ИНСТИТУТ ЯДЕРНОЙ ФИЗИКИ
им. Г.И. Будкера СО РАН

V.A. Monitch

ZTREE–DATA ANALYSIS AND GRAPHICS DISPLAY SYSTEM

BudkerINP 94-78

НОВОСИБИРСК

V.A. Monitch

ZTREE - Data Analysis and Graphics Display System

ABSTRACT

This preprint is the ZTREE users guide corresponding version 3.72 of the program which was released in July, 1994.

Chapter 1: Introduction

1.1 What is ZTREE?

The raw and processed data of CMD-2 are stored in ZEBRA sequential files (FZ-files). The ZEBRA package [1] allows the creation of dynamic data structures in FORTRAN-77 programs at execution time and also makes it possible to manipulate those structures, and even to write them to an external storage medium and to recover them intact on another computer system. For example, raw data written on tape by a PDP11/73 online computer are routinely processed on the VAXs or DECstations.

The main idea of ZEBRA is to use a big array (located in a COMMON block), reserved by the user in the main program as a dynamic memory. The simplest ZEBRA data structures (banks) are allocated in this COMMON block. Banks may be referenced by their **ZEBRA address**. The address of a bank is simply it's offset from the beginning of the array. Banks may form more complex data structure using links. Two kinds of links which form data structure are provided: **structural** or **down** links and **next** links which form a linear structure. A bank may be specified by a name (or Hollerith identifier) and a numeric identifier.

CMD-2 raw data files consist of single-bank records. These banks are named **CMD2** and contain raw events in CERN-ON-LINE [7] format. The CMD-2 processed data files begin with a variable number of banks with parameters of reconstruction and subsystem calibration data followed by a variable number of events. Each event is a complex data structure of ZEBRA banks connected by links beginning with the **HEVT** bank. The subsystem header banks under this level are followed by banks of event specific information (i.e. found tracks, their fragments, clusters in CSI calorimeter, etc.).

ZTREE is a program for operating on FZ-files. It has many general commands not concerning CMD-2, and hence may be used not only for CMD-2 needs. However, the majority of ZTREE commands are oriented towards the CMD-2 data file format (especially commands concerning graphical event display).

This Manual consists of the following chapters:

Chapter 2 describes general purpose commands.
Chapter 3 describes utilities.
Chapter 4 describes commands for output into FZ-files.
Chapter 5 describes commands concerning the graphics event display.
Chapter 6 describes a simple interface to ZEBRA.
Chapter 7 describes some KUIP features - vectors, aliases, macros and system functions (built-in and ZTREE specific).
Appendix A describes an interface to COMIS (ZTREE macros).
Appendix B describes ZTREE documentation files.

1.2 Implementation

ZTREE has been written at Budker Institute of Nuclear Physics in FORTRAN 77 and C (Motif interface), and requires standard CERN libraries. It currently runs on two systems: VAX/VMS and DECstation/Ultrix. Two kinds of graphics are supported: GKS and X11.

ZTREE is based on the **KUIP**[2] user interface package developed at CERN in the context of PAW, the Physics Analysis Workstation system. Several large CERN application programs use KUIP: PAW, GEANT, CMZ amongst others. It has some commands of its own, the **system commands** or **KUIP commands** allow the user to get help, to choose among various options in the KUIP/user dialog, to

handle aliases, vectors, macros, etc. Some of its important features are described in chapter 7. Note that this manual uses the same notation as KUIP (see figure 1.3 on page 7 as an example). Moreover, a direct interface to OSF/MOTIF and X-Windows is available inside KUIP. The first version of ZTREE using OSF/Motif based Graphical User Interface (called ZTREE++) has now been released (see figure 1.2).

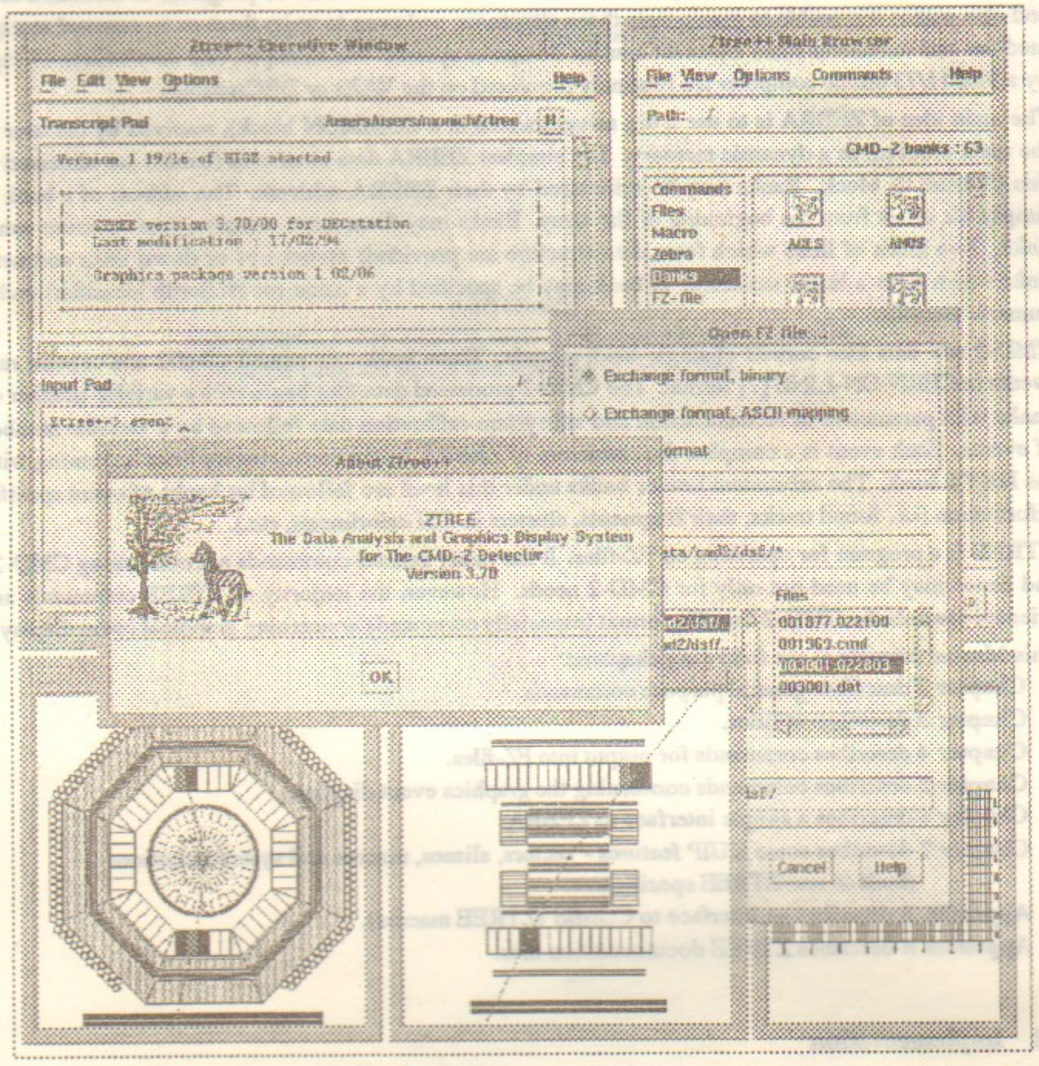


Figure 1.1: OSF/Motif based version of ZTREE

1.3 Starting ZTREE

To invoke ZTREE just type `ztree` at the shell. ZTREE may take three arguments from the command line:


```
$ ztree [file_name] [mode] [recl]
```

The meaning of parameters is the same as for FILE command (see 2.3). An example is:

```
$ ztree /vxcmd/data/cmd2/raw/003161.cmd -x 30720
```

1.3.1 VAX/VMS

When you start ZTREE, it tries to read a logon file pointed to by the logical name 'ZTREE$LOGON'. This logon file may contain commands that are typically typed at the beginning of each ZTREE session. A sample logon file is shown in figure 1.2 on page 5.

You must define (in your 'LOGIN.COM' file) the logical name 'ZTREE$LOGON' to point to the desired default logon file. If your default logon file does not exist or is not defined, ZTREE will inform you before proceeding.

If you run ZTREEX11 (X11-based version of ZTREE) on the VAX, it tries to read a file pointed to by the logical name 'ZTREEX11$LOGON' and then, if this file is not found, a file pointed to by 'ZTREE$LOGON'.

```
*
*  Create a useful alias
*
ALIAS/CREATE NUM 'RUN SYS$LOGIN:EV_NUM.ZTREE'
*
*  Set workstation type (Tektronix)
*
SET/WORKSTATION 101
*
*  Print date and time
*
SHELL TIME
*
*  Type a message
*
MESSAGE 'Hi!'
```

Figure 1.2: Example of a ZTREE logon file.

1.3.2 Unix systems

The ZTREE logon file on Unix systems has the name `.ztreelogon.kumac` and must be in your default directory (i.e. the full name is `$HOME/.ztreelogon.kumac`). ZTREE will inform you if such a file does not exist.

1.4 On-line Help

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

1.5 A Sample Session

Somewhere you have the file processed by CMD2OFF named `002048.extr`. To look through this with ZTREE you simply type

```
$ ztree 002048.extr

*** No ZTREE LOGON file defined

 File has record length 30720 bytes.
        1: Top-level bank <HEVT>   4 data words, 50 down banks.
ZTREE> TREE
```

... the tree of ZEBRA banks is displayed ...

```
ZTREE> DATA FRCS

 Bank <FRCS> contains 6 data words.

 ************************** Data of the bank <FRCS> **************************
    1:                    2          Number of crystals out of clusters
    2:           0.15450E+02         Energy deposition in the crystals, MeV
    3:                  759          Crystal number
    4:           0.16500E+01         Energy deposition in the crystal,MeV
    5:                  808          Crystal number
    6:           0.11250E+02         Energy deposition in the crystal,MeV
 ******************************************************************************
ZTREE> NEXT
        2: Top-level bank <HEVT>   4 data words, 48 down banks.
ZTREE> BACK
        1: Top-level bank <HEVT>   4 data words, 50 down banks.
ZTREE> FZ/OPEN 1 an_event.fz
ZTREE> FZ/WRITE 1
ZTREE> FZ/CLOSE 1
ZTREE> EXIT
```

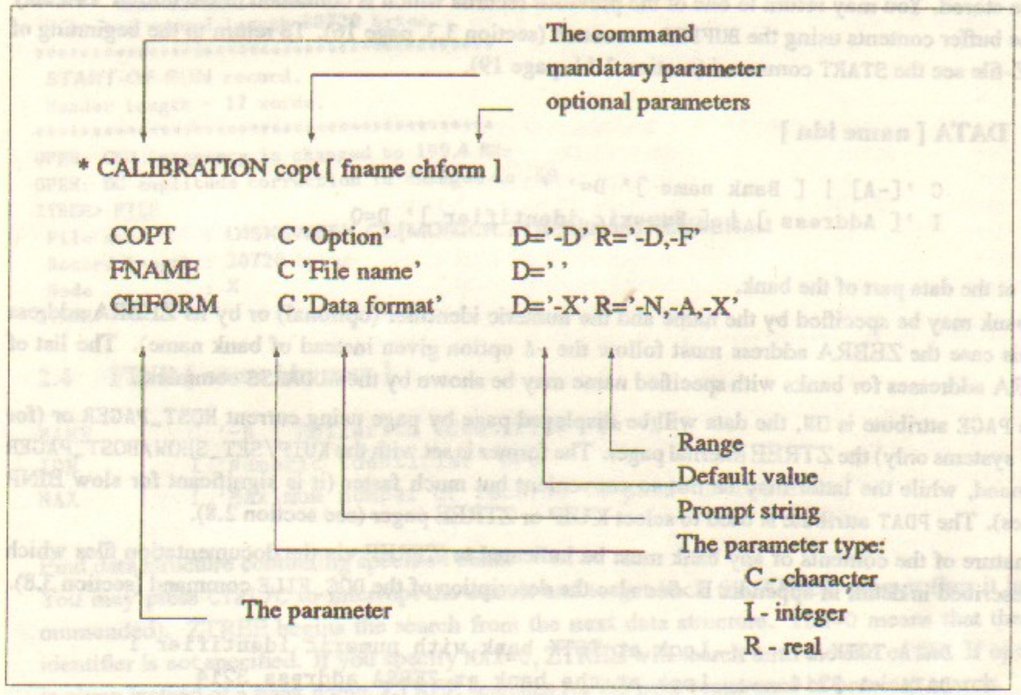



Figure 1.3: Format of notation used by KUIP and this manual.

Chapter 2: General Commands

The general purpose commands described below do not concern CMD-2 and may be used for operation on *any* FZ-files.

2.1 BACKWARDS [nrec]

```
NREC        I 'Number of records' D=1.
```

Go back to the previous record.
ZTREE has a buffer where a number (currently up to 20) of the previously read records read from the file are stored. You may return to one of the previous records which is contained in the buffer. You may see the buffer contents using the BUFFER command (section 3.3, page 16). To return to the beginning of the FZ-file see the START command (section 3.11, page 19).

2.2 DATA [name idn]

```
NAME        C '[-A] | [ Bank name ]' D=' '
IDN         I '[ Address ] | [ Numeric identifier ]' D=0
```

Look at the data part of the bank.
The bank may be specified by the name and the numeric identifier (optional) or by its ZEBRA address (in this case the ZEBRA address must follow the -A option given instead of bank name). The list of ZEBRA addresses for banks with specified name may be shown by the ADDRESS command.
If the PAGE attribute is ON, the data will be displayed page by page using current HOST_PAGER or (for VMS systems only) the ZTREE internal pager. The former is set with the KUIP/SET_SHOW/HOST_PAGER command, while the latter may be not so convenient but much faster (it is significant for slow BINP VAXes). The PDAT attribute is used to select KUIP or ZTREE pager (see section 2.8).
The nature of the contents of any bank must be indicated to ZTREE via the documentation files which are described in detail in appendix B. See also the description of the DOC_FILE command (section 3.8).

```
Ex.     DATA TCTK 1     ;  Look at TCTK bank with numeric identifier 1
        DATA -A 3214    ;  Look at the bank at ZEBRA address 3214
```

2.3 FILE [fname opt lrec]

```
FNAME       C 'File name'      D='?'
OPT         C 'Data format'    D='-X' R='-N,-A,-X'
LREC        I 'Record length' D=0 R=0:
```

Open a ZEBRA sequential access file (FZ-file).
If no parameters are given, ZTREE prints information about the previously open file. The character option OPT specifies the data format:

-X - Exchange format, binary **(default)**
-A - Exchange format, ASCII mapping
-N - Native

Usually the exchange format (-X option) is used. The record length must be given in bytes. If the record length is not specified, (or zero is given) auto-detection will be used for binary files. For ASCII files (-A option) this parameter is ignored and 80 bytes will always be taken.
When opening the file, ZTREE can check automatically significant CMD-2 parameters: CCS [1] frequency and DC amplitude correction. CCS frequency (which equals 249 *MHz* for runs up to 2150, and 199.4 *MHz* for newer runs) is checked when CSU attribute is ON. The DC amplitude correction is checked when DC option CHEK is active. The default for both is ON. The run number is extracted from the **START-OF-RUN** record. An example:

```
ZTREE> SET CSU ON
ZTREE> DC/OPT CHEK
ZTREE> FILE 003037.BHABHA
 File has record length 30720 bytes.
*****************************************
 START-OF-RUN record.
 Header length - 17 words.
*****************************************
OPEN: CSU frequency is changed to 199.4 MHz
OPEN: DC amplitude correction is changed to -50
ZTREE> FILE
 File name     : DISK$WEEK_C1:[MONICH.ZTREE]003037.BHABHA:1
 Record length : 30720 bytes
 Mode          : X
ZTREE>
```

2.4 FIND [name idn max]

```
NAME        C '-N | Hollerith identifier' D='-N'
IDN         I 'Numeric identifier' D=0
MAX         I 'Maximum number of records to pass' D=100 R=0:
```

Find data structure containing specified bank.
You may press CTRL/C to interrupt the search and to get back the ZTREE prompt (but it is not recommended). ZTREE begins the search from the next data structure. IDN=0 means that the numeric identifier is not specified. If you specify MAX=0, ZTREE will search until the end of file. If option '-N' is given instead of a bank name, ZTREE searches for the next occurrence of previous search.

```
Ex.     FIND TCTK          ;  Find the record containing TCTK bank
        FIND -N            ;  Find the next occurrence.
        FIND VRTX 2        ;  Find the record containing VRTX bank
                              with numeric identifier 2
        FIND VRTX 3 500    ;  Find the record containing VRTX bank
                              with numeric identifier 3 within the
                              nearest 500 records
```

[1] CCS is the abbrivation of the Central Control Signal.

2.5 INFORMATION name [idn]

```
NAME      C '[-A] | [ Bank name ]' D=' '
IDN       I '[ Address ] | [ Numeric identifier ]' D=0
```

Display general information about the bank.
The following information is displayed:

```
ZTREE> info TCTK 1
********************************* BANK <TCTK> *********************************
  REFERENCE LINKS:
       -4:          0
       -3:      28198   <GREC>
  STRUCTURAL LINKS:
       -2:          0
       -1:      29618   <FRAG>
 L=            29755
 LQ(L)  =          0              ( NEXT-LINK )
 LQ(L+1)=      29680   <VRTX>      ( UP-LINK )
 LQ(L+2)=      29710              ( ORIGIN-LINK )
 IQ(L-5)=          1              ( NUMERIC BANK IDENTIFIER )
 IQ(L-4)=      <TCTK>               ( HOLLERITH BANK IDENTIFIER )
 IQ(L-3)=          4              ( TOTAL NUMBER OF LINKS )
 IQ(L-2)=          2              ( NUMBER OF STRUCTURAL LINKS )
 IQ(L-1)=         30              ( NUMBER OF DATA WORDS )
 IQ(L)=       262144              ( STATUS WORD )
*******************************************************************************
ZTREE>
```

The bank may be specified by the name and the numeric identifier (optional), or by its ZEBRA address (in this case the ZEBRA address must follow the -A option given instead of bank name). For example, in the previous example we could type `info -a 29755` to get the same information. The list of ZEBRA addresses for banks with specified names may be shown by the ADDRESS command.

```
Ex.       INFO HEVT      ;  Get information about HEVT bank
          INFO -A 4523   ;  Get information about the bank at address 4523
```

2.6 NEXT

Read the next data structure from the input file previously opened with the FILE command.
If the NEXT attribute is ON, a message about data structure read is printed, for example:

```
ZTREE> NEXT
        4: Top-level bank <HEVT> , 4 data words, 68 down banks.
ZTREE>
```

2.7 RUN fname

```
FNAME     C 'File name' D=' '
```

Run a ZTREE macro.
This command actually invokes COMIS interpreter (see [4]). If FNAME does not contain an extension, FNAME.ZTREE is searched for and executed. ZTREE macro is a FORTRAN program which may also contain references to a number of standard routines from the CERN Program Library. Any KUIP or ZTREE command may also be executed from ZTREE macro through KUEXEC routine:


```
CALL KUEXEC('command')
```

In addition to the CERN Program Library routines, some special routines may be used inside ZTREE macros to allow the user to access some ZTREE commands directly with subroutine calls. Such access is much more efficient than through KUEXEC call. See Appendix A for more information.

2.8 SET [attr switch]

```
ATTR      C 'Attribute name' D='?'
SWITCH    C 'Switch' D=' ' R='*,ON,OFF'
```

Set a ZTREE attribute.
An attribute may be in one of two states: ON or OFF. ATTR is a character variable (of length up to 4) identifying the attribute to be set. Abbreviations are possible. Special values are:

- '?' A list of all attributes and their settings is printed.
- '*' All attributes are set to their default values.

SWITCH specifies the new state for the attribute. There is a special value:

- '*' The attribute is set to its default value.

All the possible ZTREE attributes are listed in the table 2.1 (page 12).

```
Ex.     SET ESC OFF    ;  Disable escape sequences
        SET BELL ON    ;  Enable sound signal
        SET WAIT ON    ;  Wait for key pressed after picture is drawn.
        SET            ;  Display the list of all attributes, their
                          default and current settings
```

2.9 SKIP nrec [opt]

```
NREC      I 'Number of ZEBRA records' D=0 R=1:
OPT       C 'Option' D='-M' R='-M,-Q'
```

Skip specified number of ZEBRA records.
Option -Q means quiet work without any output on the terminal. If -M option is given, NEXT messages will be printed (depending on the state of the NEXT attribute).

Name	Default	Explanation
BEEP	ON	If ON, some ZTREE commands (such as FIND) signals about their successful completion by a sound signal.
BOX	ON	If ON, a box is drawn around the viewport.
CASE	ON	If ON, bank names given in a command line are converted to the upper case.
CLIP	ON	If ON, graphics primitives are clipped at the boundary of the viewport.
CLRS	ON	If ON, text screen is cleared before drawing.
CSU	ON	If ON, ZTREE will automatically change CSU frequency depending on run number when opening an input file.
ESC	ON	If OFF, ZTREE endeavors not to use escape sequences for terminal output.
MONO	OFF	If ON, the graphics is black/white. This may be convenient for writing pictures into the PostScript metafile, if you dislike how your printer interprets other colors.
NEXT	ON	If ON, the NEXT command prints a message about data structure read. It's useful to turn it off in macros processing a large amount of events to prevent heavy output.
PAGE	ON	If ON, the DATA command prints data page by page using current HOST_PAGER or (on the VAX only) its internal pager depending on the PDAT attribute.
PDAT	ON	If ON, ZTREE uses its internal pager for data display - this way may be much faster then invoking HOST_PAGER. This works on the VAX only.
WAIT	OFF	If ON, ZTREE waits for key pressed after the picture has been drawn.

Table 2.1: ZTREE attributes and their default values

-N	show Numeric identifiers;
-A	show Addresses of all banks;
-A [address]	the bank with the specified Address will be displayed in a bold face;
-T <name> [idn]	set bank <name> as the Top-level level bank for the tree display: idn is a numeric bank identifier (optional);
-S <address>	set Start address for the tree display;
-O <file_name>	direct Output to the file;
-H	print a Header with an information about current day and time. If you are looking through the CMD2OFF output file, the run and event number also will be printed.

Table 2.2: Options of the TREE command.

2.10 TREE [optlist]

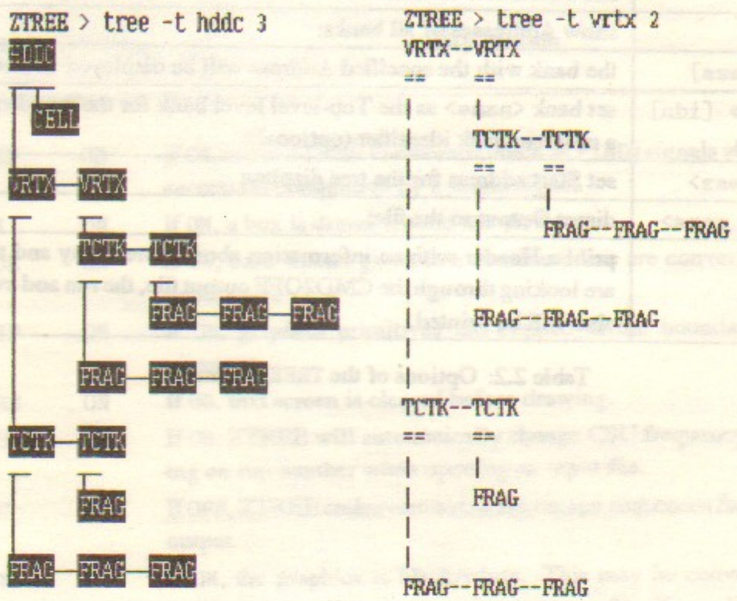
```
OPTLIST    C 'List of options' D=' '
```

Display tree of current ZEBRA structure.
Historically this was the first ZTREE command. [2] **Structural** and **next** links are represented by vertical lines and horizontal lines, respectively. **Reference** links are not displayed at all. By default the tree begins from the top-level bank, but you may display a part of the tree (-T or -S options). The bank tree may be written into a file by using the -O option. The full list of possible options is given in table 2.9 . The tree of banks may be shown in two ways depending on the state of the ESC attribute as shown in figure 2.1 (page 14).

Options -A/-N and -T/-S are mutually exclusive. Other options may be specified together in one command line.

```
Ex.     TREE              ; Display tree of the current data structure
        TREE -N           ; Display tree with numeric identifiers shown
        TREE -A           ; Display tree with ZEBRA addresses shown
        TREE -A 1233      ; Display tree marking bank with address 1233
        TREE -T HDDC -N   ; Display a partial tree beginning from HDDC
                            with numeric identifiers shown
        TREE -H -O T.DAT  ; Write the data structure tree into the file
                            T.DAT together with the header
```

[2] TREE command has distinguished honor of giving the name for ZTREE.



(a) ESC attribute is ON (b) ESC attribute is OFF

Figure 2.1: ZEBRA banks tree representation.

2.11 VERSION

Display the current version of ZTREE and the date of last modification:

```
ZTREE> version
 +-------------------------------------------------+
 |                                                 |
 |   ZTREE version 3.72/00 for DECstation          |
 |   Last modification : 16/07/94                  |
 |                                                 |
 |   Graphics package version 1.03/01              |
 |                                                 |
 +-------------------------------------------------+
ZTREE>
```


Chapter 3: UTILITIES

This chapter describes ZTREE utility commands.

3.1 ADDRESS name [idn]

```
NAME      C 'Bank name' D=' '
IDN       I 'Numeric identifier'
```

List the address of thr NAME bank.
Numeric identifier may be specified. If not specified, ZTREE lists all banks with the name NAME.

3.2 BANK fname name [idn header display]

```
FNAME     C 'File name'
NAME      C '-A       | Bank name'
IDN       I 'Address | Numeric identifier' D=0
HEADER    C 'Header option' D='-H'
DISPLAY   C 'Display option' D='-N' Minus
```

This command writes the bank contents into the file FNAME. ZTREE version, date, and time of file creation may also be stored in this file depending on the HEADER option. Possible HEADER values are:

- `-H` start event data with a header;
- `-NH` do not write a header.

Possible DISPLAY values are:

- `-V` view created file with current HOST_PAGER
- `-N` do not display created file

The bank may be specified by the name and the numeric identifier (optional) or by it's ZEBRA address (in this case ZEBRA address must follow -A option given instead of bank name). The list of ZEBRA addresses for banks with the specified name may be shown by the ADDRESS command.
If the current data structure is an event record (top-level bank is HEVT) and the -NH option is not selected, the file will start with a "header" like this:

```
ZTREE version 3.71/00    Date: 14/07/94   Time: 19.48.38

                  ***********************************
                  *  Run   number:          1840    *
                  *  Event number:             9    *
                  ***********************************
```

The contents of the bank is then printed.

3.3 BUFFER [opt nbuf]

```
OPT       C 'Option' D=' ' R=' ,-S'
NBUF      I 'Buffer size' D=20
```

Show contents of ZEBRA history buffer or set its size.
ZTREE has a buffer where currently a maximum 20 of the last records read from the input file are stored. You may return to one of the previous records from the buffer using the BACKWARDS command. It may be useful to decrease the buffer size in macros in order to reduce calculations when you do not need to go backwards.

```
Ex.     BUFFER          ;  Display the contents of the buffer
        BUFFER -S       ;  Show current buffer size
        BUFFER -S 10    ;  Set buffer size to 10
```

3.4 CALCULATOR [chcalc]

```
CHCALC    C 'Arithmetic expression' D=' '
```

Built-in calculator.
Expression is in FORTRAN where variables X,Y,Z can be used. The value of a variable can be obtained in one of two ways:

```
CALC ?      ; values of all variables will be listed
CALC X      ; the X value will be printed
```

To change the value of a variable, simply type

```
CALC <variable>=<expression>
```

If CHCALC='*', all three variables are set to zero. Typing CALCULATOR without parameters is equal to

```
ZTREE> APPLICATION CALCULATOR EXIT
```

Example of using calculator

```
ZTREE> CALC
 Welcome to the ZTREE calculator.
 Type 'exit' to return to the command processor
CALCULATOR > *
CALCULATOR > X=1.2345
  1.234500
CALCULATOR > Y=X
  1.234500
CALCULATOR > Z=SQRT(SIN(X)**2+COS(Y)**2)
 0.9999999
CALCULATOR > ?
 X =    1.234500
 Y =    1.234500
 Z =    1.000000
CALCULATOR > EXIT
ZTREE>
```


3.5 CLR

Clear the text screen.

3.6 COMIS

Invoke the COMIS FORTRAN interpreter.
COMIS allows the execution of FORTRAN routines without recompiling and relinking. It may communicate with ZTREE commands through vectors. COMIS is invoked by the RUN command, but it has also its own command structure. An example in command mode:

```
ZTREE > COMIS
CS >    do 10 i=1,10
MND>        x=sqrt(i)*10.
MND>        print *,i,x
MND> 10 continue
MND>    END
CS > QUIT
ZTREE>
```

COMIS code may be inserted into a KUIP macro. Example:

```
Vector/Create Y(10) r 1 2 3 4 5 6 7 8 9 10
*
*  In the following COMIS code, the statement "Vector Y" declares
*  to COMIS an existing KUIP vector. KUIP dimension is assumed.
*  The statement "Vector X(10)" creates a new KUIP vector.
*  (Note that SUBROUTINEs must be declared before the MAIN program)
*  (KUIP vectors cannot be created into the MAIN program)
*
APPLication COMIS QUIT
      SUBROUTINE DEMO
      Vector Y
      Vector X(10)
      do 10 i=1,10
         XX=i
         X(i)=Y(i)*sqrt(XX)*10.
 10   CONTINUE
      END
      CALL DEMO
      END
QUIT
Vector/print X        * Print KUIP vector created by COMIS
```

The most usual way to use COMIS, however, is to execute your programs with the RUN command. See section 2.7 and appendix A for more information on the RUN command and using COMIS inside ZTREE.

3.7 DESCRIPTION list

```
LIST     C '[-<option>] (List of the banks)' D=' '
```

Create a text file with the bank documentation.
The file name can be set by the DOC_FILE command (option -D). If the file name contains '*' wildcards, they are replaced with the bank name. The following options may be used:

-M	to scan only the main documentation file;
-L	to scan only the local documentation file;
-T	to direct output only to the terminal (no files are created);
-LATEX	to produce a document in the LaTeX format.

Bank names may contain wildcards '*' and '?'. See the appendix B (page 60) for more examples.

```
Ex.   DOC_FILE *_FORMAT.DOC -D     ;  Set the name of a bank description
      DESCRIPT TCTK                ;  file and write the documentation
                                      for the bank TCTK into the file
                                      TCTK_FORMAT.DOC
      DOC_FILE BANKS.DOC -D        ;  Write information about banks
      DESCRIPT Z*                  ;  beginning from Z into the file
                                      BANKS.DOC
      DOC_FILE *.TEX -D            ;  Write LaTeX descriptions of the banks
      DESCRIPT -LATEX TCTK FRAG    ;  TCTK and FRAG into the files TCTK.TEX
                                      and FRAG.TEX, respectively.
```

3.8 DOC_FILE [fname opt]

```
FNAME    C 'File name' D='?'
OPT      C 'Option' D='-M' R='-L,-M,-D'
```

This command is used to define the name of the ZTREE documentation file (see appendix B for more information about documentation files). If option -L is set then the name of the local documentation file is FNAME. If option -M is set, the name of the main documentation file is FNAME. If parameters are omitted, the list of current settings is displayed. By default the main documentation file is DISK$C1:[MONICH.ZTREE]ZTREE.DOC (see this as an example of a documentation file), the local one is ZTREE.DOC.

Using the option -D you can set the file name for the DESCRIPTION command (by default *.DOC). This file name may contain '*' symbols, which will be replaced with a bank name (see also the section 3.7 and the appendix B).

```
Ex.   DOC_FILE ZTREE.NEW -L  ;  Define ZTREE.NEW as a local ZTREE
                                documentation file
      DOC_FILE               ;  Display current names of ZTREE
                                documentation files
```


3.9 FORGET name

```
NAME     C 'Bank name' D=' '
```

Force ZTREE "to forget" about the format of the bank.
ZTREE remembers the formats of the last 10 banks that were specified in the DATA command. If you look at the bank data after the FORGET command, ZTREE scans documentation files to redefine the format of the forgotten bank. This may be useful when you have changed the bank format in your local documentation file and wish to view the data with a new representation .

3.10 HEADER [hdopt]

```
HDOPT    C 'Option' D=' ' R='-I,'
```

Look at the header of the current record.
Any ZEBRA record read from a FZ-file may have a header. Usually only Start-Of-Run (SOR) and End-Of-Run (EOR) records have one. If the option -I is given, the order of bytes in the header is inverted.

3.11 START

Restart the reading of the file that is currently open.

3.12 STATISTICS

Display some statistics.

3.13 SHOW/LUNS [luns]

```
LUNS     I 'Logical unit numbers' D=1
```

Show logical unit numbers used.
Sometimes you need to choose a logical unit number (*LUN*) not used yet (for example, in FZ/OPEN command). This command prints information about specified *LUN*(s) or the list of all *LUN*s used (or reserved for use).

```
Ex.   SHOW/LUN 10 11   ;  show information about LUNs 10 and 11
      SHOW/LUN         ;  list all LUNs used
```

3.14 SHOW/PAGE [size]

```
SIZE     C 'Page size' D='?'
```

Set or show page size. The page size is used by the ZTREE internal pager for data output (when the PAGE and PDAT attributes are ON). When you wish to change it, you may give the new page size as a parameter. If parameter '*' is given, the page size is set to its default value. When the parameter is omitted, the current page size is printed.

```
Ex.   PAGE       ;  show  current page size
      PAGE *     ;  set default
      PAGE 40    ;  set the page size to 40
```

Chapter 4: FZFILE

FZ output file control.

4.1 FZ/OPEN lun name [opt irec]

```
LUN     I 'Logical unit number'
NAME    C 'File name'
OPT     C 'Data format' D='-X' R='-N,-A,-X'
IREC    I 'Record length' D=30720
```

Open an output FZ file. Character option OPT specifies the data format :

-X -	Exchange format, binary **(default)**
-A -	Exchange format, ASCII mapping
-N -	Native

If you choose the -A option, IREC is ignored. When you have opened the output file, you can write the current data structure into this file with the FZ/WRITE command. Up to 10 output FZ-files may be opened simultaneously. You may use FZ/LIST command to see the list of open FZ-files.

4.1.1 UNIX systems.

In UNIX version of ZTREE this command has a different syntax:

FZ/OPEN name [opt irec]

```
NAME    C 'File name'
OPT     C 'Data format' D='-X' R='-N,-A,-X'
IREC    I 'Record length' D=30720
```

You do not need to specify the logical unit because ZTREE on UNIX uses the C library to access FZ-files. In this case you must use C file descriptor number instead of FORTRAN logical unit number in FZ/WRITE and FZ/CLOSE commands. ZTREE will inform you about this number:

```
ZTREE> fz/open a.dat
OPEN: output file uses LUN 3
ZTREE> fz/list
 +-----+-------------------------------------------+--------+------+
 | Lun |                 File name                 |  Recl  | Mode |
 +-----+-------------------------------------------+--------+------+
 |   3 | /users/users/monich/a.dat                 |  30720 |  X   |
 +-----+-------------------------------------------+--------+------+
ZTREE>
```

If you have have fogotten this number you may type FZ/LIST to be reminded.

4.2 FZ/LIST

List output FZ files. An example:

```
ZTREE> fz/open 1 data.dat
ZTREE> fz/open 2 out.dat
ZTREE> fz/open 3 aout.dat -a
ZTREE> fz/list
 +-----+-------------------------------------------+--------+------+
 | Lun |                 File name                 |  Recl  | Mode |
 +-----+-------------------------------------------+--------+------+
 |   3 | DISK$WEEK:[MONICH.ZTREE]AOUT.DAT;1        |     80 |  A   |
 |   2 | DISK$WEEK:[MONICH.ZTREE]OUT.DAT;1         |  30720 |  X   |
 |   1 | DISK$WEEK:[MONICH.ZTREE]DATA.DAT;1        |  30720 |  X   |
 +-----+-------------------------------------------+--------+------+
ZTREE>
```

4.3 FZ/WRITE lun

```
LUN     I 'Logical unit number'
```

Write the current data structure to the specified FZ-file.
The output file must first be opened with the FZ/OPEN command. You may type FZ/LIST to get the list of currently open files. On the VAX you also may type UNITS.

4.4 FZ/CLOSE luns

```
LUNS    C '*| List of logical units' D=' '
```

Close output FZ file(s). If LUNLIST='*' , all open output FZ-files are closed.

```
Ex.     FZ/CLOSE *      ; Close all output files
        FZ/CLOSE 1 2 3  ; Close logical units 1,2 and 3
```

Chapter 5: CMD2

This chapter describes CMD-2 related commands including an interface to the HIGZ [3] package (CMD-2 picture and event display).

5.1 CALIBRATION [copt fname chform]

```
COPT      C 'Option' D='-F' R='-D,-F'
FNAME     C 'File name' D=' '
CHFORM    C 'Data format' D='-X' R='-N,-A,-X'
```

Read calibrations from the raw data file or set defaults.
If `COPT='-D'`, default calibrations will be restored. If `COPT='-F'`, ZTREE tries to read calibrations from the file `FNAME` or from the file that is currently open if `FNAME` is not specified. Character option `CHFORM` specifies the data format :

- `-X` - Exchange format, binary **(default)**
- `-A` - Exchange format, ASCII mapping
- `-N` - Native

Only calibrations of systems selected with the `SYSTEMS` command are used. Others are ignored.

5.2 DETECTOR

Draw CMD-2 using current settings. Only systems selected with the `SYSTEMS` command are drawn.

5.3 EVENT [opt]

```
OPT       C 'List of options' D=' '
```

Display the current event.
ZTREE supports both data formats used by CMD-2 software:

- raw data format - a sequence of single-bank records containing raw data in CERN-ONLINE format;
- reconstructed events - complex data structure per event with information about found tracks, fragments from which tracks were built, etc.

When you are looking through a raw data file, you can only see just hit wires in DC, hit crystals in CSI, etc. - the only information which can be taken from raw data. If the event is reconstructed reconstructed, ZTREE can also display vertices, tracks, fragments and other information calculated by *CMD2OFF*.

ZTREE does not calculate anything except simple geometry!
It only displays what you give it.

Several options may be given in the command line. The list of possible options is given in table 5.1.

`-T`	only tracks found by CMD2OFF reconstruction program will be displayed (when you are looking through a CMD2OFF output file);
`-R`	the "raw" event will be drawn (taking an information only from raw data);
`-TCTK <tracks>`	(short form is `-NT <tracks>`) specifies tracks that you want to display. Track numbers are simply numeric identifiers of the corresponding `TCTK` banks. By default ALL tracks are displayed;
`-VERT <vert_num>`	(short form is `-NV <vert_num>`) specifies vertices that you want to display. Vertex number is a numeric identifier of the corresponding `VERT` bank;
`-FRAG <frag_num>`	(short form is `-NF <frag_num>`) specifies fragments that you want to display. Fragment number is a numeric identifier of corresponding `FRAG` bank;
`-CELL <cell_num>`	(short form is `-NC <cell_num>`) the same as `-FRAG` option, but an information is taken from the `CELL` banks;
`-HDDC <head_num>`	(short form is `-NH <head_num>`) specifies the number of the DC data structure header (`HDDC` bank) to take the information from.

Table 5.1: Options of the `EVENT` command.

To see all vertices, fragments or cells you have to set DC options `VERT`, `FRAG` or `CELL`, respectively. By default ALL tracks and raw data are displayed. If `BOX` attribute is ON, a box is drawn around the viewport. ZTREE can wait for key pressed after the picture is drawn - when you turn the `WAIT` attribute on (Not in MOTIF mode).

```
Ex:   EVENT                 ; draw whole event (raw data & tracks)
      EVENT -T              ; draw tracks only
      EVENT -T -NT 1        ; draw only track number 1
      EVENT -NT 1 -NF 1 3 ; display track number 1 and fragments number 1,3
```

Only systems selected by the `SYSTEMS` command are drawn. In order to see several projections of the event simultaneously, you must select the desired projections with the command `SECTIONS`. An example of output produced by the `EVENT` command is shown on figure 5.1.

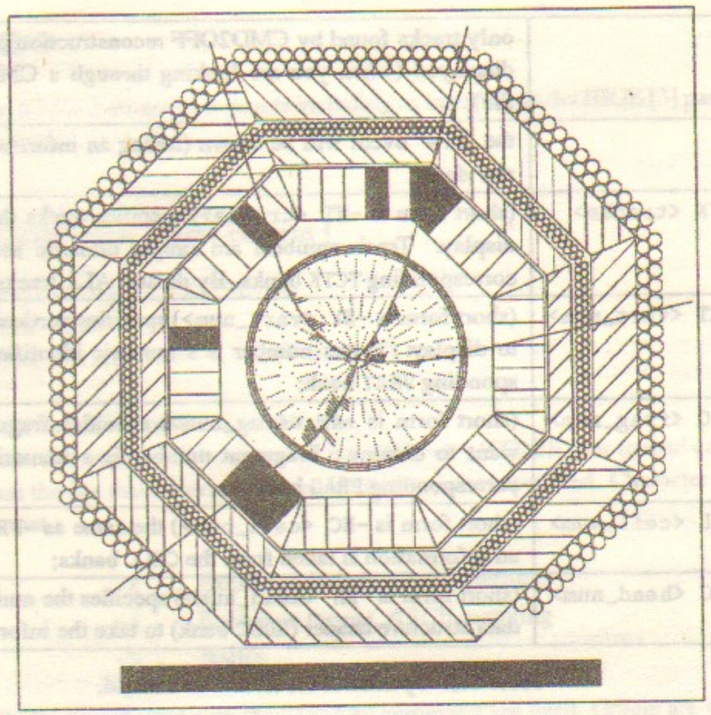


Figure 5.1: An example of output produced by the EVENT command.

5.4 PARAMETERS [freq]

```
FREQ        C 'CSU frequency' D=' '
```

Set or show general CMD-2 parameters.
Now the only parameter is CCS (Central Control Signal) frequency (called "CSU frequency" for historical reasons). This value depends on the run number and is changed automatically when opening new CMD-2 data files. If you do not want ZTREE to change it automatically, you may turn off the CSU attribute with the SET command.

```
Ex.    CMD/PAR ?        ; list of current settings
       CMD/PAR          ; change parameters in interactive mode
       CMD/PAR 200.     ; set CSU frequency to 200 MHz (new value)
```

5.5 METAFILE

A picture generated by ZTREE with DETECTOR and EVENT commands may be written into a metafile. Various metafile types are supported: GKS, PostScript, Encapsulated PostScript, PostScript Color, LaTeX and LaTeX Encapsulated. The most widely used ones from above are PostScript and Encapsulated PostScript metafiles. The former may be directly printed on a PostScript printer. Encapsulated PostScript permits the generation of files which may be included in other documents, for example in

LaTeX files. The size of the picture must be specified by the user with METAFILE/RANGE command. For example if the name of an Encapsulated PostScript file is example.eps , the inclusion of this file into a LaTeX file will be possible via (in the LaTeX file):

```
\begin{figure}
\epsffile{example.eps}
\caption{Example of Encapsulated PostScript in LaTeX.}
\label{EXAMPLE}
\end{figure}
```

Note that all the figures in this manual are included in this way. See [8] for more details. Let's consider an example of writing PostScript metafile:

```
ZTREE> MET/OPEN PICT.PS -111
ZTREE> EVENT
ZTREE> MET/CLOSE
```

That's all. The file PICT.PS may be printed on a PostScript printer.

5.5.1 METAFILE/RANGE [range]

```
RANGE       R 'Range, cm' D=16.0 R=0:
```

This command is used to determine the maximum physical dimension of a picture in a metafile.

5.5.2 METAFILE/OPEN fname [metafi opt]

```
FNAME       C 'Metafile name' D='ZTREE.METAFILE'
METAFL      I 'Metafile ID' D=4
OPT         C '-[option]' D='-S' R='-F,-S'
```

This command opens a metafile and controls the destination of the subsequent graphic output. Use the this command to open a new file, METAFILE/CLOSE to close one.

```
Ex:    MET/OPEN MY.METAFILE ; write GKS metafile
       MET/OPEN A.A -111    ; Direct graphics output to both screen
                              and PostScript metafile
       MET/OPEN B.B -111 -F ; Direct graphics output to PostScript
                              metafile only
```

If the option -F (File) is set, then the subsequent graphic output will be directed to the metafile only. If default option -S (Screen) is set, the subsequent graphic output will be directed to both screen and metafile. The following metafile IDs are valid:

```
METAFL=   4 Appendix E GKS.
METAFL=-111 HIGZ/PostScript (Portrait).
METAFL=-112 HIGZ/PostScript (Landscape).
METAFL=-113 HIGZ/Encapsulated PostScript.
METAFL=-114 HIGZ/PostScript Color (Portrait).
METAFL=-115 HIGZ/PostScript Color (Landscape).
METAFL=-777 HIGZ/LaTeX Encapsulated.
METAFL=-778 HIGZ/LaTeX.
```

5.5.3 METAFILE/CLOSE

This command closes a metafile. The subsequent graphics output will be directed to the screen only.

5.6 CALORIMETER

Set parameters for CsI and BGO calorimeters display. Some parameters are common for both systems (such as hits representation parameters), while others are different.

5.6.1 CALORIMETER/HIT_COLOR [zone color]

```
ZONE        I 'Energy zone' D=0 R=0:
COLOR       I 'Color index' R=0:
```

Set the colors for calorimeter hits display.
You may (and should) set different colors for displaying hits from different energy zones (see section 5.6.2). The number of zones may be changed with the ZONES command. Colors ID 0..7 are predefined as White, Black, Red, Green, Blue, Yellow, Violet and Cyan, respectively (or Black, White, ... in case of GKS). To use more colors (not in GKS versions) you must define the new color's ID with the SET/COLOR command and then you can give this ID as the COLOR parameter for the DC/COLOR, HIT_COLOR, etc.

5.6.2 CALORIMETER/ZONES [number enmax enmin]

```
NUMBER      I 'Number of energy zones' D=0 R=0:
ENMAX       I 'Maximum energy' D=300 R=0:
ENMIN       I 'Minimum energy' D=5 R=0:
```

Define the number of energy zones.
The energy deposition scale is divided into energy zones in order to provide a visual presentation of hits with different energy depositions. Giving ZONES without arguments or with NUMBER=0 will show you the currently defined energy zones and corresponding colors, for example:

```
ZTREE> ZONES
        3 energy zones are currently defined:
+------+-------------+-------+-------+-------+-------+
| ZONE | ENERGY, MeV | COLOR |  RED  | GREEN |  BLUE |
+------+-------------+-------+-------+-------+-------+
|   1  | 5...150     |   1   |  100  |  100  |  100  |
|   2  | 150...300   |   2   |  100  |    0  |    0  |
|   3  | >300        |   3   |    0  |  100  |    0  |
+------+-------------+-------+-------+-------+-------+
ZTREE>
```

When you change current values, ZTREE automatically divides the energy region ENMIN...ENMAX into NUMBER zones. Each zone has its own low energy boundary and color index. These parameters must be set with CALORIMETER/ENERGY and CALORIMETER/HIT_COLOR commands, respectively.

5.6.3 CALORIMETER/ENERGY [zone energy]

```
ZONE        I 'Energy zone' D=0 R=0:
ENERGY      I 'Minimal energy, MeV' R=0:
```

Set the lower boundary for a calorimeter energy zone.

5.6.4 CSI/COLOR [color]

```
COLOR       I 'Color index'  D=-1 R=-1:
```

Set the color for the CsI calorimeter display.

5.6.5 BGO/COLOR [color]

```
COLOR       I 'Color index'  D=-1 R=-1:
```

Set the color for the BGO calorimeter display.

5.6.6 BGO/OPTION [color]

```
OPT         C '? | * | [-][option]' D='?'
```

Set or show BGO display options. One option is now available:

RAW - Raw hits taken from RBG1/RBG2 or CMD2 bank are displayed (default).

```
Ex.     BGO/OPT RAW    ; raw BGO hits will be displayed.
        BGO/OPT RAW    ; BGO hits will not be displayed
        BGO/OPT        ; show active options
        BGO/OPT *      ; set default (RAW option)
```

5.7 DCHAMBER

Set the parameters for drift chamber (DC) display.

5.7.1 DC/COLOR [action object color]

```
ACTION      C 'Action' D='SHOW'
OBJECT      C 'For which object to change the color' D='HIT'
COLOR       I 'Color index' R=0:
```

Set the colors for DC display. Possible ACTION values are:

SHOW	Show colors used for DC display
SET	Set color for DC display

Possible OBJECT values are:

WIRE	DC wire
HIT	Normal hit (with both time and amplitudes)
BAD	Hit without amplitudes
TRACK	Track
VERTEX	Vertex

Colors ID 0..7 are predefined as White, Black, Red, Green, Blue, Yellow, Violet and Cyan, respectively (or Black, White, ... in case of GKS). To use more colors (not in GKS versions) you must define the new color ID with SET/COLOR command and then you can use this ID as the COLOR parameter for DC/COLOR, CSI/COLOR, etc.

5.7.2 DC/MARKER [mtype msc opt]

```
MTYPE     C 'Marker type | ? | *' D=1
MSC       R 'Marker scale factor' D=1
OPTION    C '-[option]' D='-F' R='-R,-F,-V'
```

Set the marker type and the scale factor for DC hits display.
Possible OPTION values are:

- -R Set marker type for "raw" hits
- -F Set marker type for reconstructed hits
- -V Set marker type for vertices

If no parameters are given, ZTREE displays the list of current settings:

```
ZTREE> dc/marker
+-------------------------------------------+
|   Object     | Marker type | Scale factor |
+-------------------------------------------+
| Raw hit      |      1      |     1.0      |
| Hit from FRAG|     28      |     1.0      |
| Vertex       |     30      |     2.0      |
+-------------------------------------------+
ZTREE>
```

If MTYPE='*', the default values are set.

The marker types available are shown in figure 5.2 . All workstations support at least the marker types 1..5 . Additional marker types 20..31 are supported by HIGZ [3]. An example of using different marker types for raw and reconstructed hits display:

```
ZTREE> SCALE 200 -90 0
ZTREE> DC/OPTION FRAG LINE
ZTREE> DC/MARKER  4 1
ZTREE> EVENT
```

The output produced by this example is shown in figure 5.3 .

Simplest markers

Marker type	Marker
1	·
2	+
3	*
4	o
5	×

Markers provided by HIGZ

Marker type	Marker
20	●
21	■
22	▲
23	▼
24	○
25	□
26	△
27	◊
28	✣
29	★
30	☆
31	✳

Figure 5.2: Marker types available.

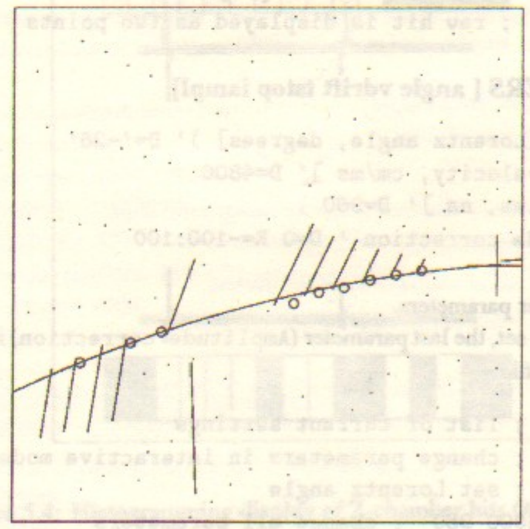


Figure 5.3: Using different marker types for raw and reconstructed hits display.

CHEK	-	Check DC amplitude correction automatically when opening new input file **(default)**
LINE	-	two points corresponding to one raw hit are connected by a straight line **(default).**
RAW	-	raw hits taken from RJDC or CMD2 bank are displayed **(default).**
TCTK	-	Display reconstructed tracks taking information from TCTK banks **(default)**
FRAG	-	display reconstructed DC hits taken from FRAG banks.
VERT	-	display vertices found by the *CMD2OFF* reconstruction program taking information from the VRTX banks.

Table 5.2: DC options.

5.7.3 DC/OPTIONS [opt]

```
OPT      C '[-][option]' D='?'
```

Set or show DC display options.
Current valid options are listed in table 5.2.

To turn option *OPT* off, type `DC/OPTION -OPT`. Note that options RAW and FRAG may be set simultaneously. Therefore, both raw hits taken from RJDC and reconstructed hits from FRAG bank may be displayed. Different markers may be used to distinguish raw and reconstructed hits (see section 5.7.2).
See an example of using different marker types in section 5.7.2

```
Ex.     DC/OPT          ; show active options
        DC/OPT *        ; set default options ( LINE, RAW )
        DC/OPT VERT     ; vertices are displayed
        DC/OPT -LINE    ; raw hit is displayed as two points
```

5.7.4 DC/PARAMETERS [angle vdrift tstop iampl]

```
ANGLE     C '( [?] | [Lorentz angle, degrees] )' D='-26'
VDRIFT    I '[ Drift velocity, cm/ms ]' D=4800
TSTOP     I '[ Stop time, ns ]' D=960
IAMPL     I ' Amplitude correction ' D=0 R=-100:100
```

Set or show the drift chamber parameters.
When the DC option CHEK is set, the last parameter (Amplitude correction) is changed automatically when you open a new input file.

```
Ex.     DC/PAR ?        ; list of current settings
        DC/PAR          ; change parameters in interactive mode
        DC/PAR -32      ; set Lorentz angle
        DC/PAR -25.5 5000 950  ; change all parameters
```


5.8 ZCHAMBER

Set parameters and options for Z-chamber (ZC) display.

5.8.1 ZC/OPTIONS [opt]

```
OPT      C '[-][option]' D='?'
```

Set or show the ZC display options.
Current valid options are listed in the table 5.3. Look at the example of using the AMPL option in figure 5.4 .

```
Ex.     ZC/OPT          ; show active option
        ZC/OPT *        ; set default option
        ZC/OPT ZSCT     ; Take information about hitted sectors
                          from ZSCT bank.
ZC/OPT -ZSCT  ; Take information from raw data banks.
```

AMPL	-	draw Z-chamber (c) hits using bars with the height proportional to their amplitudes (histogramming display).
ZSCT	-	display reconstructed ZC sectors taking information from ZSCT bank

Table 5.3: ZC options.

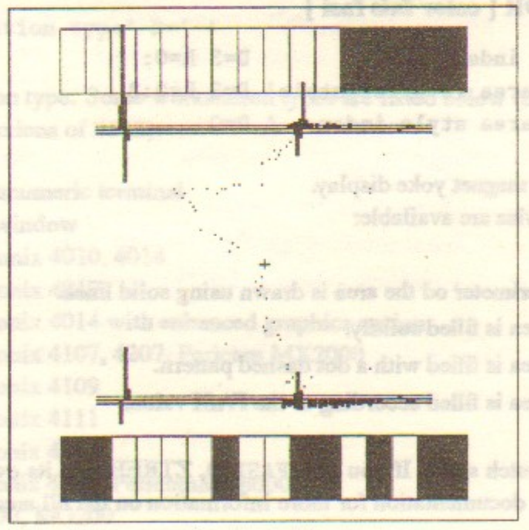


Figure 5.4: Histogramming display of Z-chamber hits (AMPL option).

5.8.2 ZC/PARAMETERS [thres upper lower scale]

```
THRES     C '( [?] | [Amplitude threshold] )' D='50'
UPPER     I 'Upper amplitude threshold' D=4000 R=0:
LOWER     I 'Maximum amplitude' D=4096 R=0:
SCALE     I 'Scale factor for histogramming display ' D=4096 R=0:10000
```

Set or show Z chamber parameters.

```
Ex.   ZC/PAR ?       ; list of current settings
      ZC/PAR         ; change parameters in interactive mode
```

5.9 MUCHAMBER

Set the parameters for MU chamber display.

5.9.1 MU/OPTIONS [muopt]

```
MUOPT     C '[option]' D='?'
```

Set or show MU chamber display options. (Reserved for future use - no options yet)

5.9.2 MU/PARAMETERS [mupar]

```
MUPAR     C 'parameter' D=' '
```

Set or show MU chamber parameters. (Reserved for future use - no parameters yet)

5.9.3 YOKE/COLOR [color fais fasi]

```
COLOR     I 'Color index'                D=3 R=0:
FAIS      I 'Fill area interior style'   D=3 R=0:3
FASI      I 'Fill area style index   '   D=0
```

Set the parameters of the magnet yoke display.
The following interior styles are available:

0	Hollow:	the perimeter od the area is drawn using solid lines.
1	Solid:	the area is filled solidly.
2	Pattern:	the area is filled with a dot dashed pattern.
3	Hatched:	the area is filled according to the FASI value.

FASI is the pattern and hatch style. If you give FASI=0, ZTREE uses its own default values to fill the yoke area. See HIGZ [3] documentation for more information on the fill area styles which are available on your machine.

5.10 SET_SHOW

Set or show various parameters.

5.10.1 SET_SHOW/COLOR_TABLE [color red green blue]

```
COLOR     I 'Color ID' D=-1    R=-1:100
RED       I 'Red intensity  '  R=0:100
GREEN     I 'Green intensity'  R=0:100
BLUE      I 'Blue intensity '  R=0:100
```

Define/change color. Colors ID 0..7 are predefined as White, Black, Red, Green, Blue, Yellow, Violet and Cyan, respectively (or Black, White, ... in case of GKS). The COLOR_TABLE command without arguments or with COLOR=-1 will show you the currently defined colors:

```
ZTREE> COLOR_TABLE
 +-------+-------+-------+-------+
 | COLOR |  RED  | GREEN | BLUE  |
 +-------+-------+-------+-------+
 |     0 |   100 |   100 |   100 |
 |     1 |     0 |     0 |     0 |
 |     2 |   100 |     0 |     0 |
 |     3 |     0 |   100 |     0 |
 |     4 |     0 |     0 |   100 |
 |     5 |   100 |   100 |     0 |
 |     6 |   100 |     0 |   100 |
 |     7 |     0 |   100 |   100 |
 +-------+-------+-------+-------+
ZTREE>
```

This command usually has NO effect in the GKS-based versions of ZTREE!

5.10.2 SET_SHOW/WORKSTATION [wtype]

```
TYPE      C 'Workstation type' D=' '
```

Set or show the workstation type. Some workstation types are listed below (the list may be different for different machines and versions of HIGZ):

0:	Alphanumeric terminal
1-10:	X11 window
101:	Tektronix 4010, 4014
102:	Tektronix 4012
103:	Tektronix 4014 with enhanced graphics option
121:	Tektronix 4107, 4207, Pericom MX2000
122:	Tektronix 4109
123:	Tektronix 4111
125:	Tektronix 4113
127:	Tektronix 4115, PericomMX8000
7800:	MG600, MG200
7878:	Falco, PericomGraph Pac (old Pericom)
1020:	VT240
1030:	VT340
7878:	FALCO terminal

7879:	xterm
8601-6:	Vaxstation GPX
10002:	Apollo DNXXXX monochrome (GPR)
10003-4:	Apollo DNXXXX colour (GPR)
9701-8:	Apollo DNXXXX (GSR)

If WTYPE=' ', the current workstation type is displayed.

If WTYPE='*' the HIGZ routine is called which inquires the workstation type from the standard input. This may be useful to get the list of valid workstation types which is machine and system dependent. NOTE that usually this routine asks you just one time. When you try to call it a second time, it will just return some value without any question.

If WTYPE='?', more information about the current workstation is shown: workstation type, display surface, and number of raster units (Only in ZTREE versions which use GKS as underlying graphics package).

If WTYPE='-L', the list of currently open workstations is shown.

```
Ex.   SET/WORKSTATION 1020  ;    Set VT240 workstation type.
      SET/WORKSTATION       ;    Display current workstation type
```

5.10.3 SET_SHOW/SYSTEMS [list]

```
LIST     C 'List of options' D=' '
```

Select the desired systems or show those selected.
This affects CALIBRATION and EVENT commands. Option from LIST is one of the following:

?	List current settings
.	Unselect all systems
ALL	Select all systems
-<system_id>	Unselect specified system
[+]<system_id>	Select specified system
<opt> <system_id>	Add/delete system's view (see below)

Option '.' unselects all systems, option 'ALL' selects all. <system_id> is a system identifier. It is one of the following:

DC - drift chamber
ZC - Z chamber
CSI - CSI calorimeter
BGO - BGO calorimeter
MU - muon system
YOKE- magnet yoke (to draw or not to draw)

For example, if option -MU is given, muon system is unselected, if option MU is given muon system is selected.

You may wish to have the system displayed only on ONE projection. In this case you must specify the system after one of the following special options:

```
+R <system_id>
-R <system_id>
+Z <system_id>
+Z <system_id>
```

If no parameters are specified with this command, you will be prompted to select systems interactively. To see the list of current settings, just type:

```
ZTREE> SYSTEMS ?
  Drift chamber    ... ON    RZ
  Z - chamber      ... ON    RZ
  CSI calorimeter  ... ON    RZ
  BGO calorimeter  ... ON    -Z
  Muon system      ... ON    RZ
  Magnet yoke      ... ON    RZ
ZTREE>
```

In this example you see that all systems are selected to be displayed in all projections except the BGO calorimeter which will not be displayed in R-φ projection (*cross section*).

```
Ex.   SYSTEMS ?          ;  list of current settings
      SYSTEMS ALL        ;  select all CMD-2 systems
      SYSTEMS .          ;  no systems are selected
      SYSTEMS . DC       ;  select only drift chamber
      SYSTEMS            ;  select systems interactively
      SYSTEMS  -MU       ;  Turn off drawing muon system.
      SYSTEMS  -R BGO    ;  Don't display BGO on cross section
```

5.10.4 SET_SHOW/SCALE [scale xc yc]

```
SCALE   C 'Scale of the picture, mm' D=' '
XC      R 'X coordinate of the picture center, mm' D=0
YC      R 'Y coordinate of the picture center, mm' D=0
```

Set or show the scale and coordinates of the center.

```
Ex.   SCALE ?             ; list of current settings
      SCALE *             ; set default values
      SCALE 500           ; change scale only
      SCALE 1000 500 0    ; change all parameters
      SCALE               ; change parameters in interactive mode
```

5.10.5 SET_SHOW/SECTIONS [opt]

```
OPT       C 'Option' D=' '
```

Select or show sections of CMD-2 to be displayed.
ZTREE provides 3 kinds of event representation (which I call *sections*):

1. *R-φ* projection *(cross section)*.
2. longitudinal projection *(Z-section)*.
3. CSI calorimeter plane projection.

When ZTREE usees X-Windows for low level graphics, it opens a new window for each new section to be displayed. When you use other graphics system, several sections of the detector may be shown simultaneously on one display. In order to see the currently selected sections, type `SECTIONS ?`. Sections may be chosen interactively (typing `SECTIONS` without parameters) or via the `OPT` option. The option must consist of 3 characters (`Y` or `N`). `Y` selects corresponding section, `N` unselects one.

```
Ex.    SECTIONS ?      ;  list of current settings
       SECTIONS        ;  choose sections interactively
       SECTIONS YYY    ;  display all CMD-2 sections
       SECTIONS YNN    ;  display R-φ section only          SECTIONS NYN
```


Chapter 6: ZEBRA

Simple interface to ZEBRA.

6.1 LOGLEVEL [logl opt lun]

```
LOGL      I 'Log level' D=-2 R=-3:4
OPT       C 'Option' D=' ' R=' ,-M,-F'
LUN       I 'Logical unit' D=0
```

Change ZEBRA log level. Possible values of `OPT` options are:

`-M` - set log level for the MZ-package;
`-F` - set log level for the FZ-package.

`LUN` specifies which logical unit to change the log level for (in case of `-F` option). If `LUN=0`, this log level will be set for all open FZ-files.

6.2 STORE [stor]

```
STOR      C 'Store' D='ZTREE' R='ZTREE,PAW'
```

Display the structure of the main ZTREE store. This command calls the DZSTOR routine for the ZTREE or PAW store depending on the parameter.

6.3 ZVERSION

Print Zebra / Kernlib version. This command simply calls the `MZVERS` routine.

6.4 Q addr [value]

```
ADDR      I 'ZEBRA address'
VALUE     R 'The value'
```

6.5 LQ addr [value]

6.6 IQ addr [value]

6.7 IQ2 addr [value]

```
ADDR      I 'ZEBRA address'
VALUE     I 'The value'
```

Using commands 6.4 - 6.7 you may show or change the contents of Q, LQ, IQ, or IQ2 arrays (see ZEBRA reference manual [1] if you do not know what's this). If the `VALUE` parameter is not given, the current value will be printed. Otherwise, it will replaced by the given value.

Chapter 7: KUIP interface

This chapter describes several KUIP features which are very useful and should be used more widely. Some of them are new, such as application defined system functions which are since the spring 1994. This chapter is not intended to be a complete description of of KUIP - the most recent and full information you may find in KUIP manual [2].

7.1 Vectors

Vectors are named arrays of numerical data, memory resident, whose content can be by entered directly by the user, or read from disk files. At the end of an interactive session they are lost, unless previously saved onto disk files.

Vectors can have up to 3 dimensions (in fact they are "arrays", called "vectors" for historical reasons).

The interesting characteristic of vectors is the possibility to handle them either interactively, or in the application program, by mean of KUIP routines which return the addresses of a given vector, thus making its content available to the application.

Simple arithmetic operations can be applied to vectors. In addition, if KUIP is linked with the SIGMA array manipulation package, all the power of SIGMA is automatically available (through the system function $SIGMA). In this case, you may see also chapters 5 (Vectors) and 6 (SIGMA) of PAW Manual[5].

Table 7.1: Addressing scheme for vectors in KUIP

Definition: VECTOR/CREATE V(NCOL)

```
+---+---+---+---+
|   |   | * |   |   * is addressed by V(3)
+---+---+---+---+
```

Definition: VECTOR/CREATE V(NCOL,NROW)

```
+---+---+---+---+          V(:,3)  is the 1-dim array representing the 3rd row
|   |   |   |   |          V(2,:)  is the 1-dim array representing the 2nd column
+---+---+---+---+                  the shortcut notation V(2) can be used as well
|   |   |   |   |
+---+---+---+---+
|   | * |   |   |   * is addressed by V(2,3)
+---+---+---+---+
```

Definition: VECTOR/CREATE V(NCOL,NROW,NPLAN)

```
    +---+---+---+---+
  +---+---+---+---+ |
+---+---+---+---+ | +
|   |   | * |   | + |   * is addressed by V(3,1,1)
+---+---+---+---+ | +
|   |   |   |   | + |
+---+---+---+---+ | +
|   |   |   |   | + |
+---+---+---+---+ | +
|   |   |   |   | + |
+---+---+---+---+
```


7.2 Aliases

Aliases are defined to provide shortcut abbreviations for the input line (either in the command elements or in the parameter list) or for some part of it. An alias name can be any string of characters (except the single quote and the blank) and whenever encountered in an input line it will be replaced literally by its value (another string of characters). Alias substitution does not apply in quoted strings. Aliases are defined by using the command ALIAS/CREATE.

Example of using aliases

```
ZTREE> ALIAS/CREATE M7 'EXEC MACRO7' C
ZTREE> ALIAS/CREATE PP '10 20 30'
ZTREE> ALIAS/LIST
 Argument aliases:
 PP          => 10 20 30
 Command aliases:
 M7          => EXEC MACRO7
ZTREE> M7PP
*** Unknown command
ZTREE> M7 PP
   ...   Executing: MACRO7 10 20 30
ZTREE> MESSAGE M7
 M7
ZTREE> MESSAGE PP
 10 20 30
ZTREE> MESSAGE 'PP'
 PP
```

Note that 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, e.g.

```
ZTREE> Alias/Create LS DIR C    is equivalent to:    ZTREE> DIR
```

Only when LS is the first token on a command line, i.e. in the case below LS will not be translated:

```
ZTREE> SHELL LS
```

Aliases need separators to be recognized in the input line, as evident from the M7PP line in the example above. Possible separators are `blank / , = : . % ' ( )`.

A double slash // can be used to concatenate aliases without any separator (i.e. to juxtapose them):

```
ZTREE> Alias/Create DIR disk$data:[cmd2.raw]
ZTREE> Alias/Create FIL mydatafile
ZTREE> FZ/OPEN 3 DIR//FIL
   ...   Executing: FZFILE/OPEN 3 disk$data:[cmd2.raw]mydatafile
```

Note that aliases are recursive. Example:

```
ZTREE> a/cr aa bb
ZTREE> a/cr bb cc
ZTREE> mess aa
cc
ZTREE> a/cr doc3 'exec doc3'
ZTREE> doc3
*** Line is too long after alias expansion
```

Another way of legally omitting EXEC before the name of a macro, is using the command DEFAULTS -AUTO. After having typed this command, a macro is searched whenever a command is not found: when CMD fails, EXEC CMD is issued automatically. But this is valid only in command mode: this logic is not active within macros, for security and portability reasons.

7.3 KUIP macros

A macro is a set of command lines stored in a file, which can be created/edited with a local editor and executed with the command EXEC. For example the command

```
ZTREE> EXEC MNAME
```

executes the command lines contained in the macro file MNAME. As a macro file can contain several macros, a dash sign (#) is used to select a particular macro inside a file:

- If MNAME does not contain the character #, the file MNAME.KUMAC is searched and the first macro is executed (it may be an unnamed macro if a MACRO statement is not found as first command line in the file),
- If MNAME is of the form FILE#MACRO, the file named FILE.KUMAC is searched and the macro named MACRO is executed.

Example of macro calls

```
ZTREE> EXEC ABC    | Execute first (or unnamed) macro of file ABC.KUMAC
ZTREE> EXEC ABC#M  | Execute macro M of file ABC.KUMAC
```

In addition to all available KUIP commands the special "macro statements" in table 7.2 are valid only inside macros (except for EXEC and APPLICATION, which are valid both inside and outside)

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

Macro Statements	
STATEMENT	DESCRIPTION
MACRO mname var1=val1 ...	begin macro mname
EXEC mname var1 var2=val2 ...	execute macro mname
RETURN	end of a macro
READ var	read macro variable var from keyboard
SHIFT	control parameters list
label:	label (must terminate with a colon)
GOTO label	jump to label
ON ERROR GOTO label	resume at label on error condition
OF ERROR	temporarily deactivate the ON ERROR GOTO handling
ON ERROR	reactivate the latest ON ERROR GOTO handling
IF logical_expression GOTO label	conditional statement
IF-THEN, ELSEIF, ELSE, ENDIF	Macro flow control
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
EXITM	Macro termination
APPLICATION command marker	Inline text passed to application command
var = arithmetic_expression	assignment statement

Table 7.2: List of statements possible inside KUIP macros

7.3.1 Macro definition and variables

A .kumac file can 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 below. For the first macro in the file the MACRO header can be omitted. For the last macro in the file the RETURN trailer can be omitted. Therefore a .kumac file containing only commands (like the LAST.KUMAC) already constitutes a valid macro. Inside a macro the EXEC statement may call other macros.

Positional parameters can be passed to a macro, separated by blanks. Inside a macro, positional parameters can be retrieved by including in brackets the number representing their order in the list.

Example of macro file

```
MACRO ABC
  MESSAGE [1] [3] [2]
RETURN
```

Example of macro execution

```
ZTREE> EXEC ABC P1 123 'This is P3'
P1 This is P3 123
```

Note that normal variables are not translated if they have not been assigned a value, whereas unassigned positional parameters are always replaced by the blank character ' '. Macro parameters can be concatenated to anything in the command line; whenever a parameter number (or name - see below), enclosed in brackets, is encountered in the command line, it will be substituted by its value before execution of the command line.

Non-positional (i.e. named) parameters can also be passed. This is useful when several parameters are associated to a macro. Initial values of parameters should be specified in the MACRO statement.

Parameters can also be read in at macro run time. When a READ statement is executed the user will be asked to provide values for the given parameters. If just <CR> is entered, the values remain unchanged.

Example of macro reading parameters at run time

```
MACRO INP
  READ PPP
  READ 1
  MESSAGE 'The value of the parameter PPP is ... ' [PPP]
  MESSAGE 'The value of the parameter 1 is ..... ' [1]
RETURN
```

The order of priority for macro parameters is such that the values given in the EXEC statement supersede those given in the MACRO statement.

7.3.2 Special Parameters

The following Three special parameters are always defined inside any macro:

- [#] number of arguments given to the macro in the EXEC command which called it.
- [*] String containing the arguments given to the macro in the EXEC command, separated by spaces.
- [@] Return code (see the description of EXITM, page 47) of the last macro called by the current one (0 if no macro has been called).

In addition, it is possible to use **indexed positional parameters** of the form:

- [%var] var is a variable with an integer value. This accesses the positional parameter corresponding to the value of var. If var does not have an integer value then parameters of this form will not be replaced by a value. This can be used in conjunction with the parameter [#] to loop through all of the parameters given to a macro.

Note that positional parameters may not be assigned values using this form.

Assignments

Assignments to a variable simply take the form

```
variable = expression
```

where variable is the name of the variable to be assigned, and expression is the expression which is evaluated to obtain the new value of variable.
Inside a macro, values may be assigned to variables without distinction of their type: an automatic mechanism is used to distinguish between integer, real or character type variables.
The right hand side of an assignment command may be a vector name with an optional subscript, as in the following.

The SHIFT command

Normally when parameters are given to a macro on an EXEC statement, the first one is called [1], the second [2] etc. The SHIFT command "shifts" the values of these parameters so that [1] gets the value of [2], [2] the value of [3], and so on. The previous value of [1] is lost, and the last parameter is assigned the "unassigned" value ' '. This can be useful for looping through the macros parameters with e.g. a WHILE loop.

Example of a macro using the SHIFT command

```
WHILE [1] <> ' ' DO
  I = $LOWER([1])
  CASE [I] IN
  (-ftn) SHIFT
         FTN = [1]
         SHIFT
  (-lib) SHIFT
         LIB = [1]
         SHIFT
  (*)    MESSAGE *** Illegal argument [1] ***
         EXITM 1
  ENDCASE
ENDWHILE
```

The WHILE, CASE and EXITM commands are described later in this chapter.
The same effect can be achieved using indexed positional parameters, a method which has the advantage that the parameters are not lost after the loop has been executed.

Example of a macro using indexed positional parameters

```
K = 1
WHILE [K] <= [#] DO
  I = $LOWER([%K])
  CASE [K] IN
  (-ftn) K = [K]+1
         FTN = [%K]
         K = [K]+1
  (-lib) K = [K]+1
         LIB = [%K]
         K = [K]+1
  (*)    MESSAGE *** Illegal argument [%K] ***
         EXITM 1
  ENDCASE
ENDWHILE
```

7.3.3 Macro Flow Control

There are several constructs available for controlling the flow of macro execution, which include conditional statement blocks, several looping constructs and variable assignments.

Flow Control

The simplest (and most obsolete) of the flow control constructs are the GOTO and IF GOTO statements. GOTO jumps unconditionally to the specified label, while IF GOTO tests the given condition and jumps to the label only if the condition is true. The condition may involve macro variables and arithmetic expressions. These statements are of the form:

```
GOTO label
IF logical_expression GOTO label
```

A label must end with a colon : and is of the form:

```
label_name:
```

The IF-THEN, CASE, WHILE-DO, REPEAT, DO and FOR constructs all involve a block of statements, rather than just a single one. The block begins with the first control statement and ends with the terminating statement for that construct (usually starting with END). The statements in a block may include other block constructs, i.e. they may be nested arbitrarily.

IF-THEN, ELSEIF, ELSE, ENDIF

This allows groups of statements to be executed conditionally depending on the status of one or more conditions.

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

Here the logical expressions are evaluated in turn until either one is found to be true, or an ELSE or an ENDIF is encountered. If an expression is found to be true, or an ELSE is encountered, the statements following it are executed, and then execution jumps to the first statement after the ENDIF. If none of the expressions are true and there is no ELSE component, then nothing happens.

Any number of ELSEIF components are allowed (including zero), and at most one ELSE component is allowed (it may be omitted).

CASE, ENDCASE

This allows groups of statements to be executed conditionally depending on the value of a parameter.

```
CASE parameter IN
(expr_list)  statement   [statements]
(expr_list)  statement   [statements]
...
(expr_list)  statement   [statements]
ENDCASE
```

where expr_list is of the form expr1, expr2, ..., exprN, i.e. a list of expressions separated by commas. Here the comma may be seen as a disjunction operator.
Each expression list is searched for an expression which matches the value of the parameter. When a matching expression is found in one of the lists, the statements following this list are executed and then execution continues from the statement following the ENDCASE. Otherwise, if no matching expression is found, none of the statements are executed and execution jumps straight to after the ENDCASE.
The wildcard character * is allowed in expressions. It matches any string of zero or more characters, e.g. F*T*N will match FORTRAN, FRICTION, FTN, etc.

Example of the use of the wild character

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

WHILE-DO, ENDWHILE

This allows loops WHILE DO where a test is made before executing the statements in the loop. While the result of the test remains true the loop is repeatedly executed. If it is false the loop is terminated and execution jumps to the first statement after the loop. Note that since the test is made at the beginning of the loop it is possible that the loop will never be executed.

```
WHILE logical_expression DO
   statements
ENDWHILE
```

REPEAT, UNTIL

This allows loops REPEAT UNTIL where a test is made after executing the statements in the loop. If the test evaluates to true, execution jumps back to the start of the loop, otherwise the loop is terminated and execution continues at the first statement after the loop. Note that since the test is made at the end, the loop will always execute at least once. The form is:

```
REPEAT
   statements
UNTIL logical_expression
```

DO, ENDDO

This allows loops DO which are executed a predetermined number of times, using an automatically updated counting variable. Its form is:

```
DO variable = start, finish [, step]
   statements
ENDDO
```

The values start, finish and step may be integer or real. step is optional (the default is 1), and may be negative for counting backwards. When the loop is first entered the variable is initialized to start. Before the loop is executed the value of the variable is compared with the limit, finish. If it has passed the limit then execution jumps to the statement after ENDDO. Otherwise the loop is executed and when ENDDO is reached, step is added to the variable and the loop is repeated, starting from the comparison.

Example of Input Macro

```
MACRO DO
   DO I = 1.3,0.5
      J = [I]*[I]
      MESSAGE [I] squared is [J]
   ENDDO
   MESSAGE Now I = [I]
RETURN
```

Output when executing

```
ZTREE> EXEC DO
1 squared is 1
1.5 squared is 2.25
2 squared is 4
2.5 squared is 6.25
3 squared is 9
Now I = 3.5
```

FOR, ENDFOR

This allows loops FOR where a list of values is specified, and these values are given to a loop variable in turn. The loop is executed once for each value of the variable. The special parameter [*] can be used to stand for the list of positional parameters given to the macro on the EXEC line.

```
FOR variable IN values
   statements
ENDFOR
```

where values is a list of values separated by spaces. One or more of the values can be the parameter [*].

Example of Input Macro

```
MACRO FOR
   READ I
   FOR P IN [*] [I]
      SQ = [P]*[P]
      MESSAGE [P] squared is [SQ]
   ENDFOR
RETURN
```

Output when executing

```
ZTREE> EXEC FOR 23 9 48
Macro FOR: I ? 100
23 squared is 529
9 squared is 81
48 squared is 2304
100 squared is 10000
```


BREAKL

The command BREAKL aborts the current WHILE, REPEAT, DO or FOR loop, i.e. jumps to the statement after the terminating statement of the loop.

```
BREAKL
```

Note that if several loops are nested, it is only the innermost loop which is aborted.

EXITM

The command EXITM exits from the current macro with a return value, which is stored in the parameter [@] of the calling macro. If this value is not given it is zero by default.

```
EXITM [value]
```

This can be useful for returning error codes, as shown in the following example:

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
     VEC/CRE VV([NUM])
  ENDIF
RETURN
```

Output when executing

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

7.3.4 Parameters in GOTO and EXEC statements

The macro name given to an EXEC statement, or the label given to a GOTO statement, may contain one or more parameters, which are decoded in the normal way. This is demonstrated in the following example:

Example of Input Macros

```
MACRO DOC10
   NAME='TEST'
   DO I=1,3
      EXEC [NAME][I]
   ENDDO
RETURN
MACRO TEST1
   MESS 'Inside macro TEST1'
RETURN
MACRO TEST2
   MESS 'Inside macro TEST2'
RETURN
MACRO TEST3
   MESS 'Inside macro TEST3'
RETURN
```

Output when executing

```
ZTREE> EXEC DOC10
Inside macro TEST1
Inside macro TEST2
Inside macro TEST3
```

7.4 System functions

KUIP provides a set of built-in which allow, for example, to inquire the current dialogue style or to manipulate strings. An application may provide additional functions. The complete list of available functions can be obtained from HELP FUNCTIONS.

The function name is preceded by $-sign. Arguments are given as a comma separated list of values delimited by "(" and ")". The arguments may be expressions containing other system functions. Functions without arguments must be followed by a character which is different from a letter, a digit, an underscore, or a colon. "$OSMOSIS" will not be recognized as the function "$OS" followed by "MOSIS" If that is the desired effect the concatenation operator has to be used: "$OS//MOSIS". Note however that two functions can follow each other, e.g. "OSMACHINE" because the $-sign does not belong to the function name.

Depending on the setting of the SET/DOLLAR command the name following the $-sign may also be an environment variable. The replacement value for $xxx is obtained in the following order:

1. If xxx is a system function followed by the correct number and types of arguments, replace it by its value.
2. Otherwise if xxx is an argument-less system function, replace it by its value.
3. Otherwise if xxx is a defined environment variable, replace it by its value.
4. Otherwise no replacement takes place.

7.4.1 Alias inquiries

- $ANUM returns the number of *argument* aliases currently defined.
- $ANAM(*n*) returns the name and
- $AVAL(*n*) returns the value of the *n*'th argument alias. No substitution takes place if *n* is not a number between 1 and $ANUM. There is no guarantee that $ANAM($ANUM) refers to the most recently created alias.

7.4.2 Vector inquiries

- $NUMVEC returns the number of vectors currently defined.
- $VEXIST(*name*) returns a positive number if a vector *name* is currently defined. The actual value returned is undefined and may even change between tests on the same *name*. If the vector is undefined the value "0" is returned.
- $VDIM(*name*, *dim*) returns the vector size along index dimension *dim*;*dim*= 1 is used if the second argument is omitted. If the vector is undefined the value "0" is returned.
- $VLEN(*name*) returns for a 1-dimensional vector the index of the last non-zero element. For 2- and 3-dimensional vectors the result is the same as for $VDIM. If the vector is undefined the value "0" is returned.

```
ZTREE> /CREATE v1(10) R 1 2 3 4 0 6
ZTREE> MESS $VDIM(v1) $VLEN(v1)
 10 6
ZTREE> V/CREATE  v2($VLEN(v1))
ZTREE> MESS $VDIM(v2) $VLEN(v2)
 6 0
```

7.4.3 Environment inquiries

- $DATE returns the current date in the format "*dd/mm/yy*"
- $TIME returns the current time in the format "*hh/mm/ss*"
- $RTIME returns the number of seconds elapsed since the previous usage of $RTIME.
- $CPTIME returns the seconds of CPU time spent since the previous usage of $CPTIME.
- $OS returns an identification for the operating system the application is running on, e.g. "UNIX", "VM" or "VMS"
- $MACHINE returns an identification for the particular hardware platform or Unix brand, e.g. "HPUX", "IBM", or "VAX".
- $PID returns the process number on Unix or "1" on other systems.
- $IQUEST(*i*) returns the *i*'th component of the status vector IQUEST. IQUEST(1) always contains the return code of the most recently executed command.

- $ENV(*name*) returns the value of the environment variable *name*, or the empty string if the variable is not defined.
- $FEXIST(*filename*) returns "1" if the file exists, or "0" otherwise.
- $SHELL(*command*) returns the output from the shell command. containing more than one line is concatenated replacing the new lines by blanks.
- $SHELL(*command*, *n*) returns the *n*'th line of output from the shell command.

7.4.4 String manipulations

- $LEN(*string*) returns the number of characters in *string*.
- $INDEX(*string*, *substring*) returns the position of the first occurrence of *substring* inside *string*. or zero if there is none.
- $LOWER(*string*) and
- $UPPER(*string*) return the argument *string* converted to lower or upper case, respectively.
- $SUBSTRING(*string*, *k*, *n*) returns the substring *string(k:k+n-1)*. The argument *n* may be omitted and the result will extend to the end of *string*. Character counting starts with 1. If $n < 0$ e an error message is emitted.

```
ZTREE> MESS $SUBSTRING(abcde,2)/$SUBSTRING(abcde,2,3)
 bcde/bcd
ZTREE> MESS $SUBSTRING(abcde,-2)/$SUBSTRING(abcde,-4,3)
 de/bcd
```

- $WORDS(*string*, *sep*) returns the number of words in *string* separated by the *sep* character. Leading and trailing separators are ignored and strings of consecutive separators count as one only. The second argument may be omitted and defaults to blank as the separator character.

```
ZTREE> MESS $WORDS(',abc,def,,ghi',',')
 3
```

- $WORD(*string*,*k*,*n*,*sep*)(*r*)eturns *n* words starting from word *k*. The last two arguments may be omitted.

```
ZTREE> MESS $WORD('abc def ghi')
 abc
ZTREE> MESS $WORD('abc def ghi',2)
 def
ZTREE> MESS $WORD('abc def ghi',2,2)
 def ghi
```

- $QUOTE(*string*) returns a quoted version of *string*, i.e. the string is enclosed by quote characters and quote characters inside *string* are duplicated. The main use of this function is if an alias value containing blanks should be treated as a single lexical token in a command line.
- $UNQUOTE(*string*) returns a string with enclosed quote characters removed. The main use of this function is if a macro variable should be treated as several blank-separated lexical tokens.

Example of macro using system functions

```
MACRO DOC11 1=' '
IF $VEXIST(VVV) <> 0 THEN
   V/DEL VVV
ENDIF
YN = $SUBSTRING([1],1,1)   | get first character
YN = $UPPER([YN])          | move to upper case
IF [YN]='Y' THEN
   MESSAGE 'You said Yes'
ELSEIF [YN]='N' THEN
   MESSAGE 'You said No'
ELSE
   MESSAGE 'ERROR'
ENDIF
RETURN
```

Output when executing macros DOC11 with various arguments

```
ZTREE> exe doc11
ERROR
ZTREE> exe doc11 y
You said Yes
ZTREE> exe doc11 YE
You said Yes
ZTREE> exe doc11 YOO
You said Yes
ZTREE> exe doc11 no
You said No
```

7.5 ZTREE specific system functions

7.5.1 General purpose functions

- $VERSION() returns ZTREE version, for example "3.72/01".
- $NEVENT() returns current event number if the current record is a CMD-2 event, otherwise the value "0" is returned.
- $FILENAME() returns the name of the current input file. If input file is not open, empty string is returned.
- $FILEMODE() returns the format of the current input file or The following values may be returned:

  X - exchange format, binary;

  N - native format, binary;

  A - ASCII format;

  '' - empty string is returned when input file is not open.

- $FILERECL() returns record length of the current input file in bytes, "0" is file is not open.

7.5.2 ZEBRA storage access

- $LTOP() returns ZEBRA address of top-level bank in current data structure.
- $NBANKS() returns number of banks in current record.
- $IADDR($i$) returns ZEBRA address of i-th bank ($0 < i \leqslant$$NBANKS()). If i is out of bounds the value "0" is returned.
- $INAME($i$) returns name of i-th bank ($0 < i \leqslant$$NBANKS()). If i is out of bounds the value "0" is returned.
- $NBANK($name$) returns number of banks 'name' in current record.
- $ADDRI($i$, $name$) Address of the i-th bank 'name' ($0 < i \leqslant$$NBANK($name$)). If i is out of bounds the value "0" is returned.
- $ADDRN($idn$, $name$) Address of the bank '$name$' with the numeric bank identifier idn. Returns "0" if there is no such bank in the current data structure.
- $ADDR($name$) Address of the first found bank 'name'. This function may be used when you exactly know that there may be only one such bank in the data structure or you simply want to check if any bank with this name exists in the data structure. If bank was not found, $ADDR($name$) returns "0".
- $NAME($addr$) name of the bank at specified ZEBRA address. This function always returns something, so it is better to be sure that there is anything at this address.
- $CTOH($s$) Hollerith representation of 4-char string.
- $HTOC($i$) Character representation of 4-byte Hollerith constant.
- $Q($i$), $LQ($i$), $IQ($i$), $IQ2($i$) return the value of the corresponding ZEBRA array, i.e. $HTOC($IQ(1234)-4) is the same as $NAME(1234)

7.5.3 Output FZ-files inquiries

- $NFILES() Number of open output FZ files.
- $FZFILE($i$) Name of the i-th file.
- $FZLUN($i$) LUN of the i-th file.
- $FZMODE($i$) Mode of the i-th file (A,X or N).
- $FZRECL($i$) Record length of the i-th file in bytes.

Example of macro using ZTREE system functions

```
MACRO find_event number='*' max=500 skip=100
IF ( [1] = '*' ) THEN
   message 'Parameters: <number> [max] [skip]'
   message '   number - event number.'
   message '   max    - maximum number of records to pass. [500]'
   message '   skip   - how often to display current event number. [100]'
   EXITM 1
ENDIF

IF ( $LEN($FILENAME()) = 0 ) THEN
   message ' *** Input file is not open.'
   EXITM 1
ENDIF

set next off
count = 0

DO i=1,[max]
   next
   IF ( $IQUEST(1) <> 0 ) THEN
      message ' *** Error reading input file'
      set next on
      EXITM 1
   ENDIF
   count = [count]+1
   nevt = $NEVENT()
   IF ( [nevt] = [number] ) THEN
      message ' Ok! Event '//[number]//' has been found.'
      set next on
      EXITM
   ENDIF
   IF ( [count] >= [skip] ) THEN
      message [i]//':    event '//[nevt]
      count = 0
   ENDIF
ENDDO

message ' *** Event number '//[number]//' has not been found.'
message ' Current event number is '//$NEVENT()
set next on

RETURN
```

Appendix A: Using COMIS

A.1 How to use COMIS?

COMIS is an interpretation system which can execute a FORTRAN program without compilation. A COMIS program can be created/edited by a local editor and executed with the RUN command. Such programs are called **ZTREE macros** for historical reasons. By default a ZTREE macro file has extension '.ZTREE'. For example the command

```
ZTREE> RUN MNAME
```

executes the program stored in the file MNAME.ZTREE. Actually, the RUN command simply invokes the COMIS FORTRAN interpreter [4]. Therefore, the ZTREE macro is a FORTRAN program which may also contain references to many CERN libraries and ZTREE internal routines. That is why such macros are useful - they can communicate with ZTREE and perform analysis of data read from FZ-file without dealing with ZEBRA, i.e using ZTREE ability to operate with FZ-files.

In the specification of arguments for the calling sequences the FORTRAN-77 conventions are followed, i.e. integer type arguments are starting with I-N, and character type arguments always start with CH.

The scope of variables is INPUT (by default), OUTPUT (if a * follows the name), INPUT-OUTPUT (if a * precedes and follows the parameter's name).

The following routines from the CERN Program Library can be referenced:

From HBOOK:

```
HBOOK1,HBOOK2,HBOOKN,HFILL,HF1,HPRINT,HDELET,HRESET
HFITGA,HFITPO,HFITEX,HPROJ1,HPROJ2,HFN,HGFIT
HROPEN,PAOPEN,PACLOS,PAREAD,PAWRIT,HCDIR,HGIVEN
HTITLE,HBFUN1,HBFUN2,HRNDM1,HRNDM2,HBARX,HBARY
HPAK,HPAKE,HUNPAK,HGIVE,HGN,HGNF,HGNPAR,HF2,HFF1,HFF2
HRIN,HROUT,HI,HIE,HIX,HIJ,HIF,HIDALL,HNOENT,HX,HXY
HTITLE,HCOPY,HSTATI,HBPROF,HOPERA,HIDOPT,HDERIV
HMAXIM,HMINIM,HMAX,HMIN,HSUM,HNORMA,HREND
HEXIST,HRGET,HRPUT,HSCR,HFIND,HCX,HCXY,HLABEL
HBPROX,HBPROY,HBANDX,HBANDY,HBSLIX,HBSLIY
HBOOKB,HBSTAT,HDIFF,HUNPKE,HREBIN,HERROR
HOUTPU,HERMES,HISTDO,HFUNC,HIJXY,HXYIJ,HLPOS,HFC1
HSPLI1,HSPLI2,HMDIR,HLDIR,HLOCAT,HFITH,HFITV,HFINAM
HBNT,HBNAME,HBNAMC,HFNT,HFNTB,HGNT,HGNTF,HGNTV,HBSET
```

From HPLOT:

```
HPLOT,HPLSYM,HPLERR,HPLEGO,HPLNT,HPLSUR,HPLSOF
HPLABL,HPLSET,HPLGIV,HPLOC,HPLTOC,HPLNEW,HPLOPT
```

From ZEBRA:

```
LZFID,LZFIDH,LZLOC,
MZLIFT,MZPUSH,MZDROP,MZFORM,ZSHUNT,
FZIN,FZOUT,FZFILE,FZENDI,FZENDO
RZCDIR,RZLDIR,RZFILE,RZEND,RZIN,RZOUT,RZVIN,RZVOUT
RZOPEN,RZIODO
```


From KUIP:

```
KUGETV,KUDPAR,KUVECT,KILEXP,KUTIME,KUEXEL,KUEXEC
KUPROS,KUNWG,KUCMD,KUGUID,KUNDPV,KUPAR,KUPVAL,KUACT
```

From HIGZ:

```
IPL,IPM,IFA,IGTEXT,IGBOX,IGAXIS,IGPIE,IGRAPH,IGHIST
IGARC,IGLBL,IGRNG,IGMETA,IGSA,IGSET,IRQLC,IRQST,ISCR
ISELNT,ISFAIS,ISFASI,ISLN,ISMK,ISVP,ISWN,ITX,ICLRWK
IGPAVE,IGTERM
```

From KERNLIB:

```
VZERO,UCOPY,RNDM,RANNOR,LENOCC,SBITO,SBIT1,SBYT
JBIT,JBYT,UCTOH,UHTOC,CLTOU,CUTOL
ERF,ERFC,FREQ,PROB,
TIMED
```

The following COMMON blocks may be referenced:

```
/PAWC/, /QUEST/, /ZTREE/
```

The format of /ZTREE/ COMMON block is described in section A.2. Note also that ZEBRA-like routines LZFID, LZFIDH and LZLOC search ONLY for banks in the current data structure.

Any KUIP or ZTREE command may also be executed from a ZTREE macro through KUEXEC routine:

```
CALL KUEXEC('command')
```

In addition to those mentioned above, some other routines may be used inside ZTREE macros to allow the user to access some ZTREE commands directly with subroutine calls. Such access is much more efficient than through KUEXEC call. Some **special routines** are also defined to simplify the work with ZEBRA inside a macro. These routines are described in section A.3.

A.2 ZTREE internal common block

The ZTREE internal COMMON block may be used inside ZTREE macros to work with ZEBRA banks, to alter the contents of a bank or to change the structure of data stored in memory. It is much more convenient than to write a separate program because you do not worry about ZEBRA initialization, reading and writing FZ-files. The /ZTREE/ COMMON block has the following format:

```
PARAMETER (NWORD=500000)
INTEGER IQ(NWORD),LQ(NWORD)
REAL Q(NWORD)
EQUIVALENCE (IQ(1),Q(1),LQ(9)),(LQ(1),LTOP)
COMMON /ZTREE/ IXSTOR,IXSYST,IXWORK,IXRUN,FENCE(16),LTOP,
+                LA(NWORD),LASTA
```

where LTOP is a pointer to the top-level bank of the current data structure. But actually you do not need to put this COMMON block in every one of your COMIS programs. Alternatively you may use the **special routines** defined for use inside ZTREE macros - LTOP(I) function instead of LTOP, IQ(N) function instead of IQ(NWORD) array, etc. See the detailed description of these routines in section A.3.

A.3 Special routines

Below is the list of special routines that can be referenced from the ZTREE macros. In the specification of arguments the FORTRAN-77 conventions are followed, i.e. integer type arguments are starting with I-N, and character type arguments always start with CH. The scope of variables is always INPUT.

A.3.1 Input/output files control

```
CALL NEXT
```

Action: reads next data structure from the input file (the same as the NEXT command).

```
FLAG = EOF  (IDUMMY)
FLAG = SOR  (IDUMMY)
FLAG = EOR  (IDUMMY)
```

Action: logical functions. Returns .TRUE. if current position is End-Of-File, Start-Of-Run or End-Of-Run, respectively.

Parameter Description:

IDUMMY dummy argument

```
CALL FZOPEN  (LUN,CHNAME)
```

Action: opens output FZ-file CHNAME on unit LUN (see also FZFILE/OPEN command).

Parameter Description:

LUN	logical unit number
CHNAME	file name

```
CALL FZWRITE  (LUN)
```

Action: writes current data structure into the FZ-file previously open on unit LUN (see also FZFILE/WRITE command).

Parameter Description:

LUN	logical unit number

```
CALL FZCLOSE  (LUN)
```

Action: closes the output FZ-file previously open on unit LUN (see also FZFILE/CLOSE command).

Parameter Description:

LUN	logical unit number

A.3.2 Data manipulation

```
NB = NBANKS  (CHNAME)
```

Action: returns the number of banks CHNAME in the current data structure.

Parameter Description:

CHNAME bank name (CHARACTER*4)

```
N = NEVENT  (IDUMMY)
```

Action: returns the event number if the current data structure is an event record, else returns 0.

Parameter Description:

IDUMMY dummy argument

```
L = LTOP  (IDUMMY)
```

Action: returns the ZEBRA address of the top-level bank in the current data structure.

Parameter Description:

IDUMMY dummy argument

```
I = IQ  (N)
I = IQ2  (N)
I = LQ  (N)
X = Q  (N)
```

Action: four above functions return the value of elements of arrays from /ZTREE/ common block. They may be used instead of ZEBRA arrays but you do not need to put ZTREE common block in your macro.

Parameter Description:

N ZEBRA address

```
CALL PUTIQ  (N,IVAL)
CALL PUTIQ2  (N,IVAL)
CALL PUTLQ  (N,IVAL)
CALL PUTQ  (N,VAL)
```

Action: Four above subroutines allow the user to change the contents of the ZEBRA arrays, i.e. IQ(N), IQ2(N), LQ(N) and Q(N), respectively.

Parameter Description:

N	ZEBRA address
IVAL,VAL	the value

A.3.3 Parameter retrieval

```
N = IARGC (IDUMMY)
```

Action: returns the number of parameters, supplied in the command line after the macro file name.

Parameter Description:

IDUMMY dummy argument

```
STATUS = GETARGS (N,CHPAR,L)
```

Action: logical function. Gets the string type parameter number N from the command line. Returns .TRUE. if parameter was given.

Parameter Description:

N	parameter's number.
CHPAR*	the CHARACTER parameter
L*	logical length of the returned string, i.e. without trailing blanks.

```
STATUS = GETARGI (N,IPAR)
```

Action: logical function. Gets the integer type parameter from the command line. Returns .TRUE. if parameter is present and successfully decoded.

Parameter Description:

N	parameter's number.
IPAR*	the parameter

```
STATUS = GETARGR (N,RPAR)
```

Action: logical function. Gets the real type parameter from the command line. Returns .TRUE. if parameter is present and successfully decoded.

Parameter Description:

N	parameter's number.
RPAR*	the parameter

A.4 Macro Example

ZTREE macro example

```
c-------------------------------------------------------------
c
c Example 1. Find an event with specified number
c
c-------------------------------------------------------------
      PROGRAM FE
      INTEGER NCUR,EVENT_NUMBER,NUMEV,I,MAX
      COMMON /QUEST/ IQUEST(100)
      DATA MAX/500/
      IF(IARGC(IDUMMY).EQ.0) THEN
        PRINT *,' Usage FE <number> [max]'
        PRINT *,'   number - event number.'
        PRINT *,'   max    - maximum number of records to pass.'
        PRINT *,'            Default is ',MAX
        STOP
      ENDIF
      CALL KUEXEC('SET NEXT OFF')
10000 FORMAT('  Searching for event number ',I7,' ...')
      IF(.NOT.GETARGI(1,EVENT_NUMBER)) THEN
        PRINT *,' *** Invalid parameter'
        STOP
      ENDIF
      IF(GETARGI(2,IPAR)) MAX = IPAR
      PRINT 10000,EVENT_NUMBER
      DO 10 I=1,MAX
        CALL NEXT
        IF(IQUEST(1).GT.2.OR.IQUEST(1).LT.0) THEN
          PRINT *,' *** Error reading input file'
          STOP
        ENDIF
        NCUR = NEVENT(IDUMMY)
        IF ( NCUR.EQ.EVENT_NUMBER ) THEN
          PRINT *,' Ok! Event is found.'
          GOTO 100
        ENDIF
10    CONTINUE
      PRINT *,' Such event is not found.'
      IF(NCUR.NE.0) PRINT *,' Current events number is ',NCUR
100   CALL KUEXEC('SET NEXT ON')
      STOP
      END
```

Appendix B: Documentation Files

B.1 ZTREE documentation files format

The nature of the contents of any bank must be indicated to ZTREE via the documentation files which may contain a description of the bank format and comments. Two documentation files may exist simultaneously: the **local** file and the **main** file. The main file usually contains a description of the banks used by many users. But if the user works with his(her) own ZEBRA banks, he(she) can create a local documentation file containing their descriptions. First ZTREE tries to open the local documentation file. If this file does not exist or the bank format is not found, ZTREE uses the main one. You may change documentation files names using `DOC_FILE` command.

The documentation files contain bank descriptions in ZEBRA format (see help on `MZFORM` routine, [1]) following bank names, for example:

```
FRCS 1I 1F / 1I 1F ! information about "free" crystals
```

A bank description line may be followed by comment lines which are started by a space (a description lines must start from the first position):

```
FRCS 1I 1F / 1I 1F ! an information about "free" crystals
 Number of crystals out of clusters
 Energy deposition in the crystals, MeV
 Crystal number
 Energy deposition in the crystal,MeV
```

Every comment line corresponds to 4 bytes of data. Note that there are some differences between ZTREE data type identifier syntax and ZEBRA. One reason is because using ZTREE you must specify not only the data type but also how to represent this data (for example, in octal or decimal notation). The possible data type identifiers are listed in table B.1.

B	bit string of 32 bits
b	bit string of 16 bits
I	integer*4
i	integer*2
O	integer*4 in octal presentation
o	integer*2 in octal presentation
F	floating-point
H	4-character Hollerith

Table B.1: ZTREE data type identifier syntax

If you use 2-byte integers stored in a ZEBRA bank, you may specify comments for every one. These comments must stay on a one line separated with '\':

```
HEVT 1i 1b 1B 1I 1B ! the header of the reconstructed event
 Run number\   Date
 Time
 Event number
 Trigger
```

B.2 Producing bank documentation

When you write programs using ZEBRA, you need a paper describing the formats of the banks used. In order to produce such kind of documentation, the `DESCRIPTION` command is provided (described in section 3.7 on page 18). For example, the command

```
ZTREE> DESCRIPT FRCS
```

will create the plain text file containing the bank description shown on figure B.1. The file name will be `FRCS.DOC`, if the name of the ZTREE description file is set to `*.DOC` (this may be set with `DOC_FILE` command described in section 3.8 on page 18). In any case ZTREE informs you of which files has been created:

```
ZTREE> DESCRIPT C*
 File CLUS.DOC is created.
 File CELL.DOC is created.
 File COND.DOC is created.
ZTREE>
```

Note, that both the bank name and the description file name using the `DESCRIPTION` command may contain wildcards. The '*' symbols in the file name are replaced by the name of the bank found in the documentation file. The documentation may be produced in the LaTeX [8] format using option `-LATEX`:

```
ZTREE> DOC_FILE *.TEX -D
ZTREE> DESCRIPT -LATEX FRCS
 File FRCS.TEX is created.
ZTREE>
```

This makes a file to be processed by LaTeX. The output is shown on figure B.2 on page 62. If you need to create a single file containing the formats of the all banks known to ZTREE, you simply type:

```
ZTREE> DOC_FILE BANKS.TEX -D
ZTREE> DESCRIPT -LATEX *
 File BANKS.TEX is created.
ZTREE>
```

Of course you may omit `-LATEX` option, if you want the plain text file.

```
        Bank FRCS - an information about "free" crystals
+-------------------------------------------------------------------------+
!  Offset     ! Type   !          Content                                  !
+-------------------------------------------------------------------------+
!    +1       ! Int*4  ! Number of crystals out of clusters               !
!    +2       ! Real   ! Energy deposition in the crystals, MeV           !
!                                                                         !
!                          . . . . . . . . . .                            !
!                                                                         !
!  +3+2*(n-1) ! Int*4  ! Crystal number                                   !
!  +4+2*(n-1) ! Real   ! Energy deposition in the crystal,MeV             !
!                                                                         !
!                          . . . . . . . . . .                            !
!                                                                         !
+-------------------------------------------------------------------------+
```

Figure B.1: An example of documentation produced by `DESCRIPTION` command in a plain text format.

Bank FRCS - an information about "free" crystals

Offset	Type	Content
+1	Int*4	Number of crystals out of clusters
+2	Real	Energy deposition in the crystals, MeV
		. . . . .
+3+2*(n-1)	Int*4	Crystal number
+4+2*(n-1)	Real	Energy deposition in the crystal,MeV
		. . . . .

Figure B.2: An example of documentation produced by `DESCRIPTION` command in LaTeX format.

Bibliography

[1] R.Brun, M.Goossens and J.Zoll. *ZEBRA Users Guide*, CERN Program Library Q100. CERN, 1991.

[2] *KUIP - Kit for an User Interface Package*, CERN Program Library I102. CERN, 1993.

[3] *HIGZ/HPLOT Users Guide* , CERN Program Library Q120 and Y251. CERN, 1993.

[4] *COMIS - Compilation and Interpretation System* , CERN Program Library L210. CERN, 1993.

[5] R.Brun, O.Couet, C.Vandoni and P.Zanarini. *PAW users guide*. CERN Program Library Q121. CERN, 1991.

[6] *CMZ - A Source Code Management System*. User's Guide and Reference Manual. CodeME S.A.R.L., 1991.

[7] PDP-11 ON-LINE DATA ACQUISITION SYSTEM MANUAL. CERN, 1983.

[8] L.Lamport, LaTeX: *A Document Preparation System*. Addison-Wesley, 1986.

Table of Contents