr` is true
`IF-THEN, ELSEIF, ELSE, ENDIF`	conditional block statement
`CASE, ENDCASE`	Macro flow control
`WHILE-DO, ENDWHILE`	Macro flow control
`REPEAT, UNTIL`	Macro flow control
`DO, ENDDO`	Macro flow control
`FOR, ENDFOR`	Macro flow control
`BREAKL`	Macro flow control
`NEXTL`	Macro flow control
`ON ERROR CONTINUE`	ignore error conditions
`ON ERROR GOTO label`	continue at `label` on error condition
`ON ERROR EXITM value`	return to calling macro on error condition
`ON ERROR STOPM value`	return to command input on error condition
`OFF ERROR`	deactivate the `ON ERROR GOTO` handling
`ON ERROR`	reactivate the previous `ON ERROR GOTO` setting

Table 6.1: *List of statements possible inside CMZ macros*

Each statement is either a command line or one of the macro constructs described below. For the first macro in the file the `MACRO` header can be omitted. For the last macro in the file the `RETURN` trailer may be omitted. Therefore a `.kumac` file containing only commands (like the `LAST.KUMAC`) already constitutes a valid macro.

NOTE

When the macros are stored as decks, the first line in a macro must be the line '`MACRO macro_name`' and the last line of the macro must be '`RETURN`'. In that case, they also can contain `+IF` directives which will be resolved before the macro will be executed.

Input lines starting with an asterisk ("`*`") are comments. The vertical bar ("`|`") acts as inline comment character unless it appears inside a quoted string. An underscore "`_`" at the end of a line concatenates it to the next line.

Envoking a macro triggers the compilation of the whole `.kumac` file—not just the single macro called for. The

```
        ENDKUMAC
```

statement fakes an end-of-file condition during the compilation. This allows to keep unfinished material, which would cause compilation errors, simply by moving it after the `ENDKUMAC` statement rather than having to comment the offending lines.

6.2 Macro execution

Inside a macro the `EXEC` statement can call other macros. A macro may call itself recursively. Although the `EXEC` *statement* has the same form

```
      EXEC  macro [ argument-list ]
      EXEC  file#macro [ argument-list ]
```

as the `EXEC` *command* there are slight differences. The command "`EXEC name`" executes the first macro in `name.kumac` while the `EXEC` statement will try first whether a macro `name` is defined within the current `.kumac` file.

Macro execution terminates when one of the statements

```
      EXITM [ expression ]
      or
      RETURN [ expression ]
      or
      STOPM
```

is encountered. The `EXITM` and `RETURN` statements return to the calling macro. They allow to pass a return value which is stored into the special variable `[@]` of the calling macro. If no value is given it defaults to “`0`”.

The `STOPM` statement unwinds nested macro calls and returns to the command line prompt immediately.

Note that the `RETURN` statement also flags the end of the macro definition, i.e. the construct

```
IF ... THEN
   RETURN         | error!
ENDIF
```

is illegal.

6.3 Macro variables

Macro variables do not have to be declared. They become defined by an assignment statement:

```
name = expression
```

The right-hand side of the assignment can be an arithmetic expression or a string expression. The expression is evaluated and the result is stored as a string (even for arithmetic expressions).

The variable value can be used in other expressions or in command lines by enclosing the name in square brackets:

```
[name]
```

For example:

```
greet = Hello
msg = [greet]//' World'
MESS [msg]
```

If the name enclosed in brackets is not a macro variable then no substitution takes place.

Variable values can also be queried from the user during macro execution. The statement

```
READ  name [ prompt ]
```

prompts for the variable value. If the prompt string is omitted it is constructed from the macro and variable names. The variable value prior to the execution of the `READ` statement will be left unchanged if the prompt is answered simply be hitting the RETURN-key.

Macro using the READ statement

```
MACRO m
READ foo
bar = abc
READ bar
MESS [foo] [bar]
READ msg 'Enter message:'
MESS You said [msg].
```

Output when executing

```
CMZ [10] EXEC m
 Macro m: foo ? (<CR>=[foo]) 123
 Macro m: bar ? (<CR>=abc)
 123 abc
 Enter message: Hello
 You said Hello.
```

6.4 Macro arguments

The `EXEC` command can pass arguments to a macro. The arguments are assigned to the numbered variables `[1]`, `[2]`, etc. For example, with the macro definition

```
MACRO m
MESS p1=[1] p2=[2]
```

we get the result

```
CMZ >  EXEC m foo bar
  p1=foo p2=bar
```

Unlike named variables undefined numbered variables are always replaced by the blank string `' '`, i.e.

```
CMZ > EXEC m foo
  p1=foo p2=' '
```

The `MACRO` statement can define default values for missing arguments. With the macro definition

```
MACRO m 1=abc 2=def
MESS p1=[1] p2=[2]
```

we get the result

```
CMZ > EXEC m foo
  p1=foo p2=def
```

The macro parameters can also be named, for example:

```
MACRO m arg1=abc arg2=def
MESS p1=[arg1] p2=[arg2]
```

Even if the parameters are named the corresponding numbered variables are created nevertheless. The named variables are a copy of their numbered counterparts rather that aliases, i.e. the above macro definition is equivalent to

```
MACRO m 1=abc 2=def
arg1 = [1]
arg2 = [2]
```

The named parameters can be redefined by a variable assignment which leaves the value of the numbered variable untouched. For example:

```
MACRO m arg=old
MESS [1] [arg]
arg = new
MESS [1] [arg]
```

yields

```
CMZ > EXEC m
  old old
  old new
```

The `EXEC` command allows to give values for named parameters in non-positional order. For example:

```
MACRO m arg1=abc arg2=def
MESS [arg1] [arg2]
```

can be used as

```
CMZ > EXEC m arg2=foo
  abc foo
```

Unnamed EXEC arguments following a named argument are assigned to numbered variables beyond the parameters listed in the MACRO definition. For example:

```
CMZ > EXEC m arg1=foo bar
 foo def
```

i.e. the second argument “bar” is not assigned to [arg2] or [2] but to [3].

The construct *name*=*value* may also be used in the EXEC command for names not defined in the macro’s parameter list. The variable *name* is implicitly defined inside the macro. For example:

```
                                         CMZ > EXEC m
MACRO m                                   [foo]
MESS [foo]               yields          CMZ > EXEC m foo=bar
                                          bar
```

A string containing a “=” must be quoted if it should be passed to the macro literally:

```
CMZ > EXEC m 'foo=bar'
 foo=bar
```

Since a undefined variable name can be thought of as having the value '[name]', the construct

```
IF [var]<>'[var]' THEN
```

allows to test whether such an external variable definition was provided.

6.5 Special variables

A numbered variable cannot be redefined, i.e. an assignment such as “1 = foo” is illegal. The only possibly manipulation of numbered variables is provided by the

```
SHIFT
```

statement which copies [2] into [1], [3] into [2], etc. and discards the value of the last defined numbered variable. For example, the construct

```
 WHILE [1] <> ' ' DO
   arg = [1]
   ...
   SHIFT
 NDDO
```

allows to traverse the list of macro arguments.

Example of Input Macros

```
MACRO EXITMAC
 MESSAGE At first, '[@]' = [@]
 EXEC EXIT2
 IF [@] = 0 THEN
   MESSAGE Macro EXIT2 successful
 ELSE
   MESSAGE Error in EXIT2 - code [@]
 ENDIF
RETURN
MACRO EXIT2
 READ NUM
 IF [NUM] > 20 THEN
   MESSAGE Number too large
   EXITM [NUM]-20
 ELSE
   CD //$FILENAME([NUM])
 ENDIF
RETURN
```

Output when executing

```
CMZ [21] EXEC EXITMAC
  At first, [@] = 0
  Macro EXIT2: NUM ? 25
  Number too large
  Error in macro EXIT2 - code 5
CMZ [22] EXEC EXITMAC
  At first, [@] = 0
  Macro EXIT2: NUM ? 16
  Macro EXIT2 successful
```

For each macro the following special variables are always defined:

`[0]`	fully qualified macro name, e.g. '`./fname.kumac#mname`'
`[#]`	number of macro arguments
`[*]`	concatenation of all macro arguments, separated by blanks
`[@]`	return value of the most recent `EXEC` call

Like for numbered variables these names cannot be used on the left-hand side of an assignment. The values or `[#]` and `[*]` are updated by the `SHIFT` statement.

6.6 Variable indirection

Macro variables can be referenced indirectly. If the variable `name` contains the name of another variable the construct

```
        [%name]
```

is substituted by its value. For example, this is another way to traverse the list of macro arguments:

```
DO i=1,[#]
  arg = [%i]
  ...
ENDDO
```

There is only one level of indirection, i.e. the name contained in `name` may not start with another '`%`'.

6.7 Macro flow control constructs

There are a variety of constructs available for controlling the flow of macro execution. Most for the constructs extend over several lines up to an end clause. The complete block counts as a single statement and inside each block may be nested other block statements.

The simplest form of flow control is provided by the

```
GOTO label
```

statement which continues execution at the statement following the target label:

```
label:
```

If the jump leads into the scope of a block statement, for example a `DO`-loop, the result is undefined. The target may be given a variable containing the actual label name, e.g.

```
name = label
  ...
GOTO [name]
  ...
label:
```

6.7.1 Conditional execution:

```
IF expression THEN
   statements
ELSEIF expression THEN
   statements
...
ELSEIF expression THEN
   statements
ELSE
   statements
ENDIF
```

The general `IF` construct executes the statements following the first `IF/ELSEIF` clause for with the boolean expression is true and then continues at the statement following the `ENDIF`.

```
MACRO CASE filename
   CASE [FILENAME] IN
   (*.ftn, *.for)  TYPE = FORTRAN
   (*.c)           TYPE = C
   (*.p)           TYPE = PASCAL
   (*)             TYPE = UNKNOWN
   ENDCASE
   MESSAGE [FILENAME] is a [TYPE] file.
RETURN
```

Figure 6.1: *Example for CASE labels with wildcards.*

The `ELSEIF` clause can be repeated any number of times or can be omitted altogether. If none of the expressions is true, the statements following the optional `ELSE` clause are executed.

```
IF expression GOTO label
```

This old-fashioned construct is equivalent to

```
IF expression THEN
   GOTO label
ENDIF
```

```
CASE expression IN
(label) statement [ statements ]
...
(label) statement [ statements ]
ENDCASE
```

The `CASE` switch evaluates the string expression and compares it one by one against the label lists until the first match is found. If a match is found the statements up to the next label are executed before skipping to the statement following the `ENDCASE`. None of the statements are executed if there is no match with any label.

Each label is a string constant and the comparison witht the selection expression is case-sensitive. A label followed by another label without intervening statement is considered as a syntax error. If the same statement sequence should be executed for distinct values a comma-separated list of values can be used.

The "*" character in a label item acts as wildcard matching any string of zero or more characters, i.e. "`(*)`" constitutes the default label (see example 6.1).

6.7.2 Loop constructs

The loop constructs allow the repeated execution of command sequences. For `DO`-loops and `FOR`-loops the number of iterations is fixed before entering the loop body. For `WHILE` and `REPEAT` the loop count depends on the boolean expression evaluated for each iteration.

```
DO loop = start_expr, finish_expr [, step_expr ]
   statements
ENDDO
```

The step size defaults to "1". The arithmetic expressions involved can be floating point values but care must be taken of rounding errors. A DO-loop is equivalent to the construct

```
count = ( finish-expr - start_expr ) / step_expr
loop = start_expr
step = step_expr
label:
IF [count] <= 0 THEN
   statements
   loop = [loop] + [step]
   count = [count] + 1
   GOTO label
ENDIF
```

where all variables except for `loop` are temporary.

Note that "`DO i=1,0`" results in zero iterations and that the expressions are evaluated only one. i.e. the loop

```
n = 10
DO i=1,[n]
   MESS [i] [n]
   n = [n] - 1
ENDDO
```

is iterated 10 times and leaves "`i = 11`" afterwards.

```
FOR name IN expr_1 [ expr_2 ... expr_n ]
   statements
ENDFOR
```

In a FOR-loop the number of iterations is determined by the number of items in the blank-separated expression list. The expression list must not be empty. One by one each expression evaluated and assigned to the variable `name` before the statements are executed. The equivalent construct is the loop-unrolling

```
name = expr_1
statements
name = expr_2
statements
...
name = expr_n
statements
```

The expressions can be of any type: arithmetic, string, or garbage expressions, and they do not need to be all of the same type. In general each expression is a single list item even if the result contains blanks. For example:

```
foobar = 'foo bar'
FOR item IN [foobar]
   MESS [item]
ENDFOR
```

results in a single iteration. The variable `[*]` is treated as a special case being equivalent to the expression list "`[1] [2] ... [n]`" which allows yet another construct to traverse the macro arguments:

```
FOR arg IN [*]
   ...
ENDFOR
```

```
WHILE expression DO
   statements
ENDWHILE
```

The `WHILE`-loop is iterated while the boolean expression evaluates to true. The loop body is not executed at all if the boolean expression is false already in the beginning. The equivalent construct is:

```
label:
IF expression THEN
   statements
   GOTO label
ENDIF
```

```
REPEAT
   statements
UNTIL expression
```

The body of a `REPEAT`-loop is executed at least once and iterated until the boolean expression evaluates to true. The equivalent construct is:

```
label:
   statements
IF .NOT. expression GOTO label
```

```
BREAKL [ levels ]
```

allows to terminate a loop prematurely. The `BREAKL` continues executing after the end clause of the innermost `DO`, `FOR`, `WHILE`, or `REPEAT` construct.

```
NEXTL [ levels ]
```

allows to terminate one loop iteration and to continue with the next one. The NEXTL statement continues executing just before the end clause of the enclosing DO, FOR, WHILE, or REPEAT block.

Both BREAKL and NEXTL allow to specify the number of nesting levels to skip as an integer constant.

Example of using BREAKL and NEXTL

```
WHILE 1=1 DO
   ...
   IF expr THEN
      BREAKL
   ENDIF
   ...
   DO i=1,[#]
      ...
      IF [%i]='-' THEN
         NEXTL
      ENDIF
      IF [%i]='--' THEN
         NEXTL 2
      ENDIF
      ...
   ENDDO
   ...
  ENDWHILE
```

Equivalent code using GOTOs

```
WHILE 1=1 DO
   ...
   IF expr GOTO break_while
    ...
   DO i=1,[#]
      ...
      IF [%i]='-' GOTO next_do
      IF [%i]='--' GOTO next_while
     ...
   next_do:
   ENDDO
   ...
next_while:
ENDWHILE
break_while:
```

6.8 Error handling

```
ON ERROR GOTO label
```

installs an error handler which tests the status code after each command and branches to the given label when a non-zero value is found. The error handler is local to each macro.

```
ON ERROR EXITM [ expression ]
```

and

```
ON ERROR STOPM
```

are short-hand notations for an ON ERROR GOTO statement with a EXITM or STOPM statement, respectively, at the target label.

```
ON ERROR CONTINUE
```

nullifies the error handling. Execution continues with the next command independent of the status code. This is the initial setting when entering a macro.

```
OFF ERROR
```

and

```
ON ERROR
```

allow to temporarily suspend and afterwards reinstate the previously installed error handling. Note that the `OFF`/`ON` settings do not nest, for example

```
ON  ERROR EXITM
OFF ERROR        | behave like ON ERROR CONTINUE
ON  ERROR STOPM
OFF ERROR
ON  ERROR        | restore ON ERROR STOPM
ON  ERROR        | unchanged, i.e. not ON ERROR EXITM !
```

Part II

Reference Manual

Chapter 7

CMZ COMMANDS

7.1 CMZ basic commands

7.1.1 AUTHOR [author]

`AUTHOR` C "User name" `D='?'`
Make yourself known to CMZ. This name is used by CMZ history mechanism to mark any changes made by AUTHOR.

```
Ex.  AUTHOR Louis Daniels    ; set the user name
     AUTHOR                  ; show the currently set user name
```

7.1.2 SELECT [olist]

`OLIST` C "List of options" `D='?'`
Select control options
(used to test IF= conditionals on CMZ directives). If no arguments are given then show a list of all active options. If the argument is a . then the option list will be cleared. If the control option starts with a - then that option will be removed from the option list.

```
Ex.  SELECT                ; show active options
     SELECT .              ; clear option list
     SELECT vax gks debug  ; set options
     SELECT -gks           ; remove option GKS from option list
```

7.1.3 SEQUENCES [dirnam]

`DIRNAM` C "Name of sequence patch" `D='?'`

Possible `DIRNAM` values are:

```
 .  | -E | -S | -F File | -I [File] | -P Patch | [-V vers_num1 [VREF vers_num2]][-U][-O]
```

Store all sequences definitions (+KEEP directives)

in directory DIRNAM into memory. Always set the control options with command SELECT before loading the sequence definitions. The control options are used to test IF= conditionals on the +KEEP directives. If no arguments are given then show all sequence definitions in memory. If the argument is a "." then clear all active sequences from memory. By default the USE bank is taken into account. If option -U is set then the corrections referenced by the USE bank are ignored. Use option -O to overwrite sequence definitions of sequences that are already in memory. By default they are ignored. If option -V vers_num1 is given then load the sequence definitions belonging to version vers_num1 into memory. If the option -VREF vers_num2 is given, all sequences which have changed between the reference version vers_num2 and the vers_num1 version are marked internally for a later use in the code extraction step via the command 'cmake -V v1 -VD v2'. Use option -E to see the full sequence definitions stored in memory. To view the sequence definitions with all +SEQ directives resolved use the -S option. Both last two options use the local pager or browser (see command READ). To write all sequence definitions stored in memory to an external file use option -F followed by a filename (all +SEQ directives will be resolved and the +KEEP will be changed to *KEEP). If the option -I is set the sequences and their descendant sequences are written on separate files, and +SEQ directives are replaced by the corresponding include lines. To write all sequence definitions stored in memory to a new or existing patch use option -P followed by a patchname (+SEQ directives will not be resolved). The sequences will be stored in the deck 00_CDES.

```
Ex.  SEQ                  ; show all stored sequences definitions
     SEQ  .               ; clear all sequences from memory
     SEQ  hcdes           ; store all sequences in patch HCDES into memory
     SEQ  //file1 //file2 ; store all sequences in both file FILE1 and
                            FILE2 into memory
     SEQ  //              ; store all sequences in all connected files
                            into memory
     SEQ -U //            ; idem, but corrections referenced in the USE
                            bankre ignored
     SEQ -O hcdes         ; overwrite the sequence definitions of the
                            sequences in memory with the ones in
                            patch HCDES
     SEQ -V 1.00/02 hcdes ; store all sequences in patch HCDES belonging
                            to version 1.00/02
     SEQ -V 1.00/02 -VREF 1.00/00 hcdes ;
                             store all sequences in patch HCDES belonging
                             to version 1.00/02. All sequences modified
                             between the reference version 1.00/00 and the
                             1.00/02 version are marked internally for a
                             later use when extracting code via the command
                             cmake -V 1.00/00 -VD 1.00/02 (see help cmake)
     SEQ -VREF 1.00/00 hcdes ; idem as above, but the latest version number
                               of the file is taken as input for option -V
     SEQ -E               ; view the full sequence definitions in
                            the local pager or browser
     SEQ -S               ; idem as above, but with all +SEQ
                            directives resolved
     SEQ -F file          ; write sequences to FILE
     SEQ -F *.h           ; split sequences in separate files
     SEQ -I hdir/*.h      ; as above but +SEQ replaced by an include line
                            instead of being developed inside the mother
                            sequence.
     SEQ -P patch         ; write sequences to PATCH
```

7.1.4 SET [fname ftype lang]

FNAME	C	“Name of the file”
FTYPE	C	“Type of file” D='␣'
LANG	C	“List of language mode”

Possible FTYPE values are:

```
-E set the editor filename
-F[E] set the extract (source) filename
-L set the object library filename
-X[A][C][D[1][2]][L[nnn]][P] set the execution filename
-S set the save filename
-LAN define the output file names
-COM define the comment character(s)
-INC[1] select the include option
-C define the compiler directives
-T[D] set the template filename
```

Change the default filenames

for the editor, extract (FORTRAN, C, text,etc.), library, execution, template or save files.Note that the Filecase may be changed via the KUIP command FILECASE.

```
SET                 ; without arguments shows the current defaults

SET cmedt.edt -E    ; set the editor filename to which command EDIT
                      writes

SET cmfor.f   -F    ; set the FORTRAN (or C or any other language)
                      extract filename to which the commands CXTRACT,
                      CCOMPILE, CLIB and CMAKE will write
SET new_*.f   -F    ; if FNAME contains the character "*", then the above
                      commands will automatically write as many files as
                      there are decks in the DECKLIST of these commands.
                      The "*" will be replaced by the deck names
SET $cmzdir_*.c -F C; Set the compile filename for the C language,
                      without changing the current language. $CMZDIR will
                      be replaced by the current cmz directory.
SET SRC/*.C -FE C++ ; idem as above. Only on UNIX systems.
                      When using the command CXTRACT, CCOMPILE, CLIB or
                      CMAKE, the extracted code will contain special CMZ
                      comments. These special comments allow CMZ to later
                      re-import (see SYNCHRONIZE) the decks that were
                      changed outside of the CMZ environment. Also when
                      using one of the above mentioned commands the most
                      recently changed decks, either from the CMZ file or
                      from the extract directory, will be used. This
                      feature allows one to edit the extract files
                      directly, while still being able to use the CMZ
                      build commands.

SET cmlib.a   -L    ; set the object library filename which will be
                      created by CLIB or CMAKE
SET $cmzfile.a -L   ; $CMZFILE will be replaced by the file name
                      corresponding to the current directory. For
                      instance, if the current file is "project.cmz",
                      then the library file will be set to "project.lib"

SET go.com -X       ; set the execution filename which will contain the
                      compile and/or archive commands created by the
                      commands CCOMPILE, CLIB and CMAKE
SET go.com -XA      ; idem as above and in addition the execution file
                      GO.COM will be automatically executed
SET go.com -XAL100  ; idem as above and in addition the Librarian (L)
                      will be invoked after all calls to the compiler and
                      by packets of 100 object files (on UNIX systems).
SET go.com -XACDL   ; idem as above and in addition a Comment (C)
                      directive is added before each compiler directive,
                      the Librarian (L) will be invoked after all calls
                      to the compiler and the object file(s) will be
                      Deleted (D) once they have been replaced in the
                      library
SET go.com -XACD1L  ; idem as above but the compiler source file(s)
                      will be Deleted (D1).
SET go.com -XACD2L  ; idem as above but the compiler source file(s) and
                      the object file(s) will be Deleted (D2).
SET go.com -XAP     ; option P prevents the setting of the PATH in the
                      execution file

SET cmzsave.dat -S  ; set the file on which the current environment is
                      saved when exiting from CMZ
SET . -S            ; unset the save file, no environment will be saved
```

The environment of a previous session can be restored at the next entry to CMZ with:

```
cmz -r [cmzsave.dat]       on Unix system
CMZ/RESTORE[=CMZSAVE.DAT]  on VAX/VMS
CMZ (RESTORE[=CMZSAVE.DAT] on VM/CMS
```

The language selection is based on the 'T=' directive on the +DECK and/or +PATCH line. Use option -LAN to specify the current language. For example:

```
SET F77 -LAN        ; set F77 the current language (the default)
SET C   -LAN        ; set C the current language, all new edited decks
                      will have the directive 'T=CC' on the +DECK line.
SET TEXT -LAN       ; set TEXT the current language, all new edited decks
                      will have the directive 'T=TEXT' on the +DECK line.
SET EIFFEL -LAN     ; set EIFFEL the current language, all new edited
                      decks will have the directive 'T=EIFFEL' on the
                      +DECK line.
SET LateX -LAN      ; set LATEX the current language, all new edited decks
```

```
                         will have the directive 'T=LATEX' on the +DECK line.
    SET ? -LAN           ; list all defined languages
```

When a new deck is created (via the EDIT command), the +DECK line will also contain the directive 'T=Current_language'. Note, however, that if the current language is Fortran, the 'T=' directive is omitted and if C is the current language, the directive is 'T=CC'. Option -LAN also allows to select a list of languages to be considered when executing the command CXTRACT, CCOMPILE, CLIB or CMAKE. The language selection is based on the 'T=' directive on the +DECK line.

```
    SET * -LAN           ; All languages are selected for compilation
    SET . -LAN           ; No languages selected for compilation
    SET F C C++ -LAN     ; Languages Fortran77, C and C++ are selected for
                           compilation
    SET F -LAN           ; Only Fortran77 is selected for compilation
    SET * -C++ -LAN      ; All languages except C++ are selected for
                           compilation
    SET -EIFFEL -LAN     ; Remove EIFFEL from the list of selected languages
                           for compilation.
```

Use option -COM to define the comment character(s). Comment character cannot be redefined for C and F77 languages.

```
    SET % -COM Latex     ; set % as comment character for Latex language
    SET /* */ -COM C++   ; set /* and */ as comment characters for C++ language
    SET . & -COM MYLNG   ; "." means no special character at the beginning of a
                           comment line but "&" at the end of the line for the
                           MYLNG language.
    SET . -com MYLNG     ; remove the definition of the comment character for the
                           MYLNG language.
```

Use option -INC to select the include option for the currently selected language. When this option is active, the sequences are written onto include files instead of being developed inside the extract file. Example:

```
    SET '\include {directory_path/$SEQ.exth}' -INC LaTeX
    SET '#include "directory_path/$SEQ.exth"' -INC C
```

The string $SEQ is replaced by the sequence name when writing the deck onto the extract file. If the extension file '.exth' is omitted, the default extension '.h' is taken. If the directory_path is not set, the include files are written in the same directory as the one of the extract files. For example:

```
    SET src/*.c -F C
    SET '#include "$SEQ.h"' -INC C
    The CMZ directive '+SEQ,PROJ' will be replaced by the include statement
    '#include "proj.h"' and the file 'proj.h' will be written in the
    directory src, if it does not exist or if the sequence PROJ has been
    modified.
```

The command 'SET . -INC' disables the include directive and sequences will be expanded in the extract files. The directive "T=NOINCLUDE" on the +SEQ card inhibits the writing of the include files for the sequences mentionned on the +SEQ card and allows the expansion of these sequences in the extract file. If the option -INC is set the sequence and its nested sequences are written onto include files. If the option -INC1 is set (the default), the nested sequences are developped in the calling sequence.

If option -C is given, FNAME is a string between quotes containing the compiler directive for the currently selected language. For example:

```
SET 'ftn $compfile -b $compfile.bin -save -indexl -zero -cpu any' -C
SET 'VFORT $COMPFILE (NOPRINT' -C
SET 'LateX $COMPFILE' -C
```

where $COMPFILE will be replaced at execution time by the name of the file to be compiled. The compiler directive must at least contain one $COMPFILE string. To change the compiler directive without switching the language, the name of the language must follow the option -C:

```
SET C -LAN
SET 'f77 -c -g +ppu $compfile' -C F77
```

Replaces the compiler directive for F77 language. The current language is still C. The F77 language mode is added to the list of the selected languages for compilation.

```
SET . -C EIFFEL  ; Remove compiler directives for EIFFEL language mode.
```

Example of settings for a user defined language (UNIX systems):

```
SET CDF -LAN
SET *.cdf -F CDF
SET 'kuipc $compfile.cdf $compfile.c; cc -c $compfile.c' -C CDF
```

A template file contains a deck header template. The deck header template will be inserted by EDIT every time a new deck is created. The template will appear just below the default history records. The strings $DECK, $AUTHOR, $DATE and $TIME are reserved and will be replaced by the deck name, the author name (as set by the AUTHOR command), the date and the time, respectively. The string $# acts as a TAB character. If option -TD is used the FNAME is considered to be a deck name. The default file extension of the template file is .CMT and need not be specified. To un-set a template filename give as FNAME a ".".

```
SET temp.cmt -T     ; to set the template filename.
```

7.1.5 EDIT dklist

`DKLIST` C “[-V vers_num] (-B | List of the decks to be edited)” `D='␣'`

Edit one or more decks using the local editor. The deck(s) to be edited can be specified by a DKLIST (see DECKLIST) or by option -B. If option -B is specified then edit all decks in the decklist buffer (see BUFFER). If option -V vers_num is given then edit the decks belonging to version vers_num. The specified decks will be written to an edit file. The name of the edit file can be changed by the command SET ... -E. When editing a new deck, you can specify all the directives you want on the +DECK card.

```
     Ex.  EDIT deck1        ; only deck1
          EDIT deck2 deck3  ; only deck2 and deck3
          EDIT deck2.deck3  ; all decks from deck2 to deck3
          EDIT .deck3       ; all decks up to deck3
          EDIT deck2.       ; all decks starting at deck2
          EDIT deck4  //file2/patch/deck6
          EDIT *            ; all decks in patch
          EDIT -B           ; all decks in the decklist buffer
          EDIT -V 1.00/01 * ; all decks belonging to version 1.00/01
          EDIT $keep1       ; edit sequence keep1
          EDIT 'NEWDK, T=C++, IF=PROJ1. Comments' ;
                              create the new deck NEWDK. The
                              '+DECK,NEWDK, T=C++, IF=PROJ1. Comments'
                              line is generated.
```

Notice that only decks that have been changed in the edit file will be stored as a new cycle if a new copy of the edit file is written on disk. A keep sequence has no +DECK line, it starts with a +KEEP line. If you want to split your sequences in keep decks, remove the +DECK and all CMZ comments including the Author line.

```
          ====> The KUIP EDIT server: KUESVR <==========
```

The Kuip Edit Server (KUESVR) makes a-synchronous editing of files during the execution of CMZ possible. To use the KUESVR, CMZ must be able to find the executable 'kuesvr' somewhere in your search path. Also the editor you want to use must not be plain vi (or for that matter any editor that runs in the same window as the application). Emacs and HP's VED are "save" editors. When your system supports X windows then any editor can be made safe by specifying as editor (see HOST_EDITOR) 'xterm -e editor' (where editor may be vi).

7.1.6 READ dklist

`DKLIST` C “[-V vers_num][-S][-H] (-B | List of the decks to be browsed)” `D='␣'`

Browse one or more decks with the local browser or pager. On Unix and Apollo systems the current host pager can be defined via the environment variable CMZPAGER. If CMZPAGER is not set, the value set by the Kuip command HOST_PAGER is used. On VMS systems "TYPE/PAGE" is used unless the logical name CMZ$PAGER is defined. The deck(s) to be browsed can be specified by a DKLIST (see DECKLIST) or by option -B. If option -B is specified then browse all decks in the decklist buffer (see BUFFER). If option -V vers_num is given then browse the decks belonging to version vers_num. If option -S is given then all +IF,

```
    +SEQ directives are resolved.
```

By default the CMZ history records, after the +DECK lines, are listed, but can be suppressed by giving the option -H. On Apollo and Unix machines it is possible to give other CMZ commands while the DECK is being shown. In that case, if the CMZPAGER variable is not defined see the help for HOST_PAGER, otherwise for X window machines the CMZPAGER variable must be set to "xterm":

```
     setenv CMZPAGER xterm     (csh)
     export CMZPAGER=xterm     (ksh)
     DEF CMZ$PAGER EDT         (VMS)
```

```
    Ex.  READ deck       ; browse DECK
         READ -S deck    ; idem as above, but with all +IF and +SEQ
                           directives resolved
         READ -B         ; browse all decks in the decklist buffer
         READ -V 1.00/05 ; browse all deck belonging to version 1.00/05
```

7.1.7 CXTRACT dklist

`DKLIST` C "[-H[L]][-U][-R] (-P | [-P][-V vers_num] -B | [-K] $USE | -F Text file | [-P][-N][-V vers_num] List of decks)" `D='␣'`

Write one or more decks onto the default extract file. The deck(s) to be written can be specified by a DKLIST (see DECKLIST), or by the options -P, -B, $USE or -F Text file. If option -P is given then all patches selected by the PILOT command will be put on the compile file. If option -B is given then all decks in the decklist buffer (see BUFFER) will be put on the compile file. If option -V vers_num is given then all decks belonging to version vers_num will be put on the compile file. If option -P precedes any other option or the DKLIST then the patches selected by the PILOT command will be used together with the control options (set by SELECT) to test IF= parameters on the CMZ directives. By default the CMZ history records will be written to the compile file. Set option -H to prevent the writing of the history records. If option -HL is set only the last CMZ history record is written to the compile file. By default the USE bank is taken into account. If option -U is set then the corrections referenced by the USE bank are ignored. If option -N is given the deck is taken from the USE bank. If option -R is set then remove all decks after processing from the USE bank. If option -F is set then the decks in the referenced file are processed. If no +DECK line is found in the file, the default name is BLANKDEK. Option -U is implied for the decks read using -F. When $USE is given as decklist then process all the decks referenced in the USE bank. The default extract file name can be changed using the command : SET filename -F language_name. Using the option -LAN of the command SET, the user can select the list of languages to be considered. By default F77 is selected (see HELP SET). The options -U, -H, -N, -R and -V must precede the DKLIST or any other option.

```
Ex.  CXTRACT deck1          ; write DECK1 to the default extract file
                              (if the corresponding language is selected)
     CXTRACT *              ; write all decks (whose language is selected)
                              in the CWD to the extract file
     CXTRACT -H *           ; idem, no CMZ history records
     CXTRACT -P             ; write all patches selected by PILOT
     CXTRACT -HP            ; idem, but without CMZ history records
     CXTRACT -P deck1       ; write DECK1 to the extract file, items in
                              PILOT bank are used together with SELECT
                              items for testing IF= sequences
     CXTRACT -B             ; write all decks in the decklist buffer
     CXTRACT //             ; write all decks in all patches in all files
     CXTRACT -V 1.23/01 *   ; write version 1.23/01 to the default extract
                              file
     CXTRACT $USE           ; process all decks referenced in the USE bank
     CXTRACT -K $USE        ; process all decks referenced in the USE bank
                              and all decks of the connected files containing
                              sequences referenced in the USE bank
     CXTRACT -U *           ; process all decks whithout using corrections
                              in USE bank
     CXTRACT -R *           ; process all decks using corrections then
                              remove decks from the USE bank
     CXTRACT -N pat1/new2   ; write deck NEW2 in patch PAT1 in the USE bank
                              to the extract file
     CXTRACT -FR f.car $USE; process decks in the file F.CAR then remove
                              them from the USE bank and process the
                              remaining decks referenced in the USE bank
```

7.1.8 CCOMPILE dklist

`DKLIST` C "[-H[L]][-U][-R] (-P | [-P][-V vers_num] -B | [-K] $USE | -F Text file | [-P][-N][-V vers_num] List of decks)" `D='␣'`

Write one or more decks onto the default extract file and compile

the file with the compiler corresponding to the language specified in the T= directive. The deck(s) to be compiled can be specified by a DKLIST (see DECKLIST) or by the options -P, -B, $USE or -F text file. If option -P is given then all patches selected by the PILOT command will be compiled. If option -B is given then all decks in the decklist buffer (see BUFFER) will be compiled. If option -V vers_num is given then the decks belonging to version vers_num will be compiled. If option -P precedes any other option or the DKLIST then the patches selected by the PILOT command will be used together with the control options (set by SELECT) to test IF= conditionals on the CMZ directives. By default the CMZ history records will be written to the extract file. Set option -H to prevent the writing of the history records. If option -HL is set only the last CMZ history record is written to the extract file. By default the USE bank is taken into account. If option -U is set then the corrections referenced by the USE bank are ignored. If option -N is given the deck is taken from the USE bank. If option -R is set then remove all decks after processing from the USE bank. If option -F is set then the decks in the referenced file are processed. If no +DECK line is found in the file, the default name is BLANKDEK. Option -U is implied for the decks read using -F. When $USE is given as decklist then process all the decks referenced in the USE bank. The default extract file name can be changed using the command : SET filename -F language_name. Using the option -LAN of the command SET, the user can select the list of languages to be considered. By default F77 is selected (see HELP SET). The options -U, -R, -H, -N and -V must precede the DKLIST or any other option.

```
Ex.  CCOMPILE *             ; compile all decks (whose language is selected)
                              in the CWD
     CCOMPILE -H *          ; idem, no CMZ history records
     CCOMPILE -P            ; compile all patches selected by PILOT
     CCOMPILE -HP           ; idem, but without CMZ history records
     CCOMPILE -P deck1      ; compile DECK1, items in PILOT bank are used
                              together with SELECT items for testing
                              IF= sequences
     CCOMPILE -B            ; compile all decks in the decklist buffer
     CCOMPILE //            ; all decks in all patches in all files
     CCOMPILE -V 2.01/05 *  ; compile all decks from version 2.01/05
     CCOMPILE $USE          ; compile all decks in the USE bank
     CCOMPILE -K $USE       ; process all decks referenced in the USE bank
                              and all decks of the connected files containing
                              sequences referenced in the USE bank
     CCOMPILE -U *          ; compile all decks in the CWD, do not use
                              corrections in USE bank
     CCOMPILE -R *          ; idem, use corrections then remove decks from
                              the USE bank
     CCOMPILE -N pat1/new2  ; compile deck NEW2 in patch PAT1 in the USE
                              bank
     CCOMPILE -FR f.car $USE; process decks in the file F.CAR then remove
                              them from the USE bank and process the
                              remaining decks referenced in the USE bank
```

7.1.9 CLIB dklist

```
DKLIST   C    “[-H[L]][-U][-R] (-P | [-P][-V vers_num] -B | [-K] $USE | -F Text file | [-P][-N][-V
              vers_num] List of decks)”  D='␣'
```

Write one or more decks onto the default extract file, compile and archive. Uses the local compiler and librarian (archiver). The deck(s) to be compiled and archived can be specified by a DKLIST (see DECKLIST) or by the options -P, -B, $USE or -F Text file. If option -P is given then all patches selected by the PILOT command will be compiled and archived. If option -B is given then all decks in the decklist buffer (see BUFFER) will be compiled and archived. If option -V vers_num is given then compile and archive the decks belonging to version vers_num. If option -P precedes any other option or the DKLIST then the patches selected by the PILOT command will be used together with the control options (set by SELECT) to test IF= conditionals on the CMZ directives. By default the CMZ history records will be written to the extract file. Set option -H to prevent the writing of the history records. If option -HL is set only the last CMZ history record is written to the extract file. By default the USE bank is taken into account. If option -U is set then the corrections referenced by the USE bank are ignored. If option -N is given the deck is taken from the USE bank. If option -R is set then remove all decks after processing from the USE bank. If option -F is set then the decks in the referenced file are processed. If no +DECK line is found in the file, the default name is BLANKDEK. Option -U is implied for the decks read using -F. When $USE is given as decklist then process all the decks referenced in the USE bank. The default extract file name and the default library file name can be changed using the command : SET ..... -F or -L. Using the option -LAN of the command SET, the user can select the list of languages to be considered. By default F77 is selected (see HELP SET). The options -U, -R, -H, -N and -V must precede the DKLIST or any other option. On VAX system, if the option -C is given a new cycle of the library is created instead of updating the previous cycle.

```
Ex.  CLIB  deck1 deck2  ; compile and archive DECK1 and DECK2 (if their
                          corresponding languages are selected)
     CLIB  *            ; compile and archive all decks in the CWD
     CLIB -P            ; compile and archive all patches selected by
                          PILOT
     CLIB -HP deck1     ; compile and archive DECK1, items in PILOT bank
                          are used together with SELECT items for testing
                          IF= sequences. No CMZ history records
     CLIB -B            ; compile and archive all decks in the decklist
                          buffer
     CLIB //            ; all decks in all patches in all files
     CLIB -V 1.44/01 *  ; compile and archive all decks of version 1.44/01
     CLIB $USE          ; process all decks in the USE bank
     CLIB -K $USE       ; process all decks referenced in the USE bank
                          and all decks of the connected files containing
                          sequences referenced in the USE bank
     CLIB -UR deck1     ; compile and archive DECK1 without corrections
                          and remove DECK1 from the USE bank
     CLIB -N pat1/new2  ; compile and archive deck NEW2 in patch PAT1
                          in the USE bank
     CLIB -FR f.car $USE; process decks in the file F.CAR then remove them
                          from the USE bank and process the remaining
                          decks referenced in the USE bank
```

7.1.10 CMAKE [opt]

```
OPT  C  “Options”  D='␣' Minus
```

Possible `OPT` values are:

```
-N | -A | -O | -P | -I [history filename] | -V [vers_num1] [-VD vers_num2]
```

Recompile decks that are out of date

compared to the date and time written on the 'FILE.history' file (the history file is written by the command 'LMODS -O'). If this file doesn't exist and the option '-I' is not specified recompile decks that are out of date compared to the default library. CMAKE compares the time of last change written in the history file (or of the date and time of the default library if the history file does not exist) with the time of last change of all the decks in the CWD and its sub-directories. Decks that are more recent than the time of the last change will be written to the default extract file, recompiled and replaced in the library. If the option '-V vers_num1' is given all decks having a version number greater than 'vers_num1' or no version set are recompiled. If 'vers_num1' is not specified, the version number written on the history file is taken. If the option '-V vers_num1 -VD vers_num2' is given all decks having changed between 'vers_num1' and 'vers_num2' are recompiled. The resulting code belongs to the version 'vers_num2'. See the help of the command SEQUENCES for the most efficient way of using the command 'cmake -V v1 -VD v2'. If option -O is set the decks are compared to their default object files. In this case the decks that are out of date will be recompiled but not replaced in the library.

CMAKE checks for every SEQ it finds the date of change of the KEEP (stored when loading the sequence definitions in memory). When a number of KEEPs are stored in one deck they will all get the same date and time of change from the decks history records. Due to this fact it may happen, although you changed only one keep in a deck containing several KEEPs, that many more decks will be recompiled than you expected. This is because all KEEPs in the deck get the same, new, modification date.

The -O option can only be used when each deck will be written to its own extract file (e.g. SET ftn/*.ftn -F, see HELP SET). If option -N is given then the decks that need recompilation will only be listed but not recompiled. If option -A is given then all connected files will be scanned. If option -P is given then the patches selected by the PILOT command will be used together with the control options (set by SELECT) to test IF= conditionals on the CMZ directives.

```
Ex.  CMAKE          ; recompile all decks in the CWD and its
                      sub-directories that are out of date
     CMAKE -O       ; recompile all decks that are out of date
                      compared to their object files
     CMAKE -PO      ; idem, but for patches selected by PILOT
     CMAKE -N       ; list decks that need to be recompiled
     CMAKE -NA      ; idem, but scan all connected files
     CMAKE -A       ; recompile all decks in all connected files
                      that are out of date
     CMAKE -V       ; recompile all decks modified since the version
                      number on the history file.
     CMAKE -V 2.03/00      ; recompile all decks modified since the version
                             2.03/00 of the file
     CMAKE -V 2.03/00 -VD 2.05/00  ;
                             recompile all decks modified between the
                             versions 2.03/00 and 2.05/00. The resulting
                             code belongs to the 2.05/00 version
     CMAKE -I user.history ; recompile all decks that are out of date
```

```
                                         compared to the date written on
                                         'user.history' file.
```

7.1.11 LMODS [author frdate todate]

AUTHOR	C	"[-]Author name | -O [hisfile]"	D='*'
FRDATE	C	"(-V[I] Version1 | From date)"	D='*'
TODATE	C	"(Version2 | To date)"	D='*'

List all decks modified by AUTHOR between FRDATE and TODATE. -AUTHOR is interpreted as all users except AUTHOR. AUTHOR can also be the special string ME. FRDATE or TODATE can also be the special strings TODAY, YESTERDAY, MONTH and LASTMONTH. If AUTHOR is set to AUTHORS then all authors that have recently made changes (i.e. appear on the top history record) are listed, including the number of times they made a change (FRDATE and TODATE are ignored in this case). If FRDATE is set to LAST then the last deck changed by AUTHOR is listed. If option -V is given, list all decks modified by AUTHOR between the versions Version1 and Version2. Option -VI is the same as option -V except that it includes the decks belonging to Version1 and Version2.

```
    Ex.  LMODS                   ; list all decks
         LMODS *  12/08/87       ; list all decks modified after
                                   12/08/87 by all authors
         LMODS *  TODAY          ; list all mods made today by all authors
         LMODS Smith  13.15      ; list all mods by user Smith since today
                                   from 13.15 pm
         LMODS *  11/07/87.13.25  11/07/87.16.15
                                 ; shows complete date and time syntax
         LMODS *   LAST          ; list last deck changed
         LMODS jan LAST          ; list last deck changed by JAN
         LMODS AUTHORS           ; list all authors that have made changes
         LMODS -Smith -V 1.28/03 ; list all mods made by all users except
                                   Smith after the version 1.28/03
         LMODS -O                ; write a 'FILE.history' file which contains
                                   the date and time of the last deck changed.
                                   'FILE' is the name of the connected file.
         LMODS -O hbook.lastmod  ; idem as above but write on 'hbook.lastmod'
```

If AUTHOR is H or HISTORY or H.AUTHOR then the complete history is listed.

```
    Ex.  Lmods H.Louis 11/01/88  ; list full history of all the decks
                                   modified by user Louis after 11/01/88
```

7.1.12 COR orig [desc corfil]

`ORIG`	C	“[-O][-H] Name of the original deck/patch/file | -V vers_num”
`DESC`	C	“Name of the descendant deck/patch/file | -VD vers_num2”
`CORFIL`	C	“-F Name of file on which to write the corrections”

Generate a correction file by comparing two decks, patches, files or versions. The correction file will be stored in the directory //file/$CORR. Each correction file will be a DECK with the name CORR1, CORR2, etc. When CORFIL is specified the correction file will be written on the file CORFIL. If option -O is set CORR compares only the decks available in the descendant file with the decks in the original file. IF the descendant file is compared to several original files, the option -O may be omitted. If option -H is set CORR generates a correction file ignoring the CMZ history records in the line number. If option -V vers_num is given CORR will compare version vers_num with the current highest cycles of all decks. If in addition -VD vers_num2 is given then vers_num2 is compared against vers_num.

```
    Ex.  COR deck1 deck2            ; compares DECK1 and DECK2 and stores
                                      the corrections in the (new)
                                      patch $CORR
         COR patch1 patch2          ; compares PATCH1 and PATCH2
         COR //file1 //file2        ; compares two entire files
         COR -O //file1 //file2     ; compares only the decks available in
                                      FILE2 with the decks in FILE1
         COR -O //f1 //f2 //f3 //usf ; compares decks available in USF with
                                      decks in F1, F2 and F3
         COR //f1 //f2 //f3 //usf   ; idem as above
         COR // //userf             ; Compares decks available in USERF
                                      with decks in all connected files
         COR pat1 pat2 -F corr.cra  ; compares two patches and writes the
                                      corrections to file CORR.CRA
         COR *                      ; compares highest cycle, max, of all
                                      decks in the current directory with
                                      cycle max-1.
         COR deck                   ; compares DECK;max-1 and DECK;max
         COR deck;6 -F corr.cra     ; compares DECK;6     and DECK;max
         COR -V 1.19/01 deck1       ; compares the highest cycle of DECK1
                                      with the version 1.19/01 of DECK1
         COR -V 1.19/01 -VD 1.23/03 ; compares version 1.23/03
                                      with 1.19/01
         COR -O -V 1.19/01 //file1 //file2 ; compares only the decks
                                             available in FILE2 with the
                                             decks belonging to the version
                                             1.19/01 in FILE1.
```

7.1.13 USE [opt corlis]

```
OPT      C   “Options”   D='?'
CORLIS   C   “List of correction files”   D='␣'
```

Possible OPT values are:

```
.  | -F | -E | -O | -P | -R | -I
```

Read the contents of correction files into memory (the USE bank). The corrections will be merged and clashes will be detected. Use the -P option to specify a correction deck in a CMZ file. If no argument is given then a list of all patches and decks referenced in the USE bank will be given. Use option -E to view the contents of the USE bank for all decks or for a specific deck in the local pager or browser (see command READ). To write the merged corrections to a file use option -F filename. A correction file which does not contain CMZ history records (created by a PATCHY user) can be specified with the -O option. Correction files loaded with the -O option can not be merged with other correction files. Use option -R to remove a deck or a list of decks from the USE bank. Use option . to clear all corrections from memory. If option -I is set, the clashes between corrections are ignored, the first one loaded in memory is taken into account.

```
Ex.  USE                     ; list patches and decks referenced
                               in the USE bank
     USE .                   ; clear all corrections from the USE bank
     USE file1.cra file2.cra ; load corrections from files into
                               the USE bank
     USE -I file1.cra file2.cra
                             ; idem as above, but in case of clashes the
                               corrections are taken from FILE1.CRA
     USE -O file1.cra file2.cra
                             ; load corrections without history records
                               into the USE bank
     USE file1.cra test.cra -P //aap/$corr
                             ; also load corrections from the directory
                               $CORR from the file AAP (AAP.CMZ must
                               have been connected with command FILE)
     USE -E patch2/deck1     ; shows the result of the merging of all
                               corrections for DECK1 in PATCH2
     USE -E                  ; idem as above but for all decks
     USE -F new.cra          ; write the merged corrections to file
                               NEW.CRA
     USE -R deck1 deck2      ; remove DECK1 and DECK2 from the USE bank
```

7.1.14 UPDATE dklist

DKLIST C “[-E] ($USE | [-REF //reffil] List of decks to be updated)” D='␣'
Apply the corrections stored in the USE bank
to the decks specified by DKLIST (see DECKLIST). For all updated decks new cycles will be created. If option -E is specified the result of the update will be shown in the editor. Option -E must precede the DKLIST. When $USE is given as decklist then process all the decks referenced in the USE bank. When the option '-REF //reffil' is set the corrections in the USE bank are applied to the referenced file REFFIL and the result is saved in the current file. The options '$USE' and '-REF ...' are mutual exclusive.

```
Ex.  UPDATE deck1 deck2           ; apply the corrections in the USE bank
                                    to DECK1 and DECK2
     UPDATE //myfile              ; apply the corrections in the USE bank
                                    to the file MYFILE.
     UPDATE -E *                  ; apply the correction in the USE bank
                                    to all patches and decks in the CWD
                                    and view the result in the editor
     UPDATE -E $USE               ; process all decks referenced in the
                                    USE bank and view the result in the
                                    editor
     UPDATE -REF //reffil *       ; apply the corrections in the USE bank
                                    to the file REFFIL and save the
                                    result in the current file.
     UPDATE -REF //reffil //myfile ; apply the corrections in the USE bank
                                    to the file REFFIL and save the
                                    result in the file MYFILE.
```

7.1.15 VERSION [versn]

VERSN C “R=[-D][+ | +0 | +00 | -S[A] vers_num][-E][-F filename] ,-L ” D='?'
Mark highest cycles of all decks in the current file as belonging to one version. A version is identified by its version number. A version number consists of three parts: the version, the release and the level part. These three parts are combined in the following, fixed, way: x[x].yy/zz.

```
Ex.   1.01/02
     13.03/11
```

If option + is given the level part of the version number is increased by one. If option +0 is given the release part is increased by one and the level part reset to 00. And if option +00 is given the version part is increased by one and the release and level parts reset to 00. With option -S vers_num the base version number can be set (after this the version number can be changed with +, +0 or +00) or the stamping of the file is forced even if no decks were changed (new TITLE is created and a new cycle of all decks is created if option SA is chosen). If option -L is set a list of all versions stored in the CMZ file is given. Without option the current highest version number is given. In case the options -S vers_num, +, +0 or +00 are given, VERSION does a global KEEP on the whole CMZ file to lock the highest cycle of all decks. The new version number is written in the decks that do not yet have a version number. A new deck with the name V[x]x_yy, corresponding to the new version number is created in directory $VERSION (if directory $VERSION does not yet exist CMZ will create one). In this new deck comments concerning the new version may be added. If option -E is set the deck V[x]x_yy is saved without appearing in the window editor. If option -F is set the text contained in FILENAME is added to the deck V[x]x_yy. If option -D is set a list of all modified decks is added to the deck V[x]x_yy. The version information can be used by the commands CXTRACT, CCOMPILE, CLIB, CAR, EDIT, CTOT, DELETE, CORR and DIFF.

```
Ex.  VERSION -S 1.00/00 ; set the base version number on the current file
     VERSION +          ; mark the current file as version x.y/z+1
     VERSION +0         ; mark the current file as version x.y+1/00
     VERSION            ; list the current highest version number
     VERSION -L         ; list all versions stored in the file
```

7.2 Interface to Data Base manager

7.2.1 CREATE fname [chopt nrecs]

FNAME	C	“Name of the CMZ file to be created” D='␣'
CHOPT	C	“Option” D='␣' Minus
NRECS	I	“Maximum number of records that the file can contain” D=32000

Possible CHOPT values are:

-C

Create a new CMZ file
and set the current directory to the top directory of this file. NRECS specifies the maximum number of records that the file eventually can contain. The default of 32000 records (of 512 bytes each) is enough for about 16Mb. If the option -C is set, the file will be compressed using the gzip algorithm (see command RIGTHS). Typical compression factors range from 3 to 6.

```
    Ex.  CREATE myfile        ; create the file MYFILE.CMZ
         CREATE biglib 48000  ; create the file BIGLIB.CMZ which can contain
                                up to 24Mb of data
```

7.2.2 FILE fname [opt exec]

FNAME	C	“Name of the CMZ file to be attached” D='␣'
OPT	C	“Option” D='-U' Minus
EXEC	C	“-X [macro [args]]”

Possible OPT values are:

-U | -R

Attach an existing CMZ file
and set the current directory to the top directory of this file. If option -U is set, the file will be attached in Update mode. If option -R is specified, the file will be attached in Readonly mode. If option -X is set then the macro MACRO with (optional) arguments ARGS will be executed. If no macro is given then by default the macro(s) in deck /$KUMACS/FLOGON will be executed. For more on macro execution see command EXEC. If options -U and -R not set, the file will be attached in Update mode if no lock set, on the contrary it will be connected in Readonly mode. A file is locked when connected in Update mode. Note that a file will be seen as locked if it is transferred via FTP and somebody else using this file in Update mode. To unlock the file use the option -U explicitly.

```
    Ex.  FILE hbook         ; connect file HBOOK.CMZ in update mode
         FILE hbook -R      ; connect file HBOOK.CMZ in readonly mode
         FILE kuip  -X load ; connect file KUIP.CMZ and execute macro LOAD
```

7.2.3 RELEASE [fname]

FNAME	C	“Name of the file to be released” D='␣'

Close and disconnect file FILENAME from the CMZ session.

```
    Ex.  RELEASE file1     ; release file FILE1
         RELEASE *         ; release all connected files
         RELEASE           ; release the current file
```

7.2.4 DIRECTORY [opt path]

```
OPT    C  “Option”  D='␣' Minus
PATH   C  “Name of directory or deck”  D='␣'
```

Possible `OPT` values are:

```
-L | -A
```

List the contents of a directory. PATH may be a directory or a deck. If you specify a directory, the decks in that directory will be listed. If you specify a deck, the highest cycle of that deck will be listed. Wildcarding is permitted (if wildcarding is used, each name is assumed to be a deck). See for the wildcarding syntax command REGEXP. The symbol "..." (ellipses) may be used in PATH to indicate that any number of directory names (including zero) may appear in its place. If PATH is omitted, the current directory will be listed. If option -L is set the listing will be in long format, giving deckname, cycle number, date-time of creation and the length of the deck in bytes. Option -A has the same effect as option -L except that it adds the number of lines per deck to the list (slower than -L) and the value of the compression factor when the file was created with the -C option (see command CREATE).

```
    Ex.  DIR dir1        ; list contents of directory DIR1
         DIR deck2       ; list highest cycle of DECK2
         DIR -L deck2;?* ; list all cycles of DECK2 in long format
         DIR cmd?*       ; list all cycles of all decks beginning with CMD
         DIR [^c]?*      ; list all cycles of all decks NOT beginning with C
         DIR /dir2/d?*   ; list all cycles of all decks in DIR2 beginning
                           with D
         DIR ...         ; list the contents of all directories
         DIR .../cd?*    ; list all cycles of all decks, in all directories,
                           beginning with CD
         DIR //          ; list the names of all currently connected files
```

7.2.5 CDIR [path]

```
PATH   C  “Pathname”  D='␣'
```

Change the current working directory (CWD) to PATH. If no pathname is given print the pathname of the CWD.

```
    Ex.  CD dir1         ; make DIR1 the new CWD
         CD //file1/dir2 ; make //FILE1/DIR2 the new CWD (FILE1 must be
                           connected with FILE)
         CD              ; print the name of the CWD
```

7.2.6 MOVE obj opt ref [type]

```
OBJ    C  “Object to move”  D='␣'
OPT    C  “Option (-A=after or -B=before)”  D='␣' Minus
REF    C  “Reference object”  D='␣'
TYPE   C  “Type of object (-P=patch, -D=deck)”  D='-P' Minus
```

Possible `OPT` values are:

```
-A | -B
```

Possible `TYPE` values are:

```
-P | -D
```

Move an object OBJ before (-B) or after (-A) object REF. The type of object to move can either be a directory or a deck, depending on TYPE. The default object type is directory.

```
    Ex.  MOVE  dir1  -A dir5     ; move DIR1 after DIR5
         MOVE  deck8 -B deck1 -D ; move DECK8 before DECK1
```

7.2.7 DELETE dklist

`DKLIST` C "[-V vers_num | -O | -I] (-B | List of decks to be deleted)" `D='␣'`

Delete decks from a CMZ file. The deck(s) to be deleted can be specified by DKLIST (see DECKLIST) or by option -B. If option -B is given then all decks in the decklist buffer (see BUFFER) will be deleted. If option -V vers_num is given then the decks belonging to version vers_num that are not associated with any other version will be deleted. If option -O is given then the contents of the decks in the DKLIST will be deleted except for the history records (deck is obsoleted). Wildcarding is permitted. If wildcarding is used, you will be asked to confirm each deletion (for help on possible answers type "?"). See for the wildcarding syntax command REGEXP.

```
Ex.  DELETE deck1           ; delete highest cycle of DECK1
     DELETE deck1.deckN     ; delete DECK1 to DECKN included
     DELETE deck1;2         ; delete cycle 2 of DECK1
     DELETE cm?* deck2      ; delete all cycles of all decks starting with
                              CM and delete the highest cycle of DECK2
     DELETE -B              ; delete all decks in the decklist buffer
     DELETE -V 2.01/02 *    ; delete all decks belonging to version
                              2.01/02
                            ; that do not belong to any other version
     DELETE -O deck1        ; delete contents of DECK1 (saving history
                              records), i.e. deck is obsolete
     DELETE -I *            ; delete the highest cycle of all decks
                              that have a previous cycle identical to
                              the highest cycle
```

7.2.8 PURGE [keep opt]

`KEEP` C "Number of cycles to keep" `D='1'`
`OPT` C "Option" `D='␣' Minus`

Possible `OPT` values are:
`-A`

Purge all but the highest cycle of all decks in the current directory. When option KEEP is given the KEEP highest cycles are kept. By default KEEP=1. If option -A is set then all directories in all connected files are scanned for decks to be purged.

```
Ex.  PURGE          ; purge all but the highest cycle
     PURGE 2        ; purge all but the 2 highest cycles
     PURGE -A       ; purge in all directories in all connected files
```

7.2.9 COPY source target

SOURCE	C	"[-C] (-B | Name of the source deck)"
TARGET	C	"Name of the target (deck or dir)"

Copy deck(s) from the source pathname to the target pathname. Wildcarding is permitted in the source pathname. If wildcarding is used, the target has to be a directory. Wildcarding only takes the highest cycle of a deck. The source pathname may also be specified by a decklist (see DECKLIST) or by option -B. In that case wildcarding may not be used. If option -B is given then all decks in the decklist buffer (see BUFFER) will be copied. In case the target is a deck, the deck name on the +DECK directive of the source will be changed to the target deck name. A shorthand for the current working directory is ".". For the wildcarding syntax see command REGEXP. By default the version field of the copied decks will not be cleared and the copied decks will be locked when a version exists. If option -C is given then the date and time of the copied decks are set to the current ones.

```
Ex.  COPY deck1 deck2          ; copy DECK1 to DECK2
     COPY /dir1/deck1.deckN .  ; copy DECK1 to DECKN included to the CWD
     COPY /dir1/de?* .         ; copy the highest cycle of all decks
                                 starting with DE in DIR1 to the CWD
     COPY -C -B .              ; copy the decks contained in the decklist
                                 buffer to the CWD and add a new *CMZ comment.
```

7.2.10 MKDIR newdir

NEWDIR C "Name of new directory" D='␣'

Create a new directory as a daughter of the current working directory. The CWD must be a top-directory.

```
Ex.  MKDIR dir2        ; create new directory DIR2
     MKDIR DIRC,T=C++ ; create a new directory DIRC of type C++
```

7.2.11 RMDIR dirnam

DIRNAM C "Name of the directory to delete" D='␣'

Delete a directory in the current working directory. This is only possible when the directory is empty, apart from the 00_PATCH (make a directory empty with the DELETE command).

```
Ex.  RMDIR dir1     ; delete directory DIR1, DIR1 must be empty
```

7.2.12 CPTREE topd1 topd2

TOPD1	C	"Name of source file" D='␣'
TOPD2	C	"Name of target file" D='␣'

Copy file TOPD1 to TOPD2. Note that TOPD2 must first have been created with the CREATE command. This command is useful to compact a fragmented CMZ file.

```
Ex.  CPTREE //top1 //top2    ; copy all directories, decks and cycles
                               from //TOP1 to //TOP2
```

7.2.13 STAT topdir

TOPDIR C "Name of the top directory" D='␣'

Print statistics about space usage of all directories in the CMZ file.

```
Ex.  STAT //myfile   ; print statistics about file MYFILE
```

7.2.14 EXIT [optlist]

OPTLIST C "[-S]" D='␣'

CMZ exit command. If option -S set no synchronization will be done (see command SYNCHRONIZE).

7.3 IMPORT commands

7.3.1 ARC fname [dklist]

```
FNAME    C   “Name of the CMZ Ascii Readable file”  D='␣'
DKLIST   C   “[-C[C]][-A][-S] List of patches or decks”  D='*'
```

Convert a CMZ Ascii Readable file to a CMZ file. Patches become directories with the decks as leaves. Using the optional parameter DKLIST it is possible to read only a subset of FNAME (see DECKLIST). If option -C is set comments in columns 73-80 are removed from statements which are not comment lines. If option -CC is set comments in columns 73-80 are removed from all statements including comment lines. If option -A is set the AUTHOR field of NEW history records will be filled in with the CMZ user name (see AUTHOR). If option -S is set a multi CMZ Ascii Readable file (i.e. a file with more than one +TITLE line) will be split into separate CMZ files. By default the extra +TITLE lines will be skipped.

```
Ex.  ARC a.car                ; read complete file A.CAR.
     ARC a.car -C             ; read A.CAR and remove comments in
                                columns 73-80
     ARC a.car -A             ; read A.CAR and set the author field of
                                new history records
     ARC a.car patch1         ; read only PATCH1 from file A.CAR
     ARC a.car patch1.patch2  ; read from PATCH1 to PATCH2
     ARC a.car patch/deck1.   ; read all decks from DECK1 of PATCH
     ARC a.car patch1 patch2/.deck1   ; read PATCH1 and all decks
                                        up to DECK1 included of PATCH2
```

7.3.2 CIMPORT tfile

```
TFILE   C   “[-A][-[I]S][-K][-T type][-E] Name of the source files”  D='␣'
```

Import one or more source files into a CMZ file. The file becomes a DECK with the type depending on the current language (+DECK followed by T= 'type of the current language') in the current working directory, with the DECK name equal to the first 32 characters of the source filename, minus the filename extension. If option -A is set, the AUTHOR field of the history records will be filled in with the CMZ user name (see AUTHOR). If option -S is set then CMZ will search for matches between the sequence definitions currently stored in memory and identical lines in the source file. When a match is found, the corresponding lines will be replaced by a +SEQ,NAME line, where NAME is the name of the sequence in memory. If option -T is set, the directive "T=type" will be written on the +DECK card. If option -E is set, the result of the command is written to an edit file.

```
Ex.  CIMPORT foo.tex          ; becomes deck FOO
     CIMPORT -K inc.h         : becomes deck $INC
                                (+KEEP card generated instead of +DECK)
     CIMPORT -T latex foo.tex ; becomes deck FOO,T=LATEX.
     CIMPORT -A hello.tex     ; idem as above but author field is
                                filled in
     CIMPORT -S hello.cdf     ; becomes deck hello  and matching
                                sequences will be replaced by a '+SEQ'
                                line. The include lines are also replaced
                                by the corresponding +SEQ command if the
                                include filename matches a sequence name.
     CIMPORT -IS hello.c      ; idem as above but only include lines are
                                replaced (faster than the -S option when
                                source files were not extracted from cmz).
     CIMPORT -T C *.c         ; on UNIX, Apollo, VMS and IBM/VM systems,
                                the source file name may contain regular
                                expression.
```

7.3.3 TTOC fname

`FNAME` C “[-W][-A] Name of the TEXT files” `D='␣'`

Import one or more plain TEXT files into a CMZ file. The TEXT file becomes a DECK of type DATA (+DECK followed by T=DATA) in the current working directory, with the DECK name equal to the first 32 characters of the TEXT filename, minus the filename extension. If the TEXT file is more 10,000 lines long, it is split in several decks. A suffix "_id" is added to the deck name.(see command CTOT to recover the file) Option -W enables line wrapping at 80 characters. If option -A is set, the AUTHOR field of the history records will be filled in with the CMZ user name (see AUTHOR).

```
Ex.  TTOC foo.cdf a1.txt  ; becomes decks FOO and A1
     TTOC -W com.txt      ; idem as above but perform line wrapping
     TTOC longfile.txt    ; becomes decks longfi_1, longfi_2 etc...
```

7.3.4 SYNCHRONIZE [option]

`OPTION` C “[-ON][-OFF]” `D='?'  Minus`

When invoking the command SYNCHRONIZE with argument "*" , the source files of which the path is defined via the command "SET ... -FE" and which were changed more recently than the corresponding decks are imported into the corresponding CMZ file(s). If the command SYNCHRONIZE is set ON, the connected CMZ files are always updated when exiting from CMZ. If option OFF set, no update of the connected files will be done when exiting from CMZ. NB: Only available on UNIX systems. See also the EXIT and 'SET ... -FE' commands.

```
Ex.  SYNCHRONIZE     ; show the option chosen for the updating of the
                       cmz files when exiting
     SYNCHRONIZE *   ; force the update of the connected CMZ files
                       (even if OFF set)
```

7.4 EXPORT commands

7.4.1 CLIST fname [dklist]

`FNAME`	C	"File on which listing will be written"	`D='␣'`
`DKLIST`	C	"[nn][-A][-S][-H][-I][-E][-V vers_num] (-B | List of decks to list)"	`D='*'`

Write a listing of a CMZ file or directory on FNAME. The listing shows the line count for each line and the deck and patch name. At the end of the listing a table of contents, a sorted index of all patches, decks, sequences and control options is given. By default when in the top directory the whole file will be listed, when in a subdirectory only that directory (patch) will be listed. A specific deck or set of decks and patches can be listed by using the optional parameter DKLIST (see DECKLIST). If option -B is given, then all decks in the decklist buffer (see BUFFER) will be written to the list file. If option -V vers_num is given then list the decks belonging to version vers_num. By default FORTRAN carriage control characters will be used to format the listing. Give option -A for ASCII carriage control characters. By default the CMZ history records, after the +DECK lines, are listed, but can be suppressed by giving the option -H. In case only an INDEX is desired give option -I. If option -E is set then the listing will be shown in the pager or browser (see command READ). If option -S is set every deck starts a new page. The default number of lines per page is 56. The number of lines per page can be changed by giving a new number nn. Options must precede the DKLIST or the -B option.

```
    Ex.  CLIST test.lis        ; list file or directory on list file TEST.LIS
                                 with FORTRAN control char. and 56 l/p
         CLIST test.lis -E     ; idem as above and view result in editor
         CLIST test.lis -A 104 ; list with ASCII car. cont. char. and 104 l/p
         CLIST test.lis -A -I 104
                               ; list only index with ASCII car. cont. char.
                                 and 104 l/p
         CLIST test.lis -AH 104 patch1 patch2
                               ; list PATCH1 and PATCH2 without history, with
                                 ASCII car. control and 104 l/p
         CLIST test.lis -AH 104 -B
                               ; idem as above with all decks from the deck-
                                 list buffer
         CLIST test.lis -AH 104 -V 1.24/01
                               ; idem as above with all decks belonging to
                                 version 1.24/01
```

7.4.2 CAR fname [dklist]

FNAME	C	"Name of the CMZ Ascii Readable file" D='␣'
DKLIST	C	"[-H[nn][L]][-A][-V vers_num][-W] (-B | $USE | List of patches)" D='*'

Convert a CMZ file to a CMZ Ascii Readable file. Directories become PATCHES. The patch(es) or deck(s) to be written can be specified by DKLIST (see DECKLIST) or by option -B. If option -B is given then all decks in the decklist buffer (see BUFFER) will be written on the CMZ Ascii Readable file. If option -V vers_num is given then write all decks belonging to version vers_num to the text file. When $USE is given as decklist then process all the decks referenced in the USE bank. By default CMZ comments after the +DECK lines are written to the text file. This may be changed by giving the option -H. If option -HL is set only the last CMZ comments are written to the CMZ Ascii Readable file. If option -Hnn, where nn is an integer less than 100, is set then the last nn CMZ comments with version numbers are written to the CMZ Ascii Readable file. By default only the highest cycle of each deck is exported. To export all cycles of all decks, to keep the history of the CMZ file, use option -A (can be in combination with -H). When the -W option is given, the command is equivalent to the command CTOY which converts a CMZ file into a CAR file compatible with Patchy. The options -H, -HL , -V vers_num, -A and -W must precede the DKLIST or the -B option.

```
Ex.  CAR file.car               ; write all decks and sub-directories
                                  from the current working directory to
                                  FILE.CAR
     CAR file.car patch1.patch2 ; write PATCH1 to PATCH2
     CAR file.car patch1 patch2/deck3.
                                 ; write PATCH1 and PATCH2 starting with
                                   DECK3 till end of patch
     CAR file.car -H patch1     ; write PATCH1 without CMZ comments
     CAR file.car -A            ; write all cycles of all patches or
                                  decks
     CAR file.car -V 2.02/04    ; write version 2.02/04 to FILE.CAR
     CAR file.car $USE          ; write all decks referenced in the USE
                                  bank to FILE.CAR
```

7.4.3 CTOY fname [dklist]

FNAME	C	"Name of the CMZ Ascii Readable file" D='␣'
DKLIST	C	"[-H[nn][L]][-A][-V vers_num] (-B | $USE | List of patches)" D='*'

Convert a CMZ file to a CAR file compatible with Patchy. This command is equivalent to the command 'CAR -W ...'. It enables line wrapping to 80 characters and expands in the CAR file the following sequences which are not defined by PATCHY : VERSQQ and VIDQQ in Fortran, VERSQQ, VIDQQ, TIMEQQ and DATEQQ in C. See the command CAR to get explanation on the possible options.

7.4.4 AXTRACT fname [dklist]

`FNAME`	C	"Name of the CMZ Ascii Readable file"	`D='␣'`
`DKLIST`	C	"[-C[C]][-H][-U] List of patches or decks"	`D='*'`

Convert a CMZ Ascii Readable file directly to a source file. Using the optional parameter DKLIST it is possible to convert only a subset of FNAME (see DECKLIST). If option -C is set, comments in columns 73-80 are removed from statements which are not comment lines. If option -CC is set, comments in columns 73-80 are removed from all statements including comment lines. Set option -H to prevent the writing of the history records. By default the USE bank is taken into account. If option -U is set then the corrections referenced by the USE bank are ignored. The source filename can be defined using the command: SET filename -F. The options -U, -H and -C[C] must precede the DKLIST.

```
Ex.  AXTRACT a.car                ; read complete file A.CAR
     AXTRACT a.car -C             ; idem as above but comments in columns
                                    73-80 are removed.
     AXTRACT a.car patch1         ; read only PATCH1 from file A.CAR
     AXTRACT a.car patch1.patch2  ; read from PATCH1 to PATCH2
     AXTRACT a.car patch/deck1.   ; read all decks from DECK1 of PATCH
     AXTRACT a.car patch1 patch2/.deck1   ; read PATCH1 and all decks
                                            up to DECK1 included of PATCH2
```

7.4.5 CTOT dklist

```
DKLIST   C   "[-S][-U][-R][-H][-N][-D][-V vers_num] (-P | -B | [-K] $USE | List of decks )"
             D='␣'
```

Write one or more decks as plain text onto the default text file. The deck(s) to be exported can be specified by a DKLIST (see DECKLIST), or by the options -P, -B or $USE. Option -D tells CTOT to write only patches or decks of type DATA or TEXT (+PATCH or +DECK followed by T=DATA or T=TEXT) to the default text file. If option -P is given then all patches and decks selected by the PILOT command will be scanned for the type specifiers DATA or TEXT. The found decks will be written to the default text file. If option -B is given then all decks in the decklist buffer (see BUFFER) will be written to the default text file. If option -V vers_num is given then write the decks belonging to version vers_num to text files. By default the +DECK directive and the history records will be stripped off. If option -H is set the history records will not be stripped off. If option -S is given then +IF's in the deck will be interpreted according to the active SELECT flags. Option -S is implied by -P. By default the USE bank is taken into account. If option -U is set then the corrections referenced by the USE bank are ignored. If option -N is given the deck is taken from the USE bank. If option -R is set then remove all decks after processing from the USE bank. When -K $USE is given as decklist then process all the decks referenced in the USE bank and all decks from the connected files containing sequences referenced in the USE bank. When $USE is given only decks referenced in the USE bank are processed. The default text file name is DECK_NAME.TXT. It can be changed using the command: SET filename -F TEXT. The options -S, -U, -R, -H, -N, -D, -T and -V must precede the DKLIST or any other option.

```
    Ex.  CTOT foo          ; write deck FOO to the default text file
         CTOT -S foo       ; interpret +IF's and write to the default text
                             file
         CTOT -D *         ; write all decks of type TEXT or DATA to the
                             default text file
         CTOT -B           ; write all decks in the decklist buffer to
                             the default text file
         CTOT -V 3.11/02 * ; write all decks from version 3.11/02 to the
                             default text file
         CTOT -R deck1     ; write deck DECK1 to the default text file
                             and remove DECK1 from the USE bank
         CTOT $USE         ; process all decks in the USE bank
         CTOT -K $USE      ; process all decks in the USE bank and all decks
                             of the connected files containing sequences
                             referenced in the USE bank.
```

To write onto a single text file all decks split by the command TTOC, follow the instructions below.

```
         SET  longfile.txt -d
         CTOT longfi?*     ; all decks LONGFI_1, LONGFI_2 ... are written on
                             the LONGFILE.TXT file.
```

7.5 UTILITIES commands

7.5.1 ALPHA_ORDER

Order all decks in the current working directory in alphabetical order.

7.5.2 BUFFER [optlis]

`OPTLIS` C "List of options" `D='?'`

Show or change the decklist buffer. This buffer is filled by the commands LMODS, GREP, FIND, UNDEF, CMAKE, TREE and UPDATE. The list of decks in this buffer can be used by the commands CXTRACT, CCOMPILE, CLIB, CTOT, EDIT, CLIST, GREP and UNDEF with the -B option. If no arguments are given then show the current decklist buffer. If the argument is a . then the decklist buffer will be cleared. To add a deck to the decklist buffer give the full pathname of that deck. Complete patches can be added by replacing the deck name by a * in the full pathname. To remove a deck from the decklist buffer give as argument the deck name preceded by a -.

```
Ex.  BUFFER                 ; show the current decklist buffer
     BUFFER .               ; clear the decklist buffer
     BUFFER //file/pat1/dec2 ; add //FILE/PAT1/DEC2 to the buffer
     BUFFER //file/pat2/*    ; add all decks in //FILE/PAT2 to the buffer
     BUFFER -deck1           ; remove DECK1 from the decklist buffer
```

7.5.3 DECKLIST

A number of commands allow as an argument a DECKLIST. The DECKLIST argument allows you to specify one or more sets of decks. A set of decks is specified by giving the first and last deck of the set separated by a . (the first and last deck are included in the set), e.g. deck1.deckN. If deck1 is the first deck in the current working directory (CWD) than deck1 may be omitted. If deckN happens to be the last deck in the CWD then deckN may be omitted.
If a set contains only a single deck name then wildcarding is permitted. See for the wildcarding syntax command REGEXP.
The first deck of the set may be given in the general form //top/patch1/deck1, which sets the CWD for the entire set to //top/patch1. A set which includes all decks in the CWD is specified by * (equivalent to ?*). Instead of decks you may also specify patches this way. Some examples:

```
Ex.  //top/patch2/deck1   ; DECK1 in PATCH2 in file TOP
     deck1                ; DECK1 in the CWD
     deck1.               ; from DECK1 included to the last deck in
                            the CWD
     .deck2               ; from the first deck to DECK2 included in
                            the CWD
     deck3.deck6          ; from DECK3 to DECK6 included in the CWD
     *                    ; all decks in the CWD or all decks in all
                            patches if the CWD is the top directory
     //top/patch2.patch4  ; from PATCH2 to PATCH4 included in file TOP
     //top1 //top2        ; all decks in all patches in the files TOP1
                            and TOP2
     //                   ; all decks in all patches in all files
```

7.5.4 DIFFERENCES orig [desc]

`ORIG` C "[-F FILNAM][-H][-O][-B][-P] Name of the original deck/patch/file | -V vers_num" `D='␣'`

`DESC` C "Name of the descendant deck/patch/file | -VD vers_num2" `D='␣'`

Find the differences between two decks. This command uses the local compare utility to compare two files (Apollo: cmf, Unix: diff, VAX: DIFF). If option -H is set the decks will be compared ignoring the CMZ history records. If option -O is set only the decks available in the descendant file will be compared with the decks in the original file. If option -B is set the decks in the decklist buffer will be compared between the original file and the descendant file. If option -V vers_num is given DIFFERENCES will compare version vers_num with the current highest cycles of all decks. If in addition -VD vers_num2 is given then vers_num2 is compared against vers_num. If option -P is set, on Unix systems the differences are browsed in the pager currently defined via the command HOST_PAGER. If option -F is set the output is written on file FILNAM.

```
Ex. DIFF deck1 deck2            ; compares DECK1 and DECK2
    DIFF deck1 deck2 -F f.diff  ; same as above but output on file F.DIFF
    DIFF -H deck1 deck2         ; compares DECK1 and DECK2 ignoring
                                  CMZ history records
    DIFF patch1 patch2          ; compares PATCH1 and PATCH2
    DIFF //file1 //file2        ; compares two entire files
    DIFF -O //file1 //file2     ; compares only the decks available in
                                  FILE2 with the decks in FILE1.
    DIFF deck                   ; compares DECK;max-1 and DECK;max
    DIFF deck;6                 ; compares DECK;6     and DECK;max
    DIFF *                      ; compares highest cycle, max, of all
                                  decks in the current directory with
                                  cycle max-1.
    DIFF -V 1.19/01 deck1       ; compares the highest cycle of DECK1
                                  with the version 1.19/01 of DECK1.
    DIFF -V 1.19/01 -VD 1.23/03 ; compares version 1.23/03
                                  with 1.19/01
    DIFF -B //file1 //file2     ; compares decks in the decklist buffer
                                  between FILE1 and FILE2
    DIFF -B //file1             ; compares highest cycle, max, of the
                                  decks in the decklist buffer with cycle
                                  max-1 of these decks in FILE1
```

7.5.5 EXEC mname [args]

```
MNAME   C   “Macro name”
ARGS    C   “Macro arguments”  D='␣' Separate
```

Execute the command lines contained in the macro MNAME. Macros may be stored in a deck or in a normal text file. If a deck or file contains several macros, the character '#' is used to select a particular macro inside the deck or file. If MNAME is of the form DECK_NAME#MACRO_NAME, the macro MACRO_NAME in DECK_NAME is executed. If MNAME does not contain the character '#', the first macro found will be executed.
If MNAME is a full CMZ pathname the macro in that deck will be executed. IF MNAME is not a full pathname it will be searched for in the deck MNAME in the CWD. If it cannot be found in the CWD it will be searched for in the CMZ directory $KUMACS. If MNAME cannot be found in the CMZ file, it will be searched in the file MNAME.kumac
A macro consists of a sequence of CMZ commands. The first line in a macro must be the line MACRO MACRO_NAME and the last line of the macro must be RETURN. If the macro contains +IF directives they will be resolved before the macro will be executed.

```
    Ex.  EXEC  install         ; execute the first macro in deck INSTALL
                                 (in the CWD, or in /$KUMACS or in file
                                 INSTALL.kumac)
         EXEC  install apollo  ; idem, but give argument APOLLO to the macro
         EXEC  mdeck#dialog    ; execute macro DIALOG in deck MDECK
```

7.5.6 FIND dname

```
DNAME   C   “[-E][-R][-A][-C] Name of the DECK”  D='␣'
```

Scan all connected files for deck DNAME. If DNAME is preceded by a file name then only that file is scanned. NOTE: the sub-directory name, of course, need not be given, since that is just what we want to find! Wildcarding is permitted (but not in the file name). See for the wildcarding syntax command REGEXP. If option -A is given then the deckname is appended to the decklist buffer (normally the decklist buffer is cleared at the start of FIND). If option -C is given then go to the directory where the found deck resides. If option -E is given then the deck is opened in edit mode. If option -R is given then the deck is browsed with the local browser or pager (see command READ).

```
    Ex.  FIND deck        ; search for DECK in all connected files
                            and print out its path
         FIND cmd?*       ; find all decks with names starting with CMD
         FIND //foo/deck  ; search for DECK in file FOO and print its
                            location
         FIND deck -C     ; search for DECK and go to its directory
         FIND deck -E     ; search for DECK in all connected files
                            and when found open it in the editor
         FIND deck -R     ; idem as above but DECK will be shown in the pager
         FIND deck -A     ; add pathname of DECK to the decklist buffer
```

7.5.7 GREP patter [dklist]

```
PATTER   C   "Text pattern to be matched"  D='␣'
DKLIST   C   "[-I] [-O FNAME] (-B | List of decks to be scanned)"  D='*'
```

Scan DKLIST for the occurrence of the text pattern PATTER. The decks to be scanned can be specified by a DKLIST (see DECKLIST). If option -B is given then all decks in the decklist buffer (see BUFFER) will be scanned. PATTER may consist of a list of (maximum 10) ANDed or ORed patterns (in that case the list must be given between ' '). To search for -I, -AND or -OR give @-I, @-AND or @-OR. A pattern may contain regular expressions. See for the regular expression syntax command REGEXP. If option -I is set ignore upper- and lowercase distinction during comparisons. This, however, can increase the scan time by a factor of two. By default the output is on the terminal, if option -O is given the output is written on file FNAME.

```
    Ex.  GREP string          ; search for "string" in all decks and
                                sub-directories
         GREP STRING          ; idem as above but for "STRING"
         GREP STRING -B       ; idem as above but for all decks in the
                                decklist buffer.
         GREP string -I       ; search for "string" as well as "STRING"
         GREP string -O scan  ; write result of search in file SCAN
         GREP %@*?*@*$        ; search for all lines starting and
                                ending with a "*"
         GREP 'str1 -AND str2' ; search for all lines containing "str1"
                                as well as "str2"
         GREP 'str1 -OR str2'  ; search for all lines containing either
                                "str1" or "str2"
         GREP string //file1 //file2  ; search for "string" in the
                                       connected files FILE1 and FILE2
```

7.5.8 KEEP dklist

```
DKLIST   C   "[-U] Name of the decks to be kept"  D='␣'
```

Decks in DKLIST will not be purged in PURGE operations. These decks can still be deleted with the command DELETE. When option -U is set the keep lock of the decks in DKLIST will be removed, unless they are associated with a version. Wildcarding is permitted.

```
    Ex.  KEEP deck1          ; keep highest cycle of DECK1
         KEEP deck1.deckN    ; keep DECK1 to DECKN included
         KEEP deck1;2        ; keep cycle 2 of DECK1
         KEEP -U deck3       ; un-keep the highest cycle of DECK3
```

7.5.9 LIBRARY [fname]

```
FNAME   C   "[-D] File with modules to be put in the object library"  D='␣'
```

Add, replace or delete the object modules in the current object library
with the modules in FNAME. If option -D is set the object module FNAME is deleted from the default object library. Uses the local librarian. If FNAME is omitted then the compiled version of the default extract file is used, i.e. filename.[bin,o,obj]. The default object library may be changed by the command SET ... -L, the default extract file by SET ... -F.

```
    Ex.  LIBRARY foo   ; put modules in FOO.[BIN,O,OBJ] in the
                         default library
         LIBRARY       ; put modules in the default extract file in
                         the default library
         LIBRARY -D foo ; delete object module FOO from the default library
```

7.5.10 PILOT [pilpat]

`PILPAT` C "[-O] Name of patch containing +USE directives" `D='?'`

Store all +USE directives in directory PILPAT in memory. All control options specified by +USE directives in directory PILPAT and the patches referred to by PILPAT are stored in memory. Also all sequence definitions (+KEEP directives) in the selected patches are stored in the sequence bank, if the option -O is set the existing sequence definitions in memory are overwritten. Single decks can also be added to the option list. If no argument is given all set pilot options are listed. Use argument . to clear all set pilot options from memory. The options selected by PILOT are used by CXTRACT, CCOMPILE, CLIB and CTOT with the -P option (see HELP CXTRACT, CCOMPILE, CLIB or CTOT). Use command SEQUENCE to see which sequences are stored in the sequence bank. ATTENTION: PILOT scans all connected files sequentially in the order in which they were connected. Therefore, the PILOT patch should precede all patches it refers to. Command PILOT is mainly added for PATCHY backward compatibility.

```
    Ex.  PILOT            ; show all options set
         PILOT .          ; clear all options from memory
         PILOT *apollo    ; scan patch *APOLLO for +USE directives
                            and all patches referred to via the
                            +USE directives
         PILOT pat1/dek3  ; add DEK3 in PAT1 to the PILOT option list
         PILOT -patch2    ; remove PATCH2 from the PILOT option list
         PILOT -pat1/dek3 ; remove PAT1/DEK3 from the PILOT option list
```

7.5.11 REGEXP

This is a summary of the regular expression syntax used for pattern matching in CMZ (e.g., by GREP). A text pattern is a concatenation of the following elements:

```
     c        literal character c
     ?        any character except newline
     %        beginning of line
     $        end of line (null string before newline)
     [...]    character class (any one of these characters (e.g. [a-z])
     [^...]   negated character class (all but these characters)
     *        closure (zero or more occurrences of previous pattern)
     @c       escaped character (e.g. @%, @[, @*)
```

Special meaning of characters in a text pattern is lost when escaped, inside [...] (except @]), or for:

```
     %        not at beginning
     $        not at end
     *        at beginning
```

A character class consists of zero or more of the following elements, surrounded by [and]:

```
     c        literal character c, including [
     c1-c2    range of characters (digits, lower or upper case letters)
     ^        negated character class if at beginning
     @c       escaped character (e.g. @^, @-, @@, @])
```

Special meaning of characters in a character class is lost when escaped, or for:

```
     ^        not at beginning
     -        at beginning or end
```

An escape sequence consists of the character @ followed by a single character:

```
     @n       newline
     @t       TAB
     @c       c (including @@)
```

When using wildcards in directory or deck names (in commands like DIR, COPY, DELETE, FIND, etc.) the regular expressions are assumed to be preceded by "%" (the beginning-of-line character) and followed by a "$" (the end-of-line character). Also, if the wildcard is a single "*" then it is assumed to be preceded by a "?" (any character). Were it not for these deviations, the wildcard name "?" would match any name, instead of only 1-character names and the wildcard name "*" would be an illegal pattern.

7.5.12 TABS [optlist]

OPTLIST C "[ISTOPS] [-F] [-C] [-OFF] " D='?'

Expand TABs to ISTOPS spaces. If option -F is set any TAB characters will be replaced by spaces using the VMS/Fortran tabulation rules (7,10,13,16,...). If option -C is set any TAB characters will be replaced by 8 spaces. If option -OFF is set then TAB characters will not be killed. Option -OFF is the default.

7.5.13 TOUCH dklist

DKLIST C "[-T TIME] -B | List of decks to be touched" D='␣'

Update the time of the decks in DKLIST (see DECKLIST) to the current time. If option -T set, the time of the decks are set to TIME.

```
    Ex. TOUCH deck1 deck2           ; update the date and time of DECK1 and
                                      DECK2 to the current date and time.
        TOUCH -T 15/12/95.14.12.35 * ; update the date and time of all decks
                                      in the current CMZ working directory
                                      to the specified date and time.
```

7.6 FORTRAN77 commands

7.6.1 FTOC fname [dklist]

`FNAME` C "[-C[C]][-A][-S] Name of the FORTRAN file" `D='␣'`
`DKLIST` C "List of routines" `D='*'`
Import a FORTRAN file into a CMZ file. Each subroutine or function becomes a DECK in the current directory. Using the optional parameter DKLIST it is possible to read only a subset of FNAME (see DECKLIST). If option -C is set, comments in columns 73-80 are removed from statements which are not comment lines. If option -CC is set, comments in columns 73-80 are removed from all statements including comment lines. If option -A is set, the AUTHOR field of the history records will be filled in with the CMZ user name (see AUTHOR). If option -S is set then CMZ will search for matches between the sequence definitions currently stored in memory and identical lines in the FORTRAN file. When a match is found, the corresponding lines will be replaced by a +SEQ,NAME line, where NAME is the name of the sequence in memory.

```
Ex.  FTOC fname               ; read all routines from file FNAME
     FTOC fname  routine1     ; read only routine ROUTINE1
     FTOC fname  routine3.    ; read all routines from ROUTINE3
     FTOC fname .routine5     ; read all routines to ROUTINE5
     FTOC fname rout3.rout5   ; read from routine ROUT3 to ROUT5 included
     FTOC -C fname rout3.rout5 ; idem as above but comments in columns
                                 73-80 are removed.
     FTOC ?*.f                ; on UNIX and Apollo systems, wilcards are
                                allowed as filenames.
```

7.6.2 FORTRAN [fname]

`FNAME` C "Name of the file to be compiled" `D='␣'`
Compile FNAME previously generated by the command CXTRACT. Uses the local FORTRAN compiler. If no filename is specified, the default CMFOR.[F,FTN,FOR,FORTRAN] file will be compiled. The default extract file may be changed by the command: SET ... -F. In the C language mode the C compiler will be invoked (for how to change to C language mode see HELP SET).

```
Ex.  FORTRAN          ; compile default FORTRAN file
     FORTRAN test     ; compile TEST.[F,FTN,FOR,FORTRAN]
```

7.6.3 INDENT [dklist]

`DKLIST` C "[OFFSET][-F] (-B | List of decks to be indented)"
Indent one or more decks containing FORTRAN code. The decks to be indented can be specified by a DKLIST (see DECKLIST) or by option -B. If option -B is given then all decks in the decklist buffer (see BUFFER) will be indented. The option -F forces the indentation. The offset for indentation is by default 3 columns. This may be changed by giving the number of columns as the first deck in DKLIST. The new value will remain valid until it is redefined.

```
Ex.  INDENT deck1     ; indent DECK1
     INDENT 2 *       ; default indentation changed to 2 columns
                        plus indentation of all decks in patch
     INDENT 3         ; default for CMZ (may be in CMZLOGON)
     INDENT           ; show current default indentation
```

7.6.4 LABEL dklist

`DKLIST` C "[-I] (-B | List of decks to be labelled)" `D='␣'`
Automatically change label numbers in one or more decks containing FORTRAN source. The decks to be re-labelled can be specified by a DKLIST (see DECKLIST) or by option -B. If option -B is given then all decks in the decklist buffer (see BUFFER) will be re-labelled. By default new labels start at 10 and are incremented by steps of 10. FORMAT statements start at 10000 and are incremented by steps of 100. END is labelled 999. These values can be changed by the command STEPS. If option -I is set then the decks are re-labelled and indented.

```
Ex.  LABEL deck1       ; re-label DECK1
     LABEL *           ; re-label all decks in patch
     LABEL -I deck1    ; re-label and indent DECK1
```

7.6.5 ROUTINES [fname]

`FNAME` C "File containing user routine definitions" `D='?'`
Load definitions of user routines and functions. These definitions will be used by UNDEFINED to check for undefined input parameters and to check if the routine is called with the correct number of arguments. The default file extension of the definition files is .CMR and need not be specified. If no arguments are given then a list of all currently stored user routines will be shown. If the argument is a . then all user routine definitions will be cleared from memory. The definition file should contain one definition per line. Lines beginning with a * are comment lines. A definition consists of a routine name followed by the argument list. Every argument is represented either by an I, for input parameters, an O, for output parameters or a K, for in- output parameters. If the definition describes a function then the argument list should be followed by a F. If there are no arguments, just omit the argument list. Some examples:

```
HARRAY     IIO
RZCDIR     KI
TSHOW
LENOCC     I     F
ZAP              F
```

Some examples:

```
Ex.  ROUTINES                       ; show known user routines
     ROUTINES .                     ; clear all user routines from memory
     ROUTINES def_file1 def_file2   ; read definition files DEF_FILE1.CMR
                                      and DEF_FILE2.CMR
```

7.6.6 STEPS [ilone ilstep ifone ifstep ilend]

`ILONE`	I	"First statement number for LABEL"	`D=10`
`ILSTEP`	I	"Increment step for labels"	`D=10`
`IFONE`	I	"First format number for LABEL"	`D=10000`
`IFSTEP`	I	"Increment step for FORMAT"	`D=100`
`ILEND`	I	"Label for END"	`D=999`

Change default parameters to be used by the command LABEL. When no arguments are given the default settings will be shown.

```
Ex.  STEPS 5 5         ; change ILONE and ILSTEP
     STEPS             ; show current settings
```

7.6.7 TREE [dklist]

`DKLIST` C "[LEVEL] [-R][-F][-L][-O FILNAM] List of decks" `D='?'`

Draw the calling tree of a routine. The tree consists of lines containing a line number, a calling level number, the routine name and possibly a reference to a previous entry (by line number) of the same routine in the calling tree. If no argument is given then a list of all decks referenced in the TREE bank will be given. Use option . to clear the bank. The number of levels printed can be set with LEVEL from 1 up to 25. By default only the subroutines are processed. If option -F is set the functions are processed as well. By default the output is on the terminal, if option -O is given the output is written on file FILNAM. A tree is only printed when a single deck is given as argument to TREE. If option -R is set then TREE gives a warning when a routine is recursive. If option -L is given only decks belonging to the connected files appear in the tree.

```
Ex. TREE deck            ; draw calling tree of routine DECK
    TREE -F deck         ; same as above but functions are also processed
    TREE -O f.tree deck  ; same as above but output on file F.TREE
    TREE -O f.tree *     ; same as above for all decks in the current
                           working directory or file
    TREE -L -O f.tree deck; same as above but only decks in connected
                           files appear in the tree of deck DECK
    TREE *               ; fill the TREE bank with all decks in the
                           current working directory or file (no printing
                           of tree)
    TREE -R *            ; same as above but all recursive routines are
                           listed
    TREE deck1.deck2     ; fill the TREE bank with DECK1 to DECK2 (no
                           printing of tree)
    TREE //              ; same as above for all decks in all patches in
                           all files
    TREE                 ; list decks referenced in the TREE bank
    TREE .               ; clear the TREE bank
```

7.6.8 UNDEFINED [dklist]

`DKLIST` C "[LEVEL] [-U] [-O FILNAM] (-B | [-K] $USE | [-N] List of decks)"

Find variables which are not defined in a list of decks containing FORTRAN code. The decks to be checked for undefined variables can be specified by a DKLIST (see DECKLIST) or by option -B. If option -B is given then all decks in the decklist buffer (see BUFFER) will be checked. UNDEFINED can not detect undefined variables in COMMON blocks. UNDEFINED can also check the consistency of routine calling sequences (number of parameters and if they are input and/or output). About 330 routines from the CERN Program Library are known to CMZ. Definition for user routines can be added using the ROUTINES command. By default the USE bank is taken into account. If option -U is given then the corrections referenced by the USE bank are ignored. If option -N is given the deck is taken from the USE bank. When -K $USE is given as decklist then process all the decks referenced in the USE bank and all decks from the connected files containingsequences referenced in the USE bank. When $USE is given only decks referenced in the USE bank are processed. With argument LEVEL the level of information printed (1 to 5) is given. The new value will remain valid until it is redefined. By default LEVEL is 1 (minimum output).

```
    LEVEL=1   ; (default) minimum output.
    LEVEL=2   ; UNDEF prints variables, commons and sequences which
                are not used in the current routine.
    LEVEL=3   ; UNDEF prints global variables.
    LEVEL=4   ; UNDEF prints local variables.
    LEVEL=5   ; same as 1+2+3+4
```

By default the output is on the terminal, if option -O is given the output is written on file FILNAM. Some examples:

```
    Ex.  UNDEF  deck1     ; check DECK1 for undefined variables
         UNDEF  2 deck1   ; change output level to 2 and check DECK1 for
                            undefined variables
         UNDEF  3         ; set default output level to 3 (may be in
                            CMZLOGON)
         UNDEF            ; show current output level
         UNDEF -B         ; check all decks in the decklist buffer
         UNDEF *          ; all decks in Current Directory or file
         UNDEF -O f.und * ; same as above but output is written on file F.UND
         UNDEF //         ; all decks in all patches in all files
         UNDEF $USE       ; check all decks referenced in the USE bank
         UNDEF -K $USE    ; check all decks referenced in the USE bank
                            and all decks of the connected files containing
                            sequences referenced in the USE bank
```

Chapter 8

KUIP COMMAND PROCESSOR

8.1 Command Processor commands

8.1.1 HELP [item option]

`ITEM`	C	“Command or menu path” `D='␣'`
`OPTION`	C	“View mode” `D='N'`

Possible `OPTION` values are:

`EDIT`	The help text is written to a file and the editor is invoked,
`E`	Same as ’EDIT’.
`NOEDIT`	The help text is output on the terminal output.
`N`	Same as ’NOEDIT’

Give the help of a command. If ITEM is a command its full explanation is given: syntax (as given by the command USAGE), functionality, list of parameters with their attributes (prompt, type, default, range, etc.). If ITEM=’/’ the help for all commands is given.
If HELP is entered without parameters or ITEM is a submenu, the dialogue style is switched to ’AN’, guiding the user in traversing the tree command structure.
’HELP -EDIT’ (or just ’HELP -E’) switches to edit mode: instead of writing the help text to the terminal output, it is written into a temporary file and the pager or editor defined by the command HOST_PAGER is invoked. (On Unix workstations the pager can be defined to display the help text asynchrously in a separated window.)
’HELP -NOEDIT’ (or just ’HELP -N’) switches back to standard mode. The startup value is system dependent.

8.1.2 USAGE item

`ITEM`	C	“Command name”

Give the syntax of a command. If ITEM=’/’ the syntax of all commands is given.

8.1.3 KUIP/EDIT fname

`FNAME`	C	“File name”

Invoke the editor on the file. The command HOST_EDITOR can be used to define the editor.
If FNAME does not contain an extension the default filetype ’.KUMAC’ is supplied. The search path defined by the command DEFAULTS is used to find an already existing file. If the file does not exist it is created with the given name.

8.1.4 LAST [n fname]

`N` I "N last commands to be saved" `D=-99 R=-99:`
`FNAME` C "File name" `D='␣'`

Perform various operations with the history file.
If FNAME is not specified, the current history file is assumed by default (the startup history file name is LAST.KUMAC). To change the history file the command LAST 0 NEW-FNAME must be entered.
If N.EQ.-99 (default case) the default host editor is called to edit the current history file, containing all the commands of the session.
If N.LT.0 the last -N commands are printed on the screen. On MVS this allows to edit and resubmit commands. On workstations this allows to resubmit blocks of commands by mouse-driven cut-and-paste operations.
If N.EQ.0 the history file FNAME is rewound and set as the current one (the command LAST 0 FNAME itself is not recorded).
If N.GT.0 the last N commands of the session are saved in the current history file.
See also the command RECORDING.

8.1.5 MESSAGE [string]

`STRING` C "Message string" `D='␣' Separate`

Write a message string on the terminal. A useful command inside a macro. Several message strings can be given in the same command line, each of them separated by one or more spaces (the usual parameter separator); therefore multiple blanks will be dropped and only one will be kept. If multiple blanks should not be dropped, the string must be surrounded by single quotes.

8.1.6 SHELL [cmd]

`CMD` C "Shell command string" `D='␣'`

Execute a command of the host operating system. The command string is passed to the command processor defined by HOST_SHELL. If CMD=' ' the shell is spawned as interactive subprocess. To return from the shell enter 'RETURN' (the full word, not just ⟨CR⟩) or 'exit' (depending on the operation system).

8.1.7 WAIT [string sec]

`STRING` C "Message string" `D='␣'`
`SEC` R "Number of seconds" `D=0 R=0:`

Make a pause (e.g. inside a macro). Wait a given number of seconds (if SEC.GT.0) or just until ⟨CR⟩ is entered (if SEC.EQ.0). A message string is also written on the terminal before waiting.

8.1.8 FUNCTIONS

```
          *** KUIP System Functions ***
```

The function name (and arguments) is literally replaced, at run-time, by its current value. At present, the following functions are available:

```
    $DATE  .......................  Current date in format DD/MM/YY
    $TIME  .......................  Current time in format HH.MM.SS
    $CPTIME  .....................  CP time elapsed since last call (in sec)
    $RTIME  ......................  Real time elapsed since last call (in sec)
    $SUBSTRING(STRING,IX,NCH)  ...  STRING(IX:IX+NCH-1)
    $UPPER(STRING)  ..............  STRING changed to upper case
    $LOWER(STRING)  ..............  STRING changed to lower case
    $LEN(STRING)  ................  Length of STRING
    $INDEX(STR1,STR2)  ...........  Position of first occurrence of STR2 in STR1
    $WORDS(STRING,SEP)  ..........  Number of words separated by SEP
    $WORD(STRING,K,N,SEP)  .......  Extract N words starting at word K
    $QUOTE(STRING)  ..............  Add quotes around STRING
```

$UNQUOTE(STRING)	Remove quotes around STRING
$EXEC('macro args')	EXITM value of EXEC call
$DEFINED('var_name')	List of defined macro variables
$FORMAT(number,format)	Format a number according to a Fortran format string, e.g. $FORMAT(1.5,F5.2) ==> ' 1.50' $FORMAT(123,I5.5) ==> '00123'
$ARGS	Command line at program invocation
$KEYNUM	Address of latest clicked key in style GP
$KEYVAL	Value of latest clicked key in style GP
$LAST	Latest command line executed
$ANUM	Number of aliases
$ANAM(I)	Name of I-th alias
$AVAL(I)	Value of I-th alias
$OS	Operating system name, e.g. UNIX or VMS
$MACHINE	Hardware or Unix brand, e.g. VAX or HPUX
$PID	Process ID
$IQUEST(I)	Value of IQUEST(I) status vector
$ENV(var)	Value of environment variable
$PUTENV(string)	string has the form 'name=value'. On UNIX systems $PUTENV makes the value of the environment variable 'name' equal to value'. 1 on success or 0 otherwise.
$FEXIST(file)	1 if file exists or 0 otherwise
$SHELL(cmd,N)	N'th line of shell command output (Unix only)
$SHELL(cmd,sep)	Shell output with newlines replaced by sep
$SHELL(cmd)	Same as $SHELL(cmd,' ')
$CWD()	Current working directory
$CMZFILE()	Name of the current CMZ file
$VERSION(fname)	Version number of the file fname
$IVERSION(fname)	Integer value of the version number of the file fname
$IVERS(xx.yy/zz)	Integer value of the version number xx.yy/zz
$CHVERS(xxyyzz)	Converts the integer value xxyyzz into xx.yy/zz
$NDECKS('?')	Number of decks in the CWD
$NDIRS('?')	Number of directories in the CWD
$NFILES('?')	Number of connected files
$DECKNAME(id)	Name of the IDth deck
$DIRNAME(id)	Name of the IDth directory
$FILENAME(id)	Name of the IDth file
$BUFFERS('?')	Number of items in the decklist buffer
$BUFFER(id)	Pathname of the IDth items in the decklist buffer
$COMPILER_D(lname)	Compiler directives for the language lname
$ISINBUFFER(string)	1 if string occurs in one of the items of the buffer list or 0 otherwise
$NSELECT('?')	Number of control options
$SELECT(i)	Value of the Ith option
$ISSELECTED(string)	1 if string is an option selected via the SELECT command or 0 otherwise
$AUTHOR()	Name of the current author

8.1.9 BUGREPORT [chopt]

`CHOPT` C "Options" `D='B'`

Possible `CHOPT` values are:

`B`	Send a bug report
`C`	Send a comment, suggestion, etc.

Email a bug report or comment to the CMZ developers. The local editor is invoked with a template to be filled out. After the template has been edited, version information about the cmz and the operating system is appended. The user is asked for a confirmation before the report is sent. (The command is not yet implemented on all systems)

8.2 ALIAS

Operations with aliases. Aliases are defined to provide shortcut abbreviations for the input line or some part of it. When encountered on an input line an alias is replaced by its string value which can contain further aliases. (Be careful not to define recursive aliases.)

To juxtaposition aliases, a double slash can be used as concatenation sign. Inside quoted strings and for the ALIAS commands themselves the alias substitution is inhibited. Otherwise

```
    ALIAS/CREATE ALPHA BETA
    ALIAS/CREATE ALPHA BETA
```

whould create an recursive alias BETA and

```
    ALIAS/CREATE ALPHA BETA
    ALIAS/CREATE BETA GAMMA
    ALIAS/DELETE ALPHA
```

would delete the alias name BETA instead of ALPHA itself.

8.2.1 ALIAS/CREATE name value [chopt]

`NAME`	C	"Alias name"
`VALUE`	C	"Alias value"
`CHOPT`	C	"Option" `D='A'`

Possible `CHOPT` values are:

`A`	create an Argument alias
`C`	create a Command alias
`N`	No alias expansion of value

Create an alias NAME which should be substituted by VALUE. An alias name is a sequence of letters and digits starting with a letter. The underscores ('_'), the at-sign ('@') and the dollar-sign ('$') count as letters.
There are two types of aliases: Command aliases are recognized only if they occur in the command position, i.e. as the first token on the line. Argument aliases are recognized anywhere on the command line (except inside quoted strings) if they are surrounded by one of the following separators:

```
    blank  /  ,  =  :  .  %  '  (  )
```

If CHOPT='C' then the alias is a command alias, i.e. an alias that will only be translated when it is the first token on a command line. Example:

```
    Alias/Create GG Graph/Struct/Scratch
    Alias/Create FF File1/Name1/Name2
    GG FF/ID
```

is equivalent to

```
    Graph/Struct/Scratch File1/Name1/Name2/ID
```

```
    Alias/Create LS DIR C
```

is equivalent to

```
    DIR
```

only when LS is the first token on a command line. In the following case LS will not be translated

```
    SHELL LS
```

Aliases occuring inside an value are expanded indepedent whether the value is enclosed by quotes. The option -N allows to suppress this implicit alias expansion.

8.2.2 ALIAS/LIST [name]

`NAME`	C	"Alias name wildcard" `D='*'`

List all aliases matching the wildcard (names and values).

8.2.3 ALIAS/DELETE name

`NAME`	C	"Alias name wildcard" `Loop`

Delete the definition of aliases matching the wildcard. NAME='*' deletes all aliases.

8.3 SET_SHOW

Set or show various parameters and options.

8.3.1 APPLICATION path [cmdex]

`PATH` C "Application name" `D='␣'`
`CMDEX` C "Exit command" `D='EXIT'`
Set the application name. This means that all input lines will be concatenated to the string PATH (until the command specified by the parameter CMDEX is executed, which resets the application to the null string). The value of CMDEX may be specified if the default value EXIT has to be changed (i.e. because already used by the application). APPLICATION can also be inserted in a macro: in this case at least 4 characters must be specified (i.e. APPL).

8.3.2 TIMING [option]

`OPTION` C "Option" `D='ON'`
Possible `OPTION` values are:
`ON`

`OFF`

`ALL`

Set ON/OFF/ALL the timing of commands. If ON, the real time and the CPU time for the latest executed command (or macro) are presented. If ALL, the time is shown for each command being executed within a macro. The startup value is OFF.

8.3.3 BREAK [option]

`OPTION` C "Option" `D='ON'`
Possible `OPTION` values are:
`ON`

`OFF`

`TB`

`?`

Set ON/OFF the break handling. If OPTION='?' the current value is shown. The startup value is ON.
Hitting the keyboard interrupt (CTRL/C on VMS or CTRL/Q on the Apollo) under break ON condition, the current command or macro execution will be interrupted and the user will get again the application prompt.
BREAK TB switch ON the traceback of the routines called, with their line numbers, when an error occurs. This allows the detection of the routines which provoked the error.

8.3.4 COLUMNS [ncol]

`NCOL` I "Number of columns for terminal output" `D=80 R=-1:`
Set the maximum number of columns for terminal output. If NCOL=0 the current number of columns is shown. If NCOL=-1 the current number of columns is taken from the environment variable COLUMNS. If COLUMNS is undefined the startup value is 80.

8.3.5 RECORDING [nrec]

`NREC` I "Rate for recording on history file" `D=25 R=0:`
Set the recording rate for the history file. Every NREC commands of the session the current history file is updated. If NREC=0 the history is not kept at all (i.e. the file is not written). See also the command LAST.

8.3.6 HOST_EDITOR [editor top left width height dxpad dypad npads]

`EDITOR`	C	"Host editor command" `D='?'`
`TOP`	I	"Top position of the edit window" `D=20 R=0:`
`LEFT`	I	"Left position of the edit window" `D=20 R=0:`
`WIDTH`	I	"Width of the edit window" `D=0 R=0:`
`HEIGHT`	I	"Height of the edit window" `D=0 R=0:`
`DXPAD`	I	"X offset for help PAD windows" `D=30 R=0:`
`DYPAD`	I	"Y offset for help PAD windows" `D=20 R=0:`
`NPADS`	I	"Maximum number of shifted pads" `D=4 R=1:`

Set the host command to invoke the editor. The EDIT command will invoke this editor. If EDITOR='?' the current host editor command is shown.
On Apollo the special value EDITOR='DM' invoke Display Manager pads. The special values EDITOR='WINDOW' and 'PAD' can be used to specify the window positions (in pixel units). 'WINDOW' defines the parameters for edit pads, while 'PAD' defines the parameters for read-only pads (e.g. used by 'HELP -EDIT').
On VMS the special values EDITOR='EDT' and 'TPU' invoke the callable editors. The startup time is considerably lower compared to spawning the editor as a subprocess. The callable EDT has one disadvantage though: after an error, e.g. trying to edit a file in a non-existing directory, subsequent calls will always fail. The TPU call can be augmented by command line options, e.g.

```
    HOST_EDITOR TPU/DISP=DECW    | DECwindow interface to EVE
```

On Unix a variety of editors are available, e.g.

```
    HOST_EDITOR vi
    HOST_EDITOR 'emacs -geometry 80x48'
```

On Unix workstations it is possible to do asynchronous editing via the KUIP edit server, i.e. to start an editor in a separate window while the application can continue to receive commands. In order to do that the following conditions must be fulfilled:

```
    - The KUIP edit server 'kuesvr' must be found in the search path.
    - The editor command set by HOST_EDITOR must end with an ampersand ('&').
    - The environment variable 'DISPLAY' must be set.
```

The ampersand flags your intention to use the edit server if possible. If the edit server cannot be used the ampersand will be ignored, i.e. even with

```
    HOST_EDITOR 'vi &'
```

the KUIP/EDIT command will block until the editor terminates if either the 'kuesvr' is not available or 'DISPLAY' is undefined. When using the edit server the editor command is expected to create its own window. 'vi' being a frequent choice, the above command is automatically interpreted as

```
    HOST_EDITOR 'xterm -e vi &'
```

The startup value can be defined by the environment variable 'EDITOR'. Otherwise it is set to a system dependent default: 'DM' (Apollo), 'EDT' (VMS), 'XEDIT' (VM/CMS), 'vi' (Unix).

8.3.7 HOST_PAGER [pager]

`PAGER`	C	"Host pager command" `D='?'`

Set the host command to view a file in read-only mode. If OPTION='?' the current host pager command is shown. The 'HELP -EDIT' command will invoke this pager, e.g.

```
    HOST_PAGER more
```

On Unix workstations the pager can be asynchronous by creating a separate window, e.g.

```
    HOST_PAGER 'xterm -e view &'
    HOST_PAGER 'ved &'
```

On Apollo the special value PAGER='DM' defines the use of Display Manager read-only pads. The pad positions can be adjusted by the HOST_EDITOR command.
The startup value can be defined by the environment variables 'KUIPPAGER' or 'PAGER'. If neither of them is defined the value set by the HOST_EDITOR command is used. On VAX/VMS the startup value is 'TYPE/PAGE'.

8.3.8 HOST_SHELL [shell]

`SHELL` C "Host shell command" `D='?'`
Set the default host shell invoked by the KUIP/SHELL command. If OPTION='?' the current host shell is shown. The startup value is taken from the 'SHELL' environment variable.

8.3.9 RECALL_STYLE [option]

`OPTION` C "Command recall and editing style" `D='?'`

Possible `OPTION` values are:

`?`	show current setting
`KSH`	Korn shell : Emacs like command line editing
`KSHO`	Korn shell + Overwrite : like 'KSH' but overwrite instead of insert mode
`DCL`	VAX/VMS DCL : DCL command line editing
`DCLO`	VAX/VMS DCL + Overwrite : like 'DCL' but overwrite instead of insert mode
`NONE`	disable command line editing

Set the command recall and editing style. If OPTION='?' the current style is shown. The startup value is 'DCL' on VAX/VMS, 'NONE' on Cray and Apollo DM pads, and 'KSH' on other systems.

If the terminal emulator returns ANSI escape sequences (hpterm doesn't!) the up/down arrow keys can be used to recall items from the command history list and the left/right arrow keys to move the cursor.

'KSH' style provides the following control keys for editing:

```
^A/^E   : Move cursor to beginning/end of the line.
^F/^B   : Move cursor forward/backward one character.
^D      : Delete the character under the cursor.
^H, DEL : Delete the character to the left of the cursor.
^K      : Kill from the cursor to the end of line.
^L      : Redraw current line.
^O      : Toggle overwrite/insert mode. Text added in overwrite mode
          (including yanks) overwrites existing text, while insert mode
          does not overwrite.
^P/^N   : Move to previous/next item on history list.
^R/^S   : Perform incremental reverse/forward search for string on
          the history list.  Typing normal characters adds to the
          current search string and searches for a match.  Typing
          ^R/^S marks the start of a new search, and moves on to
          the next match.  Typing ^H or DEL deletes the last
          character from the search string, and searches from the
          starting location of the last search.
          Therefore, repeated DELs appear to unwind to the match
          nearest the point at which the last ^R or ^S was typed.
          If DEL is repeated until the search string is empty the
          search location begins from the start of the history
          list. Typing ESC or any other editing character accepts
          the current match and loads it into the buffer,
               terminating the search.
     ^T      : Toggle the characters under and to the left of the cursor.
     ^U      : Kill from the prompt to the end of line.
     ^Y      : Yank previously killed text back at current location.
               Note that this will overwrite or insert, depending on
               the current mode.
     TAB     : By default adds spaces to buffer to get to next TAB stop
               (just after every 8th column).
     LF, CR  : Returns current buffer to the program.
```

'DCL' style provides the following control keys for editing:

```
     BS/^E   : Move cursor to beginning/end of the line.
     ^F/^D   : Move cursor forward/backward one character.
     DEL     : Delete the character to the left of the cursor.
     ^A      : Toggle overwrite/insert mode.
     ^B      : Move to previous item on history list.
     ^U      : Delete from the beginning of the line to the cursor.
     TAB     : Move to next TAB stop.
     LF, CR  : Returns current buffer to the program.
```

8.3.10 DOLLAR [option]

`OPTION` C "Substitution of environment variables" `D='?'`

Possible `OPTION` values are:

`?`	show current setting
`ON`	enable substitution
`OFF`	disable substitution

Set or show the status of environment variable substitution.
This command allows to enable/disable the interpretation of environment variables in command lines. The startup value is 'ON', i.e. "$var" is substituted by the variable value.
Note that the system function "$ENV(var)" allows using environment variables even for 'DOLLAR OFF' .

8.3.11 FILECASE [option]

`OPTION` C "Case conversion for filenames" `D='?'`

Possible `OPTION` values are:

`?`	show current setting
`KEEP`	filenames are kept as entered on the command line
`CONVERT`	filenames are case converted
`RESTORE`	restore previous FILECASE setting

Set or show the case conversion for filenames.
This command has only an effect on Unix systems to select whether filenames are kept as entered on the command line. The startup value is 'CONVERT', i.e. filenames are converted to lowercase.
On other systems filenames are always converted to uppercase.
The 'RESTORE' option set the conversion mode to the value effective before the last FILECASE KEEP/CONVERT command. E.g. the sequence

```
        FILECASE KEEP; EDIT Read.Me; FILECASE RESTORE
```

forces case sensitivity for the EDIT command and restores the previous mode afterwards.

8.3.12 LCDIR [directory]

`DIR*ECTORY` C “Directory name” `D='␣'`
Set or show the local working directory.
The current working directory is set to the given path name or the current directory is shown.
To show the current directory used LCDIR without argument. ’LCDIR ’ switches to the home directory. ’LCDIR .’ switches back to the working directory at the time the program was started.

Chapter 9

THE MACRO PROCESSOR

9.1 Macro commands

9.1.1 MACRO/LIST [mname]

`MNAME` C “Macro name pattern” `D='␣'`

List all macros in the search path defined by MACRO/DEFAULTS. Macros are files with the extension KUMAC. MNAME may be specified to restrict the list to the macros containing such a string in the first part of their name. For example,

```
    MACRO/LIST ABC
```

will list only macros starting with ABC.

9.1.2 MACRO/TRACE [option level]

`OPTION` C “Option” `D='ON'`
`LEVEL` C “Level” `D='␣'`

Possible `OPTION` values are:

`ON`

`OFF`

Possible `LEVEL` values are:

`'␣'`

`TEST`

`WAIT`

`FULL`

`DEBUG`

Set ON/OFF the trace of commands during macro execution. If TRACE='ON' the next command is written on the terminal before being executed. If LEVEL='TEST' the command is only echoed but not executed. If LEVEL='WAIT' the command WAIT is automatically inserted after the execution of each command. The startup values are OPTION='OFF' and LEVEL=' '.

9.1.3 MACRO/DEFAULTS [path option]

PATH	C	“Search path for macro files” D='?'
OPTION	C	“Automatic EXEC” D='?'

Possible OPTION values are:

?	show current setting
Command	search for commands only
C	same as 'Command'
Auto	search for commands before macros
A	same as 'Auto'
AutoReverse	search for macros before commands
AR	same as 'AutoReverse'

Set or show MACRO search attributes.
On Unix and VMS systems PATH defines a comma separated list of directories in which the commands KUIP/EDIT, MACRO/EXEC, and MACRO/LIST search for macro files. For example,

```
MACRO/DEFAULT '.,macro,~/macro'          | Unix
MACRO/DEFAULT '[],[.macro],[macro]'      | VMS
```

defines to search files first in the current directory, then in the subdirectory 'macro' of the current directory, and last the subdirectory 'macro' of the home directory.
On VM/CMS system PATH defines a comma separated list of filemodes. E.g.

```
MACRO/DEFAULT '*'       | search all disks
MACRO/DEFAULT 'A,C'     | search only disks A and C
```

If PATH='?' the currently defined search path is shown. If PATH='.' the search path is undefined, i.e. files are search for in the current directory (A-disk on VM/CMS) only. The startup value is PATH='.'.
The search path is not applied if the file specification already contains an explicit directory path or if it starts with a '-' character (which is stripped off).
OPTION allows to define whether macros can be invoked by their name only without prepending the KUIP/EXEC command:

```
DEFAULT -Command
CMD                     | CMD must be a command
DEFAULT -Auto
CMD                     | if CMD is not a command try EXEC CMD
DEFAULT -AutoReverse
CMD                     | try EXEC CMD first; if not found try command CMD
```

The startup value is 'Command' (also reset by PATH='.').
Important note:
Inside macros the DEFAULT -A (or -AR) logic is disabled, i.e. DEFAULT -C is always assumed.

9.1.4 MACRO/DATA

Application command to store immediate data into a file. Example:

```
Application DATA vec.dat
1  2  3
4  5  6
7  8  9
vec.dat
```

9.2 Macro global variables

Operations on global variables.

9.2.1 GLOBAL/CREATE name [value text]

`NAME`	C	“Variable name”	`Loop`
`VALUE`	C	“Initial value”	`D='␣'`
`TEXT`	C	“Comment text”	`D='␣'`

Create a global variable.
If used inside a macro the variable [name] is declared as global.

9.2.2 GLOBAL/IMPORT name

`NAME`	C	“Variable name”	`Loop`

Import global variables.
If used inside a macro the variables listed are declared as global. The name may contain '*' as a wildcard matching any sequence of characters.

9.2.3 GLOBAL/DELETE name

`NAME`	C	“Variable name”	`Loop`

Delete global variables.
The global variables listed are deleted. The name may contain '*' as a wildcard matching any sequence of characters.

9.2.4 GLOBAL/LIST [name file]

`NAME`	C	“Variable name”	`D='*'`
`FILE`	C	“Output file”	`D='␣'`

List global variables.
If a file name is specified the output is the list of GLOBAL/CREATE commands to define the selected global variables. The default file extension is .kumac.

9.3 Macro syntax

A macro is a set of command lines stored in a file, which can be created and modified with any text editor.

In addition to all available commands the special "macro statements" listed below are valid only inside macros. Note that the statement keywords are fixed. Aliasing such as "ALIAS/CREATE jump GOTO" is not allowed.

9.3.1 MACRO

A .kumac file may contain several macros. An individual macro has the form

```
MACRO macro-name [ parameter-list ]
   statements
RETURN
```

Each statement is either a command line or one of the macro constructs described in this section (MACRO/SYNTAX). For the first macro in the file the MACRO header can be omitted. For the last macro in the file the RETURN trailer may be omitted. Therefore a .kumac file containing only commands (like the LAST.KUMAC) already constitutes a valid macro.

9.3.2 RETURN

Ending a macro definition

```
RETURN [ value ]
```

The RETURN statement flags the end of the macro definition and not the end of macro execution, i.e., the construct

```
IF ... THEN
  RETURN         | error!
ENDIF
```

is illegal. See MACRO/SYNTAX/EXITM.
The value is stored into the variable [@] in the calling macro. If no value is given it defaults to zero.

9.3.3 EXITM

Terminate macro execution and return to calling macro.

```
EXITM [ value ]
```

In order to return from a macro prematurely the EXITM statement must be used. The value is stored into the variable [@] in the calling macro. If no value is given it defaults to zero.

9.3.4 STOPM

Terminate macro execution and return to command line prompt.

```
STOPM
```

The STOPM statement unwinds nested macro calls and returns to the command line prompt.

9.3.5 ENDKUMAC

Ignore rest of KUMAC file.
A logical "end of file" marker. The KUIP parser will not read any part of a .kumac file which appears after the "ENDKUMAC" command.

9.4 Macro expression syntax

KUIP has a built-in parser for different kinds of expressions: arithmetic expressions, boolean expressions, string expressions, and "garbage expressions".

9.4.1 Arithmetic

Explanation of arithmetic expression syntax.
The syntactic elements for building arithmetic expressions are:

```
expr ::=  number
        | vector-name                (for scalar vectors)
        | vector-name(expr)
        | vector-name(expr,expr)
        | vector-name(expr,expr,expr)
        | [variable-name]            (if value is numeric or
                                      the name of a scalar vector)
        | [variable-name](expr...)   (if value is a vector name)
        | alias-name                 (if value is numeric constant)
        | $system-function(...)
        | - expr
        | expr + expr
        | expr - expr
        | expr * expr
        | expr / expr
        | (expr)
        | ABS(expr)
        | INT(expr)
        | MOD(expr,expr)
```

They can be used in the macro statements DO, FOR, and EXITM, in macro variable assignments, as system function arguments where a numeric value is expected, or as the argument to the $EVAL function.
Note that all arithmetic operations are done in floating point, i.e., "5/2" becomes "2.5". If a floating point result appears in a place where an integer is expected, for example as an index, the value is truncated.

9.4.2 Boolean

Explanation of Boolean expression syntax.
Boolean expressions can only be used in the macro statements IF, WHILE, and REPEAT. The possible syntactic elements are shown below.

```
 bool   ::= expr rel-op expr
          | string eq-op string
          | expr eq-op string
          | .NOT. bool
          | bool .AND. bool
          | bool .OR. bool
          | ( bool )
rel-op ::= .LT. | .LE. | .GT. | .GE.
          |  <   |  <=  |  >   |  >=
          | eq-op
eq-op  ::= .EQ. | .NE.
          |  =   | <>
```

9.4.3 String

Explanation of string expression syntax.
String expressions can be used in the macro statements CASE, FOR, and EXITM, in macro variable assignments, as system function arguments where a string value is expected, or as the argument to the $EVAL function. They may be constructed from the syntactic elements shown below.

```
string ::= quoted-string
         | unquoted-string
         | string // string              (concatenation)
         | expr // string                (expr represented as string)
         | [variable-name]
         | alias-name
         | $system-function(...)
```

9.4.4 Garbage

Explanation of "garbage" expression syntax.
Expressions which do not satisfy any of the other syntax rules we want to call "garbage" expressions. For example,

```
s = $OS$MACHINE
```

is not a proper string expression. Unless they appear in a macro statement where specifically only an arithmetic or a boolean expression is allowed, KUIP does not complain about these syntax errors. Instead the following transformations are applied:

```
o  alias substitution
o  macro variable replacement; values containing a
   blank character are implicitly quoted
o  system function calls are replaced one by one with
   their value provided that the argument is a syntactically
   correct expression
o  string concatenation
```

9.5 Macro variables

Macro variables do not have to be declared. They become defined by an assignment statement,

```
name = expression
```

The right-hand side of the assignment can be an arithmetic expression, a string expression, or a garbage expression (see MACRO/SYNTAX/Expressions). The expression is evaluated and the result is stored as a string (even for arithmetic expressions).

A variable value can be used in other expressions or in command lines by enclosing the name in square brackets, [name]. If the name enclosed in brackets is not a macro variable then no substitution takes place.

9.5.1 Numbered

Accessing macro arguments.
The EXEC command can pass arguments to a macro. The arguments are assigned to the numbered variables [1], [2], etc., in the order given in the EXEC command. The name of the macro, including the file specification, is assigned to [0].
A numbered variable cannot be redefined, i.e., an assignment such as "1 = foo" is illegal. See MACRO/SYNTAX/SHIFT.

9.5.2 Special

Predefined special macro variables.
For each macro the following special variables are always defined:

```
[0]     Fully qualified name of the macro.
[#]     Number of macro arguments
[*]     List of all macro arguments, separated by blanks
[@]     EXITM return code of the last macro called by
        the current one.  The value is "0" if the last
        macro did not supply a return code or no macro
        has been called yet.
```

As for numbered variables these names cannot be used on the left-hand side of an assignment. The values or [#] and [*] are updated by the SHIFT statement.

9.5.3 Indirection

Referencing a macro variable indirectly.
Macro variables can be referenced indirectly. If the variable [name] contains the name of another variable the construct

```
[%name]
```

is substituted by that other variable's value. For example, this is another way to traverse the list of macro arguments:

```
DO i=1,[#]
  arg = [%i]
  ...
ENDDO
```

There is only one level of indirection, i.e., the name contained in "name" may not start with another "%".

9.5.4 Global

Declaring a global variable.

```
EXTERN name ...
```

The variable names listed in the EXTERN statement are declared as global variables. If a name has not been defined with the GLOBAL/CREATE command, it is created implicitly and initialized to the empty string. The name list may contain wildcards, for example

```
EXTERN *
```

makes all defined global variables visible.

9.5.5 READ

Reading a variable value from the keyboard.

```
READ name  [ prompt ]
```

Variable values can be queried from the user during macro execution. The READ statement prompts for the variable value. If name is already defined the present value will be proposed as default.

9.5.6 SHIFT

Manipulation numbered variables.
The only possible manipulation of numbered variables is provided by the SHIFT statement which copies [2] into [1], [3] into [2], etc., and discards the value of the last defined numbered variable. For example, the construct

```
WHILE [1] <> ' ' DO
  arg = [1]
  ...
  SHIFT
ENDDO
```

allows to traverse the list of macro arguments.

9.6 Macro flow control

9.6.1 CASE

Select one of many branches.

```
CASE expression IN
(label)  statement  [ statements ]
...
(label)  statement  [ statements ]
ENDCASE
```

The CASE switch evaluates the string expression and compares it one by one against the label lists until the first match is found. If a match is found the statements up to the next label are executed before skipping to the statement following the ENDCASE. None of the statements are executed if there is no match with any label.
Each label is a string constant and the comparison witht the selection expression is case-sensitive. If the same statement sequence should be executed for distinct values a comma-separated list of values can be used.
The "*" character in a label item acts as wildcard matching any string of zero or more characters, i.e., "(*)" constitutes the default label.

9.6.2 GOTO_and_IF_GOTO

Unconditional and conditional branching.

```
GOTO label
```

The simplest form of flow control is provided by the GOTO statement which continues execution at the statement following the target "label:". If the jump leads into the scope of a block statement, for example a DO-loop, the result is undefined.
The target may be given by a variable containing the actual label name.

```
IF expression GOTO label
```

This old-fashioned construct is equivalent to

```
IF expression THEN
   GOTO label
ENDIF
```

9.6.3 IF_THEN

Conditional execution of statement blocks.

```
IF expression THEN
   statements
ELSEIF expression THEN
   statements
...
ELSEIF expression THEN
   statements
ELSE
   statements
ENDIF
```

The general IF construct executes the statements following the first IF/ELSEIF clause for with the boolean expression is true and then continues at the statement following the ENDIF. The ELSEIF clause can be repeated any number of times or can be omitted altogether. If none of the expressions is true, the statements following the optional ELSE clause are executed.

9.6.4 ON_ERROR

Installing an error handler.
Each command returns a status code which should be zero if the operation was successful or non-zero if any kind of error condition occurred. The status code can be tested by $IQUEST(1) system function.

```
ON ERROR GOTO label
```

installs an error handler which tests the status code after each command and branches to the given label when a non-zero value is found. The error handler is local to each macro.

```
ON ERROR EXITM  [ expression ]
```

and

```
ON ERROR STOPM
```

are short-hand notations for a corresponding EXITM or STOPM statement at the targat label.

```
ON ERROR CONTINUE
```

continues execution with the next command independent of the status code. This is the initial setting when entering a macro.

```
OFF ERROR
```

An error handler can be deactivated by this statement.

```
ON ERROR
```

An error handler can be reactivated by this statement.

9.7 Macro construction loops

9.7.1 DO

Loop incrementing a loop counter.

```
DO loop = start_expr, finish_expr  [, step_expr ]
   statements
ENDDO
```

The step size (setp_expr) defaults to "1". The arithmetic expressions involved can be floating point values but care must be taken of rounding errors.
Note that "DO i=1,0" results in zero iterations and that the expressions are evaluated only once.

9.7.2 FOR

Loop over items in an expression list.

```
FOR name IN expr_1 [ expr_2 ... expr_n ]
   statements
ENDFOR
```

In a FOR-loop the number of iterations is determined by the number of items in the blank-separated expression list. The expression list must not be empty. One by one each expression evaluated and assigned to the variable name before the statements are executed.
The expressions can be of any type: arithmetic, string, or garbage expressions, and they do not need to be all of the same type. In general each expression is a single list item even if the result contains blanks.
The variable [*] is treated as a special case being equivalent to the expression list "[1] [2] ... [n]" which allows yet another construct to traverse the macro arguments:

```
FOR arg IN [*]
   ...
ENDFOR
```

9.7.3 REPEAT

Loop until condition becomes true.

```
REPEAT
   statements
UNTIL expression
```

The body of a REPEAT-loop is executed at least once and iterated until the boolean expression evaluates to true.

9.7.4 WHILE

Loop while condition is true.

```
WHILE expression DO
   statements
ENDWHILE
```

The WHILE-loop is iterated while the boolean expression evaluates to true. The loop body is not executed at all if the boolean expression is false already in the beginning.

9.7.5 BREAKL

Terminate a loop.

```
BREAKL [ level ]
```

Allows to terminate a loop prematurely. The BREAKL continues executing after the end clause of a DO, FOR, WHILE, or REPEAT block, where "level" indicates how many nested constructs to terminate. The default value level=1 terminates the innermost loop construct.

9.7.6 NEXTL

Continue with next loop iteration.

```
NEXTL [ level ]
```

Allows to continue with the next loop iteration without executing the rest of the loop body. Execution continues just before the end clause of a DO, FOR, WHILE, or REPEAT block, where "level" indicates how many nested blocks to skip. The default value level=1 skips to the end of the innermost loop construct.

Appendix A

From PATCHY to CMZ

A.1 Importing CAR Files

CMZ uses the same source code directives (see section 3.3) as the batch oriented PATCHY source code manager[1]. This makes it trivial to convert an existing PATCHY card image file into a CMZ file A CMZ file may contain source code (FORTRAN, C and others) as well as plain text. The material that can be imported by CMZ can be either in CAR, FORTRAN, C, plain text or other format. CMZ can output the contents of the CMZ file in CAR, pure FORTRAN, pure C or other compilable languages. (i.e. compile-ready, with sequences and conditional material resolved) or plain text format.

A.1.1 How to Import your PATCHY CAR Files into CMZ Library Files

The command `ARC` is used to import a CAR file into the CMZ file.

`ARC` *car_file* `[-C[C]][-A][-S] [`*decklist*`]`

With the *decklist* argument (see section 4.1) you specify the set of patches and/or decks you want to import from *car_file*. Omitting *decklist* results in the import of the whole CAR file. Patches will become directories in the CMZ file with the decks as the elements. In every directory a deck '`00_PATCH`' will be created containing the '`+PATCH`' directive. If a patch has a blank deck it will be stored in a deck named '`BLANKDEK`'. If the CAR file does not contain history records the following default history records will be added to every deck[2] after the '`+DECK`' directive:

```
*CMZ :          01/06/88  18.02.57  by  Louis Daniels
*-- Author :
```

If you set option `-A` in `ARC` then also the author field of the history records will be filled in:

```
*CMZ :          01/06/88  18.02.57  by  Louis Daniels
*-- Author :    Louis Daniels   01/06/88
```

[1] *PATCHY Reference Manual (4.09)*, H. Klein, J. Zoll, CERN PROGRAM LIBRARY.

[2] Except to the '`00_PATCH`' decks which will only get the '`*CMZ :`' record after the '`+PATCH`' directive.

If the CAR file already contains history records (from a previous CMZ run) no new history records will be added. For more information on the history mechanism see section 4.11. If you give the `-C` option `ARC` will strip off anything appearing in the columns 73–80 from all statements that are not comment lines[3]. When you set option `-CC` the comments in columns 73–80 are stripped off from all lines.

When reading in a multiple CAR file, i.e. a CAR file with more than one '`+TITLE`' directive (see section 3.3.2), `ARC` will, by default, ignore the '`+TITLE`' directives and write everything in the same CMZ file. You have to make sure that the CAR file does not contain patches with identical names. When you want to split the multiple CAR file in multiple CMZ files you have to set option `-S`. In that case `ARC` will create, every time it encounters a '`+TITLE`' directive, a new CMZ file. This new CMZ file will get the name specified in the title followed by a sequence number. The sequence number shows the order in which the CMZ files were created.

A.1.2 A Sample Session

Let us consider the following example. Somewhere on your VAX you have the card image files of the GEANT package ('`GEANT.CAR`', '`GEANX.CAR`'). You want to create the two corresponding CMZ library files and in addition you would like to create a new file in which you want to place your user routines for GEANT. Start by invoking CMZ:

```
$ CMZ
CMZ [0] HELP CREATE
CMZ [1] HELP ARC
CMZ [2] HELP MKDIR
CMZ [3] HELP CD
CMZ [4] HELP COPY
CMZ [5] CREATE GEANT
geant [6] ARC GEANT.CAR
geant [7] CREATE GEANX
geanx [8] ARC GEANX.CAR
geanx [9] CRAETE USER
user [10] MKDIR GEXAM1
user [11] CD GEXAM1
user/gexam1 [12] COPY //GEANX/GEXAM1/* .
user/gexam1 [13] DIR
Current Working Directory = //USER/GEXAM1
OO_PATCH;1  BLANKDEK;1  MAIN;1      UFILES;1    UGINIT;1    UHINIT;1
UGEOM;1     GUKINE;1    GUTREV;1    GUSTEP;1    GUOUT;1     UGLAST;1
DATA1;1
Number of DECKS =  13 Number of CYCLES =  13

user/gexam1 [14] SHELL
$
```

You are back to VMS. You can check that three new files have been created in your working directory:

```
GEANT.CMZ   GEANX.CMZ   USER.CMZ
```

As an exercise you want to generate the FORTRAN code (VAX version with GKS) of GEXAM1 which is now in your '`USER.CMZ`' file. So lets resume CMZ:

[3]Comment lines are lines starting with a '`*`', '`C`' or a '`c`'.

```
$ RETURN
user/gexam1 [15] HELP SELECT
user/gexam1 [16] HELP SEQ
user/gexam1 [17] CD //GEANT
geant [18] SELECT VAX GKS
geant [19] SEQ GCDES
geant [20] CD //USER
user [21] SEQ GEXAM1
user [22] CD GEXAM1
user/gexam1 [23] CEXTRACT *
Number of DECKS   written:  9
Number of RECORDS written:  586
user [24] EXIT
```

As a last example, suppose you want to modify the deck '`UGINIT`' and then make a comparison of the new deck with the original deck given in '`//GEANX/GEXAM1`'. Start a new CMZ session:

```
$ CMZ
CMZ [0] HELP FILE
CMZ [1] HELP EDIT
CMZ [2] HELP COR
CMZ [3] FILE GEANX
geanx [4] FILE USER
user [5] CD GEXAM1
user/gexam1 [6] EDIT UGINIT
user/gexam1 [7] COR //GEANX/GEXAM1/UGINIT UGINIT
Patch GEXAM1

Number of lines in correction file =     5
user/gexam1 [8] EXIT
```

The command `COR` will generate the correction directives ('`+REP`', '`+ADD`', '`+ADB`' and '`+DEL`' (see section 5.2.1) needed to convert '`//GEANX/GEXAM1/UGINIT`' to '`UGINIT`'. It will optionally store a new deck, '`CORR1`', containing the changed lines in a new directory called '`$CORR`'.

To get some hands-on experience with CMZ we advise you to make a copy of one of your card image files and try something similar as described above. Doing this will give you a better idea of the possibilities of CMZ.

A.2 The Flow Control Structure Directives

In addition to the conditional directives described in chapter 1.3.3, the PATCHY compatible directive

```
+SELF [,IF=...].
```

is still maintained. It can be used analogously and with similar construct. The `+SELF` directive starts a section of conditional material, which is skipped if the IF-result is 'false' or is accepted if 'true'.

A section of conditional material is terminated by any CMZ directive other then '+SEQ', i.e. sequence calls within conditional material are part of it.

An example showing the use of the '+SELF' directive is shown in Figure A.1.

Machine dependent (or version, e.g. debug version dependent) code can also be inserted in the code itself (including KEEP definitions) using the '+SELF' directive. For example:

```
<general code>                                        <general code>
+SELF, IF=VAX.                                        +IF,VAX.
<VAX dependent code>                                  <VAX dependent code>
+SELF, IF=APOLLO.             is equivalent to        +ELSEIF,APOLLO.
<Apollo dependent code>                               <Apollo dependent code>
+SELF.                                                +ENDIF.
<general code>                                        <general code>
```

Also in this case the machine dependent code that will be used depends on which option was set with the SELECT command.

The logical operations that are normally accomplished as described in chapter 1.3.1, can be performed in the usual PATCHY way, that is, the logical operators OR, AND and NOT are possible in conjunction with the IF parameters, which are written in general as follows:

```
..., IF=p11,p12,...,  IF=p21,p22,..., IF=p31,p32,...
```

The p*ij* are options, possibly preceded by '-' to indicate the logical NOT. The truth-value of a particular name 'p*ij*=*pname*' is 'true' if the '*pname*' compile option is set by the SELECT command and 'false' otherwise; for 'p*ij*=–*pname*' the value is 'true' if the flag '*pname*' is **not** set by SELECT. The elements p*ij* of the IF group '*i*' combine by the logical OR; all IF groups combine by the logical AND. Hence a line with IF parameters is accepted if *each* IF group is 'true'; the IF group '*i*' is 'true' if at least one of its p*ij* is 'true'. A compile option may consist of up to **32** alphanumeric characters.

Associated material belonging to a rejected directive is skipped. For example:

```
+PATCH, AAP, IF=VAX.
<patch material>
```

the material of patch 'AAP' will only be used if the compile option 'VAX' has been set with the SELECT command.

In order to be PATCHY backward compatible, CMZ supports the calling of sequences with

```
+SELF, sname [, s2, ...]  [, T=PASS] [, IF=...]
```
.

This use of '+SELF', however, is not recommended. For more details see the PATCHY manual.

A.3 Differences between CMZ and PATCHY Directives

Most of the CMZ source code directives are fully compatible with the PATCHY directives. Some directives, however, are changed slightly or do not support all PATCHY options. If options were omitted it was mainly

```
+DECK,CREAD.
*CMZ :  1.44/11 29/07/93  11.50.46  by  B. Smith
*-- Author :    B. Smith   30/09/88
      SUBROUTINE CREAD(EDECK)
*
+SEQ,CMZPAR,CMLUN,CMLIST,CREADO.
*
      CHARACTER*(*)      EDECK
      CHARACTER*(LGLINE) KLINE
*
      IF (NDECK.EQ.0) THEN
+SELF,IF=NEWLIB.
         CALL ITOFT(LUNSCR,FILNAM,IERR)
+SELF,IF=IBMMVS.
         FILNAM = 'CMZ.BROWSE.LIST'
+SELF.
         I1 = 1
         IF (EDECK(1:1).EQ.'$') I1 = 2
+SELF,IF=APOLLO,CRAY,UNIX,IF=-MSDOS.
         FILNAM = '/tmp/'//EDECK(I1:LENOCC(EDECK))//'.TMP'
         CALL CUTOL(FILNAM)
+SELF,IF=-APOLLO,IF=-CRAY,IF=-UNIX,IF=MSDOS.
         FILNAM = EDECK(I1:LENOCC(EDECK))//'.TMP'
+SELF.
*
+SELF,IF=-IBM.
         CALL CMOPEN(LUNSCR,FILNAM,' ',IERR)
+SELF,IF=IBM.
         CALL CMOPEN(LUNSCR,FILNAM,'N',IERR)
+SELF.
         IF (IERR.NE.0) RETURN
      ENDIF
*
      CALL WDECK(LUNSCR,EDECK)
*
      END
```

Figure A.1: *Example of the usage of the '*`+SELF`*' directive.*

because of the different way CMZ accesses its data files. While PATCHY always scans the files sequentially, CMZ uses the direct-access method, resulting in very short access times to any piece of information on the data file.

In CMZ patch names may be 16 characters long and deck names up to 32 characters. In PATCHY only the first 8 characters are significant.

CMZ allows to make any PATCHY directive (including the correction directives '+ADB', '+ADD','+REP' and '+DEL') conditional on the status of some options (set by the SELECT command, see section 4.7 and 7.1.2) by adding *IF parameters*.

A.3.1 The '+PATCH', '+DECK', '+KEEP' and '+SEQ' Directives

CMZ does not allow the redefinition of the same DECK or same KEEP within the same PATCH. For example, the following construct would not work with CMZ:

```
+DECK,NAME1,IF=SUN.
       Subroutine Name1
       end
+DECK,NAME1,IF=-SUN.
       Subroutine Name1
         xx=yy
       end
```

The CMZ version of '+KEEP' does not support the 'P=', 'D=' and 'T=' options. These options are skipped when encountered. This means that it is not possible to have patch and/or deck directed sequences. All sequences are global sequences, i.e. they are available to all decks in all connected CMZ files, any time during the CMZ session. In CMZ a sequence definition is terminated by the next CMZ directive other than '+SEQ', '+CDE' or '+SELF'. In PATCHY it is not allowed to have '+SELF' directives in sequence definitions. If PATCHY compatibility is required this feature should not be used.

The CMZ version of '+SEQ' or '+CDE' does not support the option 'T=' (except for 'T=PASS').

A.3.2 The Directive '+DEL'

In PATCHY the CMZ directives to delete a complete deck

```
+DEL, pname, dname, *.
```

or a complete patch

```
+DEL, pname, ****, 0.
```

are not supported. In PATCHY a complete deck can be deleted by '+DEL,*pname*,*dname*.'.

A.3.3 The '+USE' Directive

The '+USE' directive is only supported via the PILOT command for PATCHY backward compatibility reasons. The CMZ version of '+USE' does not support the option 'T=' (except for 'T=INHIBIT'). For more details see the PATCHY manual.

A.4 Creating your correction set

A.4.1 The '`COR`' CMZ command

Use the command `COR` with the option `-H` if the correction set has to be used by PATCHY on a file without CMZ history records. In that case the '`*-- Author :`' card is considered card 0 instead of the '`+DECK`' card.

A.4.2 Note on the '`CLIST`' CMZ command

The command `CLIST` (see section 4.14) with the option `-H` gives a listing without the CMZ history records. To have a listing without history records might be useful in case you want to check the positions where the corrections in a PATCHY style cradle (without history records) will be applied. The line numbers on the directives in such a correction file are off by at least two lines (this are the two default history records that are added when importing a CAR file for the first time into CMZ, see section A.1.1) in case you don't remove the history records.

A.5 The '`PILOT`' PATCH

The command `PILOT` has been added to CMZ for PATCHY backward compatibility. With `PILOT` it is possible to set all control options and load all sequence definitions referenced via the *pilot_patch*. `PILOT` scans the directory *pilot_patch* for '`+USE`' directives. If the '`+USE`' directive has *not* the option '`T=INHIBIT`' set, the patch and/or deck name(s) on the '`+USE`' directive will be added to the pilot option list. Next, `PILOT` will scan all directories, of which the name occurs in the current pilot option list, for more '`+USE`' directives. Also, all sequence definitions found in the selected patches will be loaded into memory. Additional patches and decks (i.e. *pilot_options*) may be added to or removed from the option list by hand. Decks must be specified as '`[-]patch/deck`', while complete patches may be specified either as '`[-]patch`' or as '`[-]patch/*`'.

Only one *pilot_patch* or *pilot_option* may be specified at a time. If no arguments are given, a list of all active pilot options will be given (completely selected patches are listed as '`patch/*`'). To clear all pilot options at once from memory use argument `.` (please note that the sequence definitions stay in memory and should be cleared with the `SEQUENCES` command, see the next section). The pilot options selected by `PILOT` are used by the commands `CXTRACT`, `CCOMPILE` and `CLIB` with the `-P` option.

NOTE

Before calling `PILOT` you might have to set some machine dependent control option(s) to be able to test '`IF=`' parameters on the '`+USE`' directives. Check your CAR file.

`PILOT` scans all connected files sequentially in the order in which they were connected. Therefore, the *pilot_patch* should precede all patches in all files it refers to.

If you use option `-P` all decks in the pilot option list will be used (for creating a pilot option list see section 4.7). Option `-P` may be followed by *decklist* or option `-B`, in which case the pilot options will be used as ordinary control options (like the options set by SELECT). Use option -P only in this way when the patches or decks that need to be recompiled were initially extracted with the -P option (i.e. the 'IF=' parameters may *quote* names which are in the pilot option list).

A.6 The 'PILOT' command

PILPAT C "Name of patch containing +USE directives" D='?'

Store all +USE directives in directory PILPAT in memory. All control options specified by +USE directives in directory PILPAT and the patches referred to by PILPAT are stored in memory. Also all sequence definitions (+KEEP directives) in the selected patches are stored in the sequence bank. Single decks can also be added to the option list. If no argument is given all set pilot options are listed. Use argument . to clear all set pilot options from memory. The options selected by PILOT are used by CXTRACT, CCOMPILE, CLIB and CTOT with the -P option (see HELP CXTRACT, CCOMPILE, CLIB or CTOT). Use command SEQUENCE to see which sequences are stored in the sequence bank. ATTENTION: PILOT scans all connected files sequentially in the order in which they were connected. Therefore, the PILOT patch should precede all patches it refers to. Command PILOT is mainly added for PATCHY backward compatibility.

```
Ex.  PILOT             ; show all options set
     PILOT .           ; clear all options from memory
     PILOT *apollo     ; scan patch *APOLLO for +USE directives
                         and all patches referred to via the
                         +USE directives
     PILOT pat1/dek3   ; add DEK3 in PAT1 to the PILOT option list
     PILOT -patch2     ; remove PATCH2 from the PILOT option list
     PILOT -pat1/dek3  ; remove PAT1/DEK3 from the PILOT option list
```

Appendix B

H1 Software Organization with CMZ

by
A. De Roeck, G. Bernardi, S. Egli, G. Patel, C. Kiesling
H1 Collaboration, DESY, Hamburg.

This note contains a set of rules and recommendations for code management with CMZ in H1. In this note we summarize the basic procedures used to develop and manage code with CMZ in the H1 collaboration.

Sections B.3 and B.4 give the detailed procedures and rules for using CMZ in H1. CMZ has such a degree of intrinsic flexibility that for large software communities the prescription of a number of rules for code management is necessary. The rules are constructed in the spirit of the code manager requirements identified in section B.1.

It is of course likely that some items may be revised as our experience with CMZ evolves, or get updated in order to exploit additional new features in future releases of the CMZ code manager.

B.1 Prologue

A source code manager should:

- make sure that identical versions of the programs are available at all institutes of the collaboration;
- allow to make frequent updates of the programs in a well organized way;
- allow to unambiguously restore earlier versions of a program;
- facilitate the work of the librarian, who is responsible for large programs to which many people contribute code.

CMZ delivers these possibilities. Additionally it provides a pleasant and powerful environment for code development. CMZ is self-documenting, so at any time you can find out how a command works by using the HELP feature.

The CMZ code manager is in terminology strongly coupled to the former CERN code manager PATCHY. We decided to keep the use of the PATCHY directives to a minimum. No pre-knowledge on, or earlier experience with PATCHY is required to use CMZ within H1 (but it helps of course). Users only need to get acquainted with the PATCHY terminology, explained in next section of this document

B.2 CMZ Terminology

In CMZ the code is collected in files. The CMZ card image file (CAR file) correspond to the PATCHY card image files. The following set of CMZ files is proposed for H1:

`'H1UTIL'`	general utility routines for H1
`'H1SIM'`	H1 simulation program (all modules)
`'H1REC'`	H1 reconstruction program
`'H1SIMTXT'`	BOS text banks for the H1SIM program
`'H1RECTXT'`	BOS text banks for the H1REC program

Other large programs (DISPLAY/GENERATORS/PSI...) will be organized in a similar way in the near future. Additionally it is proposed to have also the CERN GEANT code in our management system, since H1 modifications are produced also for these libraries. The original CERN CAR file structure of this program is kept.

In the future we also foresee a CAR file which contains installation jobs for the different H1 programs:

`'H1INSTAL'`	installation jobs for H1 programs

NOTE

CMZ File Names
All official H1 CMZ files must be defined with the same file names on all different sites.

A CMZ session is started by invoking the CMZ program (consult your local responsible) and by attaching one or more files. A subdirectory structure emerges for each CMZ file as shown in Figure 3.1. A CMZ file consists of a number of subdirectories (in PATCHY called PATCHES) which consist of logically related subroutines or functions (DECKS). Special subdirectories will contain MACROs or BOS TEXT banks.

NOTE

Deck Content
A DECK consists of ONE complete routine or function, or ONE complete TEXT bank.

Each routine (DECK) in CMZ has a cycle number. An update of a routine will be saved with the same name in the same directory but the cycle number is incremented by one. Thus all history is kept and the version of the routine with the highest cycle is the one which was updated most recently.

To show the BOS module structure of the code the subdirectories will have mnemonic names, e.g. for the program 'H1SIM' routines belonging to the modules 'GEA', 'DIG' or 'TRIG' will be easily recognized through the subdirectory names of the type 'GEA_CENT', 'GEA_BEMC', 'DIG_BEMC', 'TRIG_MWPC' etc...

The COMMONS etc., to be inserted in the FORTRAN code during expansion, will all be contained in one PATCH for each CMZ-file. This PATCH will be named '*filename*_MACROS', e.g. for 'H1SIM' it becomes 'H1SIM_MACROS'. These COMMONS and other sequences to be inserted are defined in CMZ by so called '+KEEP' cards. An example from the 'H1UTIL' file is given:

```
+KEEP,H1VOL.
      PARAMETER ( NVOLMX = 400 )
      COMMON / H1VOL / NVOL,MNAMES(NVOLMX),ZCENT(NVOLMX)
+KEEP,H1OSMA.
      COMMON /H1OSMA/ NEVRUN,LSIMST,LSIMEN,LIMSEC
     & ,LUNIN,LUNOUT,LUNDAF,IRANS1,IRANS2,IRANFL,KEVENT
      CHARACTER*4 TYPIN,TYPOUT
      COMMON/H1OSM2/TYPIN,TYPOUT
```

These sequences can be inserted in the relevant routines during expansion with the '+SEQ' (or '+CDE') directives of CMZ. The usage of these directives is (almost) identical to the usage within PATCHY. To use the sequences from the example above in the code the user has to insert into his routines the following lines:

```
+SEQ,H1VOL.
+SEQ,H1OSMA.
```

As stated before it is proposed to limit the usage of the features of PATCHY to a minimum in H1, i.e. to make use only of sequences and conditional flags. '+USE' directives are not allowed and the PILOT feature will not be used by H1. An exception to the rule are the CERN GEANT libraries; see below.

A file in CMZ is identified by its name and version number. The version number consists of three parts:

XX.YY/ZZ	e.g. GEANT	3.13/05
	CMZ	1.32/00

In CMZ terminology these different parts are called: the LEVEL (ZZ), the RELEASE (YY) and the VERSION (XX).

A new VERSION is defined by XX.00/00, a new RELEASE is defined by XX.YY/00. Some examples: 2.01/00 is a new RELEASE of VERSION 2.00/00; the next release will be called 2.02/00. 2.02/03 is the third level of the RELEASE 2.02/00, etc.

It is proposed to increment the LEVEL part when small improvements/changes and bug fixes are implemented. The RELEASE part is incremented if significant changes have occurred or if a considerable amount of new code has been included. The VERSION part is incremented if a general major step has been made (e.g. a program where all bank formats have been changed) or a considerable clean-up of code has been carried out. The VERSION number is not expected to change very often.

NOTE

Version Numbers
Only the master librarian is allowed to define version numbers for H1 CMZ files.

When inspecting a CMZ directory you may encounter routines with a name in uppercase, and routines with a name in lower case. The latter indicates that this cycle of the routine belongs to a well defined version number of the program. When editing such a routine, the newly created cycle will have an uppercase name, since it does not belong to a defined version number. The `VERSION` command will change the case of the routine automatically.

B.3 Responsibilities for Code Management

An important issue towards successfully developing and managing code in a large collaboration is the definition of responsibilities for the code. In this Section we define the tasks of the MASTER LIBRARIAN, LOCAL LIBRARIAN and PATCH RESPONSIBLE.

A MASTER LIBRARIAN will be appointed for each CMZ file: e.g. S. Egli for '`H1SIM`'. Only the master librarian is allowed to update the master library.

The MASTER LIBRARIAN has the right to make changes throughout the whole file, if he considers it necessary. However he/she must inform the person responsible for this part of the code on the changes.

One person will be appointed for each subdirectory of a CMZ file; the so called PATCH RESPONSIBLE. This person will in most practical cases be naturally connected with the subdetector/algorithm code contained in that subdirectory. The patch responsible is responsible for the status of this part of the library at all times. Other users should send their changes (after testing!) to the responsible for that part of the code. He/she collects and evaluates the changes coming from the different users and forwards them to the master librarian. In practice, the above scheme will often involve less people than it looks like at first sight!

Each institute of the H1 collaboration has to appoint a LOCAL LIBRARIAN who will be responsible for collecting updates and installing the new versions of the CMZ files in the local computing area. This librarian is the contact person for the master librarians of the different programs

B.4 Code Management with CMZ in H1

The Code Flow

The scheme for code exchange/update organization proposed for H1 is shown in Figure B.1. The different operations are explained below (more details on the commands can be found in the Reference Manual.

Briefly the code flows as follows in the H1 collaboration: the master gets the updates from the various patch responsibles and updates the master library. This new version of the program gets a unique version number: XX.YY.ZZ.

NOTE

New Program Release
Each officially released state of a program must have a unique version number!

The master creates the update files which can bring the status of the libraries at external sites to the same status as the master library.

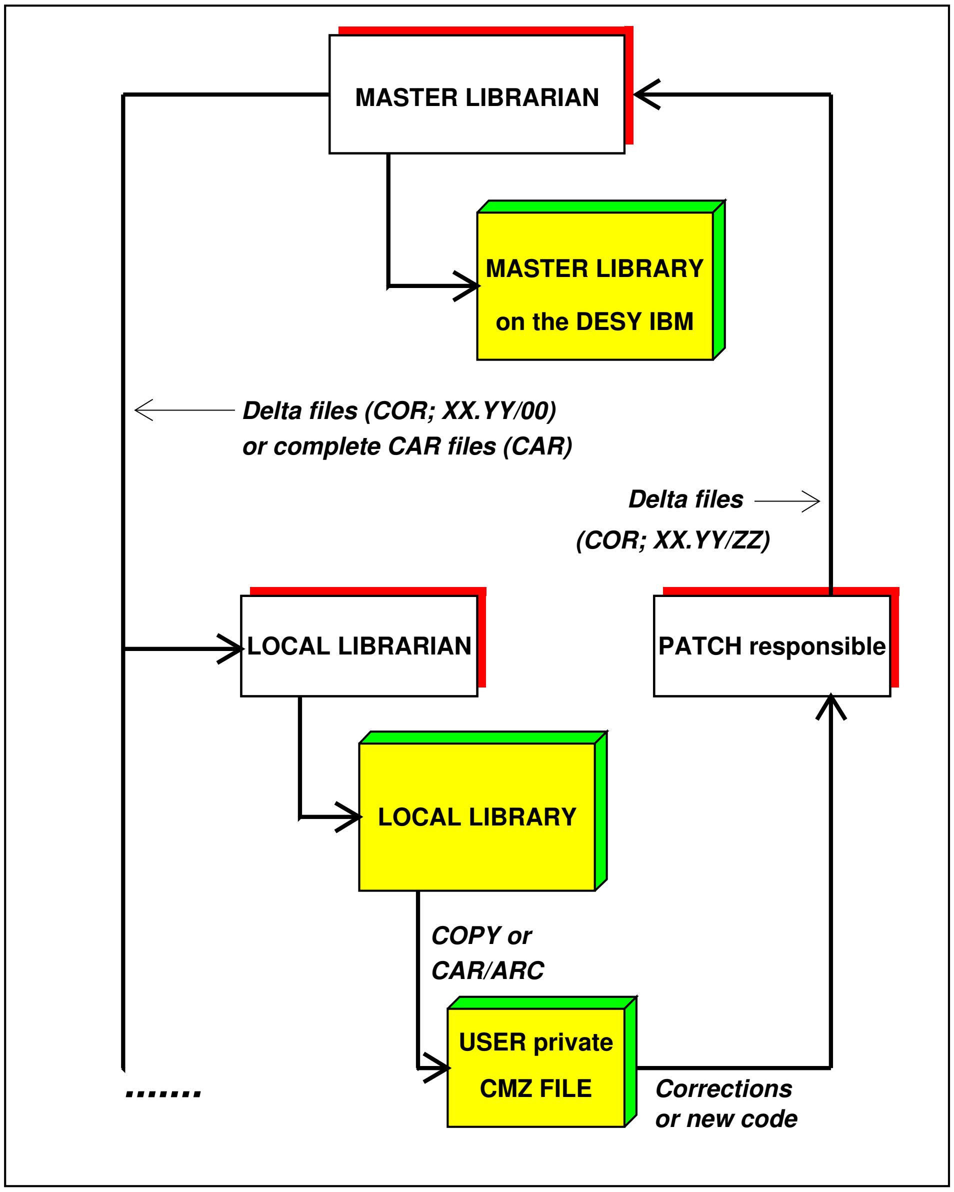


Figure B.1: *Scheme for code exchange and update organization proposed for H1.*

As soon as a new installation is performed, all local librarians and all patch responsibles get an electronic mail message that a new version is released and where the necessary updates or new CMZ files can be found on the DESY IBM. For some CMZ files, like the 'H1SIM' program there will be a copy of the master on the DESY VAX, so that local librarians get the updates the way it is most convenient for them. The presence of a master library on the VAX is no rule and depends on the master librarian. The messages on updates will also appear on the newly installed pinboard H1SOFT on the DESY IBM. All users interested in getting information on the evolution of the general H1 software should attach this pinboard to their checklist (ask any of the H1 software gurus for help if you need it!)

NOTE

Transferring CMZ Files
DELTA-files work on line numbers in DECKS. Therefore, each local librarian of an outside lab should check that his/her favorite technique of file transfer leaves the original line numbering of the CAR file unchanged! Problems occur if e.g. during the transfer blank lines are deleted by the transfer mechanism. In that case the original line numbering is destroyed!!

Local librarians have to get the new code from the IBM (or VAX), or in case of problems, may request to have it sent over the network or by tape. The later will of course introduce delays! In case of a new RELEASE or new LEVEL the updates consist of DELTA-files i.e. files which contain the difference between the new and a well defined older version of the program (see below). In case of a new VERSION a complete new CMZ file will be created in the form of a card image CAR file which has to be transferred. The local librarian installs the new code and informs his user community. Users attach the local library and copy the PATCHES (or single routines) of interest for their work. When they have finished (and tested!) their modifications the new code is sent to the patch responsible. The latter has to make an update for his modified PATCHES and transmits the changes in the form of a DELTA-file to the master.

In the following we will explain explicitly the procedures to be used for updates, installation etc.

B.4.1 Installing New CMZ Files

The local librarian installs the new code with the following commands:

```
> CREATE CMZ_filename
> ARC    CMZ_filename
```

where 'CMZ_filename' can be 'H1SIM', 'H1REC', etc. The CMZ files must have the same names on all sites!! These names are defined in section B.2.

B.4.2 Installing Updates

NOTE

Delta Files from the Master Librarian
The master librarian creates always DELTA-files with respect to the LEVEL 00 of the current RELEASE of the file, i.e. with respect to the version number XX.YY/00.

We call this the starting version.

The local librarian has to make an update of his CMZ-file in the following way: the local librarian puts himself in the top directory of the CMZ-file. This is the directory you are normally in when you attach a CMZ-file; it is the directory which shows you the subdirectory list. Then type the following commands, where file_name is the local name of the received update file:

```
> USE      file_name
> UPDATE   *
```

CMZ checks whether the starting version is available in the local library and makes the updates with respect to the routines which belong to this version. The CMZ file gets automatically updated to the new version number. If the starting version does not exist in the local library CMZ does not make an update and prints an error message.

The librarian should always make updates from the top directory otherwise the version number of the local file will not be updated. The proposed scenario of creating updates with respect to LEVEL 00 of the latest RELEASE implies that the local librarian does not need to have all intermediate level updates to get the program in the correct status: the librarian can step in any moment to update his local library to the official H1 status by including the last available correction file. For example, to get the file to the status 2.01/15 it is not necessary that the local library already contains 2.01/14, 2.01/13... since the received update contains ALL the changes since the last RELEASE 2.01/00, and only this one needs to be present in the library.

Thus one avoids the danger of creating libraries which look the same as far as the last included updates are concerned, but which are actually different because somebody has missed a file of corrections sometime! It also means that if a large number of updates have been collected in the mailbox of the local librarian, before he gets around of including them, he only needs to considers the last file.

B.4.3 Extracting Source Code from CMZ

In general the local librarian not only installs new CMZ files but also provides source code files or load libraries for his local community. The following actions must be taken:

- First, the necessary environment or purpose dependent flags must be selected which are used to steer the expansion of the code. These are flags such as '`DEBUG`xx', to steer certain debug options, or '`VAX`', '`IBM`', etc. to steer the selection of computer dependent code. An example for the relevant flags to be selected should be provided by the master librarians for each program!

  These flags are activated with the command:

  ```
  >SELECT flag     e.g.   SELECT VAX
  ```

- Secondly, the subdirectories containing sequences such as the COMMONS must be loaded with the command:

  ```
  >SEQUENCE subdirectory_name   e.g.  SEQUENCE H1SIM_MACROS
  ```

  For some CERN programs, such as GEANT a PILOT PATCH needs to be activated which contains a collection of selects. The usage of this feature is however not recommended for general software in H1! The command to activate a PILOT PATCH is given by:

  ```
  >PILOT subdirectory_name  e.g. PILOT *GEANT
  ```

To make life easier an installation sequence will be provided by the master librarians for each of the CMZ files. For regular users of the software and for librarians it is advised to include the `SELECT` and `SEQUENCE` commands in their '`CMZLOGON.KUMAC`' file (see section B.5) such that the sequences and selects are loaded automatically during the startup of each CMZ session.

The actual expansion of the code is done with the CMZ commands

```
        > SET output_filename -F     e.g. SET H1SIM -F
        > CX  *
```

The first command sets the file name of the fortran output stream. The second command creates the complete file The code is now ready to be compiled.

It is possible to expand and create a load module or library directly from CMZ with the commands

```
        > SET output_filename -L     e.g. SET H1SIM -L
        > CC  *
   or   > CL  *
```

NOTE

Missing Sequences
During CXTRACT warnings on missing sequences may appear. Do not ignore these messages, since they may indicate unresolved COMMONS, etc.

B.4.4 Code Development in CMZ

Experience on the several machines has shown that CMZ is not only a good source code management system, but also a pleasant environment for program development. Several tools are provided to perform basic checks of the newly developed software, such as compilation, undefined variable check, etc. Other useful features are the `GREP` and `FIND` commands, to locate strings or routines in the CMZ-file. The users are advised to consult the reference manual for more details.

Additionally it allows to work directly with the unexpanded code and to expand it with simple CMZ directives, when needed for testing.

Therefore we advise that all development should be done within a CMZ session. In the following we present the basic rules for this option.

We call the CMZ-file installed by the local librarian the reference CMZ-file. Several users may attach these files during a CMZ session at the same time, however:

NOTE

Local CMZ Files
Users are not allowed to edit directly the reference CMZ files.

This can be easily realized on most machines by putting the CMZ files on the directory of the local librarian, who has to deny write access to the other users.

The user will open his own private CMZ-file, copies complete PATCHES or single routines from the reference CMZ-file into his private file, and then introduces his changes. To avoid confusion for later updates the user should keep the original PATCH names in his private file!

It is recommended to release the reference CMZ file when it is no longer needed by the user!

NOTE

User Updates
Users must check that they start their work on the code from the most recent state of the program (highest VERSION/RELEASE/LEVEL).

Users starting a major development project on a part of the code for which they are not responsible, must inform the patch responsible!

The sequence of CMZ commands for code development may thus look like the following:

```
>CREATE chh1rec      (create personal CMZ file with changes to H1REC)

>MKDIR input         (create PATCH ‘‘input’’ and make it the
                      current directory)

>FILE h1rec -R      (attach original file to system in read mode)
>SEQUENCES cdename  (put CDE definitions into memory; the
                     latter two directives could be put into the
                     CMZLOGON.KUMAC file)

>COPY -K //h1rec/input/*  //chh1rec/input
                    (copy the complete subdirectory input to
                     the local subdirectory //chh1rec/input;
                     -K: don’t remove the version number from the
                     routines)

>CD //chh1rec/input (change to the subdirectory input in the
                     local file chh1rec)

>RELEASE h1rec      (release the file h1rec)

>DIR                (show directory)

>EDIT routine_name  (make the changes in a routine)

>CMAKE              (compile and create object code of the
                     changed routines)
```

It is convenient to use the `CMAKE` command in order to create or update the object libraries for subsequent execution during the test phase. If sequences, such as COMMONS were changed during the update phase, all those routines referencing the modified sequences will be expanded and compiled with these new COMMONS. In case of doubt it is recommended to use the commands `CCOMPILE` and `CLIB` (see manual) instead of `CMAKE`.

B.4.5 Corrections to the Patch Responsible or Master Librarian

Corrections are always sent to the MASTER librarian in the form of DELTA-files by using the `COR` command in CMZ.

NOTE

Delta-File Creation
The DELTA-files are always created by making the difference between the updated routines and the version of the routines from which the code development started which should in practical cases be the most recent state of the program.

This rule allows the patch responsible or master librarian to detect the original state of the code from which the development started.

The procedure to make a DELTA-file is in general:

```
> FILE file1 -R
> COR -O -V xx.yy/zz  //file1   //file2  -F output_name
```

where `-O` directs CMZ to compare only the DECKS available in ‘`file2`’, `-V xx.yy/zz` directs CMZ to compare with a given version in ‘`file1`’, ‘`file1`’ is in this case the local reference library file, ‘`file2`’ is the user file, `-F output_name` is the name of the DELTA-file to be created. It is of course possible to be more specific and, instead of comparing CMZ files, to compare PATCHES or DECKS. In this case use the CMZ filename syntax `//file1/patch1/deck1` as a directive.

In case the reference library state is part of the private library (e.g. for patch responsibles) the command to create a DELTA-file simply is:

```
> COR  -V  xx.yy/zz  PATCH_name -F  output_name
```

where PATCH_name is the name of the subdirectory for which this person is responsible. Instead of a PATCH_name it is also allowed to give an explicit list of subroutines (DECKLIST) instead. The filename is a mnemonic name of the output file which will be sent to the master librarian.

NOTE

Sending Corrections
It is forbidden to send corrections to the master librarian which are made with respect to an older VERSION of the file than the last one. In case updates w.r.t. an older LEVEL of the program, the patch responsible must make sure that no clashes occur with the most recent status of the program!

B.4.6 The Role of the Master Librarian

Here we summarize the procedures for the master librarian:

- inform the patch responsibles on the deadline for sending code to be included in a new update of the program: typically a few days before a new LEVEL increment and a week before a new RELEASE increment. If the program is in a state where one or more new LEVELS are released per week, this rule drops for the LEVEL part.
- collect and check newly received code. Special attention must be payed to starting version number of the created corrections, in case it is not the most recent one. Make changes to the code if it is considered necessary.
- install the new code and establish a new working program
- define a new version number for the file
- update the master library on the DESY IBM
- create a DELTA-file (for new RELEASES and LEVELS) or a new CMZ file (for a new VERSION) The DELTA-files are always made w.r.t. the most recent RELEASE of the program i.e. a version number of the type XX.YY/00!
- document the changes and current status of the program
- inform the users community by e-mail and the pinboard H1SOFT

B.5 Remarks

B.5.1 Not recommended Features of CMZ

It is NOT recommended to:

- to use the CMZ lock features
- to end a CMZ session with QUIT (use EXIT instead)
- to modify with different users the same CMZ file at the same time. Of course several users can attach the same file to extract the relevant routines or use the MACROs
- to overuse the SHELL command, which suspends CMZ and brings in the local command processor.
- to give WRITE access to a large number of users on the locally installed library. It would be advisable that only the local librarian is authorized to modify these libraries

B.5.2 The LOGIN File

General utilities and definitions will be put in the CMZLOGON.KUMAC file. This file is read in at the start of each CMZ session. Typically it would contain:

- the AUTHOR definition
- to SELECT all COMMONS of all H1-CMZ files existing
- to SELECT the machine (VAX, IBM...) dependent sequences

Example on the DESY IBM:

```
AUTHOR 'Albert De Roeck'
SEL IBM TYPE
FILE .HERA01.H1UTIL -R
SEQ H1UTIL_MACROS
REL H1UTIL
FILE .HERA01.H1SIM -R
SEQ H1SIM_MACROS
MESSAGE 'WELCOME TO H1SIM DEVELOPMENT: H1SIM ATTACHED'
SET 'CO $COMPFILE CPRM DC(BCS,GcBANK) ELOAD CPGM FVS' -C
```

B.5.3 CMZ Version Number Identification in FORTRAN Programs

All MAIN programs mastered by CMZ in H1 should contain the sequence '`VERSQQ`', such that the CMZ version number becomes available in the FORTRAN program. The version number should be printed by the program during runtime and (in case of DATA output in BOS banks) stored in a BOS bank for later identification.

B.6 Epilogue

The recent experience with CMZ has shown that it is a useful tool for a code librarian. It should be however clear that this tool is only a support for code management and that is can be only successful if it is used in an intelligent way and with the necessary discipline. It cannot create order out of chaos. Therefore, to avoid major clashes in software it will be still necessary that people obey the rules and communicate with each other if they want to make a valuable contribution to the software.

B.7 Proposal for a Standard Installation-Patch in H1 CMZ-Files

by *U. Berthon*

B.7.1 General ideas

A patch P=$KUMACS should exist in every H1 cmz-file, where one can find, under standard names, everything necessary for installation of the library and , if applicable, executables, documentation, etc. Initialisation of machine-dependent flags, and also configuration of packages by cmz-selections should be done in a central macro-file (cmzsys.kumac). The installation-decks will assume that this macro has been executed automatically (via the new feature of a 'cmzsys.kumac', available in a comfortable way from cmz version 1.46/03 on). The fact that the selection flags for configuration of the packages should be given in cmzsys.kumac leads to the following rule:

Rule on the cmz-selection flags

If selection flags in different packages have the same name, they should also mean the same thing. For example MOTIF-flag already used in LOOK and H1LOOK may be used in a different package only if it means the same.

B.7.2 Directory structure for Unix

We assume that

- the H1-code is installed under one top-directory the name of which is given by the variable H1_ROOT
- each package is situated in a subdirectory of this top-directory having its name : i.e. the name of the cmz-file for a package is $H1_ROOT/¡name-of-the package¿/¡name-of-the-package¿.cmz Attention for h1look-look and h1sim-h1geant this is not always true for the moment!

With additional conventions for doc and fortran files we assume:

cmz-files:	$H1_ROOT/<soft>/<soft>.cmz
doc :	$H1_ROOT/man/cat1/<soft>.1
compilable files:	$H1_ROOT/ftn
libraries:	$H1_LIBS (rather than $H1_ROOT/lib, this allows transparent installation also on file systems with heterogenuous CPU-s)

B.7.3 Structure of the installation-patch

The patch P=$KUMACS contains the decks:

D=cmzlogon	the cmzlogon macro
D=install	a macro for installation of the library, help-files, macros and documentation (see example in figure B.2)
D=makefile	a makefile for UNIX-machines, a script for other machines, containing creation of binaries or anything to be done outside cmz after creation of the library (see example in figure B.3)

NOTE

The decks D=install and D=cmzlogon are obligatory, D=makefile is not.

D=install and D=makefile are separated, since it is preferable not to do creation of library and the binaries in the same step:

- on some machines some part might need to be done outside of cmz
- control in between the 2 steps might be wanted, for example adding the version-number to libraries etc

B.7.4 The cmzsys.kumac file

This file (see example in figure B.4), that can be automatically called at the beginning of a cmz-session with cmz version 1.46/03 and higher, should handle the following:

1) set selection-flags for

```
+DECK,INSTALL,T=DATA.
*CMZ :          14/04/94  10.31.19  by  U.Berthon
*-- Author :
*
    macro install
*
mess Welcome for ... on $MACHINE with $OS
*
SEL  .....                   <- selections generally used for this package
*                               therefore put here and not in cmzsys.kumac
* attach files                  Attention:       no SEL . !!!!!!!
*
SEQ .
if $OS=UNIX then
  FILE $H1_ROOT/h1util/h1util -R
  ...
elseif $OS=VM then
  file h1util.cmz.*
  ...
elseif $OS=VMS then
  file H1$H1UTIL:H1UTIL -R
  ...
elseif $OS=MVS then
  file .HERA01.H1UTIL
  ...
endif
*
* get sequences  *****************************
*
SEQ .
SEQ //H1UTIL/H1UTIL_MACROS
SEQ ...
release H1UTIL ...
*
*  compile  (fortran and C )
*
mess 'Start COMPILATIONS'
CLIB *
MESSAGE ' *** compilation finished ***'
*
mess 'LIBRARY IS CREATED'
*
* Anything else (get main program, doc-files etc)
* This part should never be put before the compilation
* in order not to overwrite the set-statements from cmzsys.kumac !
*
...
...
*
return
```

Figure B.2: *Schema of an install macro*

```
+DECK,makefile  ,T=DATA.
*CMZ :          14/04/94  10.31.19  by  U.Berthon
*-- Author :
#
+SELF,IF=HPUX.
FTN         = f77
FFLAGS      =  -K -O +ppu
LD          = ld
LDFLAGS     = /lib/crt0.o -lcl -lc
+SELF,IF=SGI.
FTN         = f77
FFLAGS      =  -static -G 0
LD          = ld
LDFLAGS=
+SELF,IF=IBMRT.
FTN         = xlf
FFLAGS      = -qextname -qrndsngl -qcharlen=32767
LD          = xlf
LDFLAGS     = -lc -lxlf -lm
+SELF.
#
h1rec:  h1rec.o
        $(LD) -o h1rec  h1rec.o \
         -L$(H1_LIBS)  -lh1rec -llook -lh1util -lfpack -lbos
         -L/cern/pro/lib -lpacklib -lgenlib -lkernlib $(LDFLAGS)

.f.o:
        $(FTN) -c $(FFLAGS) $*.f
```

Figure B.3: *Example of a makefile (UNIX)*

- general selections (for example the site-flag, the machine dependency will be handled by ' sel $OS $MACHINE ' in the installation macro)
- configuration of packages for the specific installation (examples: SEL NET for fpack, SEL MOTIF in case LOOK and H1LOOK should be installed in MOTIF version)

Important remark

It is much more convenient to put the SEL-flags here and to use an install macro directly from the cmz-file without any changes!! This means also that an installation macro should never start with 'SEL .'.

2) set options for FORTRAN and C compilation
(in order to be able to do this in one macro one needs version cmz 1.43 or higher)

B.7.5 How to handle machine dependency

For 'install' and 'cmzlogon' the aim is to have one single deck (and not many different decks more difficult to maintain). Most of the machine dependent stuff is already handled in cmzsys.kumac. The rest, essentially the name of files, can be handled by conditional KUIP macro-statements. There may also be conditional self-code in case parts of the install-deck vary depending on SEL-flags for configuration of the package. This selfcode will be interpreted, before execution of the 'install' macro (for example by cmz -install ...), according to the selection-flags given in cmzsys.kumac.

Makefiles are necessarily different from one machine to the other, at least for UNIX-machines and all the others. This will be handled by conditional selfcode.

```
***********************************************************************
*                          UNIX ONLY!                                 *
* macro for selection of general selection flags and for setting      *
* directories/flags for Fortran and C will be called automatically if *
* variable CMZSYS is set accordingly                                  *
*                                                                     *
                          macro cmzsys
**********************************************************************
*
* general selections
*
sel $OS $MACHINE $H1_SITE
* configuration of packages  LOOK and H1LOOK
sel MOTIF
* DATMAN
sel RANGEC LOOK GKSTXT DEMO
* FPACK
sel NET
*
* generalities for compilations ***************************************
*
set $cmzfile.a -l  ;  set cmexec.exec -xacld2 ; set F C -LAN
*
* set options for FORTRAN and C compilation ****************************
*
set $H1_ROOT/ftn/*.f  -f f77   ; set $H1_ROOT/ftn/*.c  -f C
if $machine=APOLLO then
  opt = '-save -zero -indexl -opt 2 -inline !'
  set 'ftn $compfile -b $compfile.bin '//[opt] -c f77
  set 'cc $compfile -b $compfile.bin' -c C
elseif $machine=SGI    then
  set 'f77 -c -static -G 3 -Nn3000 $compfile' -c f77
  set 'cc -mips2 -c $compfile -cckr -g -I/usr/local/include' -C C
elseif $machine=SUN    then
  set 'f77 -O -N n5000 -N l99 $compfile' -c f77
  set 'cc  -c $compfile -cckr  -I/usr/local/include' -C C
elseif $machine=HPUX   then
  set . -INC C
  set 'f77 -c -K -O +ppu   $compfile' -c f77
  set 'cc -G -g -c $compfile ' -C C
elseif $MACHINE=IBMRT then
  set 'xlf -qextname -qrndsngl -qcharlen=32767 -c $compfile' -C f77
  set 'cc -c $compfile' -C C
endif
*
mess
mess ********** Fortran and C-options are now : ********************
set
return
```

Figure B.4: *Example of cmzsys.kumac (UNIX)*

Index