CERN Program Library Long Writeup Q121

PAW

Physics Analysis Workstation

User's guide

Information Technology Division

CERN, Geneva, Switzerland

```
CERN Program Library Office
CERN-IT Division
CH-1211 Geneva 23
Switzerland
Tel.  +41 22 767 4951
Fax.  +41 22 767 8630
Internet:  cernlib@cern.ch
```


Contact Person: Olivier Couet (`Olivier.Couet@cern.ch`)

Document Consultant: Michel Goossens (`Michel.Goossens@cern.ch`)

Edition – January 1999

About this guide

Preliminary remarks

In this manual examples are in `monotype face` and strings to be input by the user are `underlined`. In the index the page where a command is defined is in **bold**, page numbers where a routine is referenced are in normal type.

Related Manuals

This document can be complemented by the following manuals:

- COMIS, Compilation and Interpretation System [1]
- HBOOK User Guide — Version 4 [2]
- HIGZ-HPLOT — High level Interface to Graphics and ZEBRA and HPLOT User Guide [3]
- KUIP — Kit for a User Interface Package [4]
- MINUIT — Function Minimization and Error Analysis [5]
- ZEBRA — Data Structure Management System [6]

This document present the basic concepts of PAW. For more detailed and up to date informations on the system it is strongly recommended to look at the following URL:

`http://wwwcn.cern.ch/pl/paw/`

Acknowledgements

The authors of PAW would like to thank all their colleagues who, by their continuous interest and encouragement, have given them the necessary input to provide a modern and easy to use data analysis and presentation system.

Table of Contents

Chapter 1: A few words on PAW

1.1 A short history

At the beginning of 1986 the **P**hysics **A**nalysis **W**orkstation project **PAW** was launched at CERN. The first public release of the system was made at the beginning of 1988. At present the system runs on most of the computer systems used in the High Energy Physics (HEP) community (Mainframes, Workstations, PC's). In addition to its powerful data analysis, particular emphasis has been put on the quality of the user interface and of the graphical presentation.

1.2 What is PAW?

PAW is an interactive utility for visualizing experimental data on a computer graphics display. It may be run in batch mode if desired for very large and time consuming data analyses; typically, however, the user will decide on an analysis procedure interactively before running a batch job.

PAW combines a handful of CERN High Energy Physics Library systems that may also used individually in software that processes and displays data. The purpose of PAW is to provide many common analysis and display procedures that would be duplicated needlessly by individual programmers, to supply a flexible way to invoke these common procedures, and yet also to allow user customization where necessary.

1.3 What Can You Do with PAW?

PAW can do a wide variety of tasks relevant to analyzing and understanding physical data, which are typically statistical distributions of measured events. Below we list what are probably the most frequent and best-adapted applications of PAW; the list is not intended to be exhaustive, for it is obviously possible to use PAW's flexibility to do a huge number of things, some more difficult to achieve than others within the given structure.

Typical PAW Applications:

- **Plot a Vector of Data Fields for a List of Events.** A set of raw data is typically processed by the user's own software to give a set of physical quantities, such as momenta, energies, particle identities, and so on, for each event. When this digested data is saved on a file as an Ntuple, it may be read and manipulated directly from PAW. Options for plotting Ntuples include the following:
  - *One Variable.* If a plot of a one variable from the data set is requested, a histogram showing the statistical distribution of the values from all the events is automatically created. Individual events are not plotted, but appear only as a contribution to the corresponding histogram bin.
  - *Two or Three Variables.* If a plot of two or three variables from the data set is requested, no histogram is created, but a 2D or 3D scatter plot showing a point or marker for each distinct event is produced.
  - *Four Variables.* If a plot of four variables is requested, a 3D scatter plot of the first three variables is produced, and a color map is assigned to the fourth variable; the displayed color of the individual data points in the 3D scatter plot indicates the approximate value of the fourth variable.
  - *More than Four Variables.* More than four variables can be plotted but it is up to the user to customize the system in order to assign the additional variables to graphics attributes like the size or the shape (type) of the markers.
  - *Vector Functions of Variables.* PAW allows the user to define arbitrary vector functions of the original variables in an Ntuple, and to plot those instead of the bare variables. Thus one can easily plot something like $\sqrt{(P_x^2 + P_y^2)}$ if P_x and P_y are original variables in the data without having to add a new data field to the Ntuple at the time of its creation.
  - *Selection Functions (Cuts).* PAW does not require you to use every event in your data set. Several methods are provided to define Boolean functions of the variables themselves that pick out subsets of the events to be included in a plot.
  - *Plot presentation options.* The PAW user can set a variety of options to customize the format and appearance of the plots.

- **Histogram of a Vector of Variables for a List of Events.** Often one is more interested in the statistical distribution of a vector of variables (or vector functions of the variables) than in the variables themselves. PAW provides utilities for defining the desired limits and bin characteristics of a histogram and accumulating the bin counts by scanning through a list of events. The following are some of the features available for the creation of histograms:

  – *One Dimensional Histograms.* Any single variable can be analyzed using a one-dimensional histogram that shows how many events lie in each bin. This is basically equivalent to the single-variable data plotting application except that it is easier to specify personalized features of the display format. A variety of features allow the user to slice and project a 2D scatter plot and make a 1D histogram from the resulting projection.

  – *Two-Dimensional Histograms.* The distribution of any pair of variables for a set of events can be accumulated into a 2D histogram and plotted in a various of ways to show the resulting surface.

  – *Vector Functions of Variables.* User-defined functions of variables in each event can be used to define the histogram, just as for an Ntuple plot.

  – *Selection Functions (Cuts).* Events may also be included or excluded by invoking Boolean selection functions that are arbitrary functions of the variables of a given event.

  – *Event Weights.* PAW allows the user to include a multiplicative statistical bias for each event which is a scalar function of the available variables. This permits the user to correct for known statistical biases in the data when making histograms of event distributions.

  – *Histogram Presentation Options.* Virtually every aspect of the appearance of a histogram can be controlled by the user. Axis labels, tick marks, titles, colors, fonts, and so on, are specified by a large family of options.

- **Fit a Function to a Histogram.** Once a histogram is defined, the user may fit the resulting shape with one of a family of standard functions, or with a custom-designed function. The parameters of the fit are returned in user-accessible form. Fitted functions of one variable may be attached to a 1D histogram and plotted with it. The capability of associating fits to higher dimensional histograms and overlaying their representations on the histogram is in the process of being added to PAW.

  The fitting process in PAW is normally carried out by the MINUIT library. To user this package effectively, users must typically supply data with reasonable numerical ranges and give reasonable initial conditions for the fit before passing the task to the automated procedure.

- **Annotate and Print Graphics.** A typical objective of a PAW user is to examine, manipulate, and display the properties of a body of experimental data, and then to prepare a graph of the results for use in a report, presentation, or publication. PAW includes for convenience a family of graphics primitives and procedures that may be used to annotate and customize graphics for such purposes. In addition, any graphics display presented on the screen can be converted to a PostScript file for black-and-white or color printing, or for direct inclusion in a manuscript.

1.4 A User's View of PAW

In order to take advantage of PAW, the user must first have an understanding of its basic structure. Below we explain the fundamental ways in which PAW and the user interact.

Initialization. PAW may be invoked in a variety of ways, depending on the user's specific computer system; these are described in the following chapter. As PAW starts, it prompts the user to select an interaction mode (or non-interactive mode) and window size and type (if interactive). The available window sizes and positions are specified in the user file `"higz_windows.dat"`. User-specific intializations are specified in the file `"pawlogon.kumac"`.

Command Mode Interface. The most basic interface is the **KUIP "command mode" interface.** KUIP provides a basic syntax for commands that are parsed and passed on to the PAW application routines to perform specific tasks. Among the basic features of KUIP with which the user interacts are the following:

- *Command Entry.* Any unique partially entered command is interpreted as a fully entered command. KUIP responds to an ambiguous command by listing the possible alternatives. On Unix systems, individual command lines can be edited in place using individual control keystrokes similar to those of the emacs editor, or the `bash` or `tcsh` Unix command shells. On other systems, a command line that is in error can only be revised after it is entered, using the VAX/VMS editor "EDT" style text line editing language.
- *Parameters.* Parameters are entered after the basic command on the same line and are separated by spaces. If a parameter has embedded blanks, it must be it must be put between quotes. An exclamation point `(!)` can be used to keep the default parameters in a sequence when only a later parameter is being changed. If an underscore `(_)` is the last character on a line, the command may be continued on the next line; no spaces are allowed in the middle of continued parameter fields.
- *On-Line Assistance.* The `"usage"` and `"help"` commands can be used to get a short or verbose description of parameters and features of any command.
- *Command History.* A command history is kept both in memory for interactive inspection and on a disk file. The command history file can be recovered and used to reconstruct a set of actions carried out interactively.
- *Aliases.* Allow the abbreviation of partial or complete command sequences.
- *Macros.* A text file containing PAW commands and flow control statements.

KUIP/MOTIF Interface. If the user's workstation supports the OSF/Motif windowing system, PAW can be started in the KUIP/MOTIF mode: the executable module to be run in that case is called PAW++. However, a small text panel and a command history panel keep track of individual actions, and permit entry and recall of typed commands similar to the command mode interface.

The basic features of this interface are:

- *Pull-Down Menu "Commands".* Each PAW command (that can be given in input) has a corresponding item in a hierarchical pull-down menu (entry "Commands"). Commands that require arguments cause a parameter-entry dialog box to appear; when the arguments are entered and command execution requested (button "OK" or "Execute"), the command is executed as though typed from the command mode interface.
- *Action Panel(s).* A user may have a family of frequently executed macros or commands assigned to specific buttons on the action panel(s). These panels are totally user definable.
- *Object Browser.* All the objects known in PAW (Histograms, Ntuples, Vectors etc...) can be manipulated via icons and pull-down menus in the "Object Browser".
- *Direct Graphics Interaction.* One can click in the graphics area and identify automatically which object has been selected. A pop-up menu appears with a list of possible actions on this object.

Graphics Output Window. The graphics image produced by PAW commands, regardless of the command interface, appears on a separate graphics output window. The actual size and position of this window on the screen is controlled by a list of numbers of the form `x-upper-left y-upper-left x-width y-height` in the user file `higz_windows.dat`. The width and height of the drawing area within this window are subject to additional user control, and the user can specify "zones," which are essentially ways of dividing the window into panes to allow simultaneous display of more than one plot. Some picking facilities are also available.

1.5 Fundamental Objects of PAW

PAW is implicitly based on a family of fundamental objects (see figure 1.1 on the following page). Each PAW command performs an action that either produces another object or produces a "side-effect" such as a printed message or graphics display that is not saved anywhere as a data structure. Some commands do both, and some may or may not produce a PAW data structure depending on the settings of global PAW parameters. In this section, we describe the basic objects that the user needs to keep in mind when dealing with PAW. The reader should perhaps note that the PAW commands themselves do not necessarily reflect the nature of PAW objects as clearly as they might, while the MOTIF interactive graphics interface in fact displays distinct icons for most of the object types listed below.

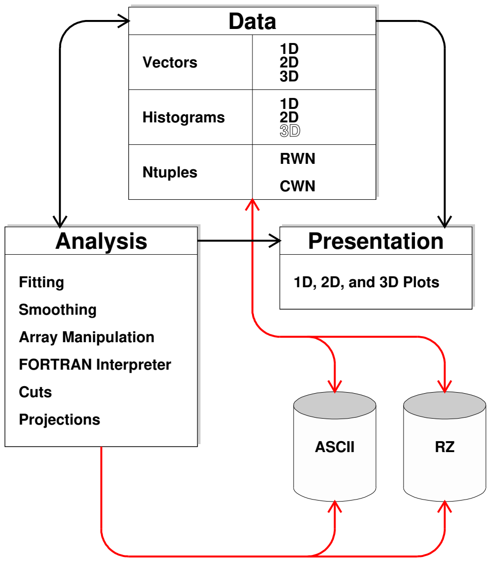


Figure 1.1: PAW's fundamental "data" objects

Objects:

- **1D Histograms.** A histogram is the basic statistical analysis tool of PAW. Histograms are created ("booked") by choosing the basic characteristics of their bins, variables, and perhaps customized display parameters; numbers are entered into the histogram bins from an Ntuple (the histogram is "filled") by selecting the desired events, weights, and variable transformations to be used while counts are accumulated in the bins. Functional forms are frequently fit to the resulting histograms and stored with them. Thus a fit as an object is normally associated directly with a histogram, although it may be considered separately.

- **2D Histograms.** 2D (and higher-dimensional) histograms are logical generalizations of 1D histograms. 2D histograms, for example, are viewable as the result of counting the points in a the sections of a rectangular grid overlaid on a scatter plot of two variables. Higher-dimensional histograms can also be fitted, and support for associating the results of a fit to a higher-dimensional histogram is currently being incorporated in PAW.

- **Ntuples.** An Ntuple is the basic type of data used in PAW. It consists of a list of identical data structures, one for each event. Typically, an Ntuple is made available to PAW by opening a HBOOK file; this file, as created by HBOOK, contains one or more Ntuples and possibly also directories, which may store a hierarchy of Ntuples and histograms. A storage area for an Ntuple may be created directly using `NTUPLE/CREATE`; data may then be stored in the allocated space using the `NTUPLE/LOOP` or `NTUPLE/READ` commands. Other commands merge Ntuples into larger Ntuples, project vector functions of the Ntuple variables into histograms, and plot selected subsets of events.

- **Cuts.** A cut is a Boolean function of Ntuple variables. Cuts are used to select subsets of events in an Ntuple when creating histograms and ploting variables.

- **Masks.** Masks are separate files that are logically identical to a set of boolean variables added on the end of an Ntuple's data structure. A mask is constructed using the Boolean result of applying a cut to an event set. A mask is useful only for efficiency; the effect of a mask is identical to that of the cut that produced it.

- **Vectors.** PAW provides the facilities to store vectors of integer or real data. These vectors, or rather arrays with up to 3 index dimensions, can be manipulated with a set of dedicated commands. Furthermore they are interfaced to the array manipulation package SIGMA and to the Fortran interpreter COMIS. They provide a convenient and easy way to analyse small data sets stored in ASCII files.

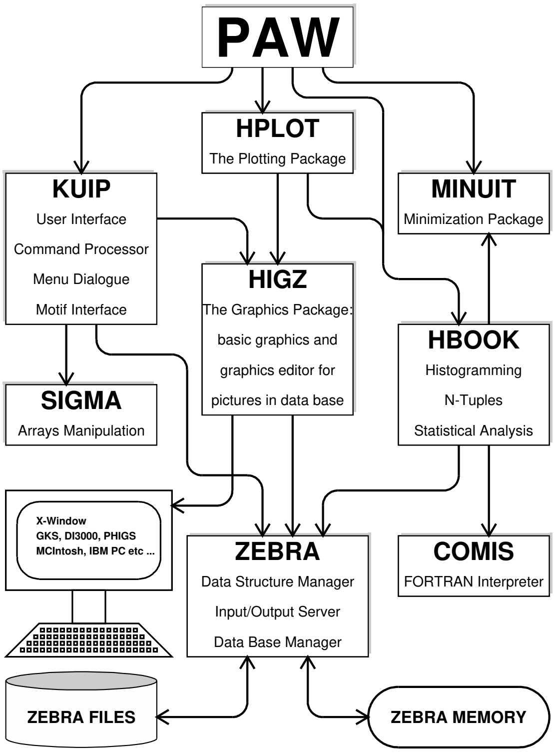


Figure 1.2: PAW and its components

- **PostScript (meta)files.** PostScript format (meta)files are especially useful because they can be directly printed on most printers; furthermore, the printed quality of graphics objects such as fonts can be of much higher quality than the original screen image.

- **Pictures.** A *picture* is an exact copy of the screen image, and so its storage and redisplay time are independent of complexity. Pictures are also intensively used for object picking in the Motif version of PAW.

- **ZEBRA(RZ) Logical Directories.** In a single PAW session, the user may work simultaneously with many Ntuples, histograms, and hierarchies of Ntuple and histograms. However, this is not accomplished using the native operating system's file handler. Instead, the user works with a set of objects that are *similar* to a file system, but are instead managed by the ZEBRA RZ package. This can be somewhat confusing because a single operating system file created by RZ can contain an entire hierarchy of ZEBRA logical directories; furthermore, sections of internal memory can also be organized as ZEBRA logical directories to receive newly-created PAW objects that are not written to files. A set of commands `CDIR`, `LDIR`, and `MDIR` are the basic utilities for walking through a set of ZEBRA logical directories of PAW objects; Each set of directories contained in an actual file corresponds to a logical unit number, and the root of the tree is usually of the form `//LUNx`; the PAW objects and logical directories stored in internal memory have the root `//PAWC`. A macro is a set of command lines stored in a file, which can be created or modified with any text editor. In addition to all the PAW commands, special macro flow control statements are also available.

- **Operating System File Directories.** Many different ZEBRA files, some with logically equivalent Ntuples and histograms, can be arranged in the user's operating system file directories. Thus one must also keep clearly in mind the operating system file directories and their correspondence to the ZEBRA logical directories containing data that one wishes to work with. In many ways, the operating system file system is also a type of "object" that forms an essential part of the user's mental picture of the system.

1.6 The Component Subsystems of PAW

The PAW system combines different tools and packages, which can also be used independently and some of which have already a long history behind them (e.g. HBOOK and HPLOT, SIGMA, COMIS, MINUIT). Figure 1.2 shows the various components of PAW.

1.6.1 KUIP - The user interface package

The purpose of KUIP (**K**it for a **U**ser **I**nterface **P**ackage) is to handle the dialogue between the user and the application program (PAW in our case). It parses the commands input into the system, verifies them for correctness and then hands over control to the relevant action routines.

Commands are grouped in a tree structure and they can be **abbreviated** to their shortest unambiguous form. If an ambiguous command is typed, then KUIP responds by showing all the possibilities. **Aliases** allow the user to abbreviate part or the whole of commonly used command and parameters. A sequence of PAW commands can be stored in a text file and, combined with flow control statements, form a powerful **macro** facility. With the help of **parameters**, whose values can be passed to the macros, general and adaptable task solving procedures can be developed.

The user has the choice between different dialogue styles ranging from the conventional command line interface to a high-level windowed environment based on OSF/Motif . In order to save typing, **default values**, providing reasonable settings, can be used for most parameters of a command. A **history file**, containing the n most recently entered commands, is automatically kept by KUIP and can be inspected, copied or re-entered at any time. The history file of the last PAW session is also kept on disk.

1.6.2 HBOOK and HPLOT - The histograming and plotting packages

HBOOK and its graphics interface HPLOT are libraries of FORTRAN callable subroutines which have been in use for many years. They provide the following functionality:

- One- and two-dimensional histograms and Ntuples
- Projections and slices of two-dimensional histograms and Ntuples
- Complete control (input and output) of the histogram contents
- Operations and comparison of histograms
- Minimization and parameterization tools
- Random number generation
- Histograms and Ntuples structured in memory (directories)
- Histograms and Ntuples saved onto direct access ZEBRA files
- Wide range of graphics options:
  - Contour histograms, bar chart, shaded histograms, error bars, colour
  - Smoothed curves and surfaces
  - Scatter, lego, contour and surface plots
  - Automatic windowing
  - Graphics input

1.6.3 HIGZ - The graphics interface package

A **H**igh level **I**nterface to **G**raphics and **Z**EBRA (HIGZ) has been developed within the PAW project. This package is a layer between the application program (e.g. PAW/HPLOT) and the basic graphics package (e.g. X11) on a given system. Its basic aims are:

- Full transportability of the picture data base.
- Easy manipulation of the picture elements.
- Compactness of the data to be transported and accessibility of the pictures in direct access mode.

– Independence of the underlying basic graphics package. Presently HIGZ is interfaced with several GKS packages, X- Windows (X11), PHIGS, Mac, PC's graphic systems, GL (Silicon Graphics), GDDM (IBM), GPR (Apollo) as well as with the DI3000 system. Note that some of these graphics systems are now obsolete. PAW is now mainly used in its X11 version.

These requirements have been incorporated into HIGZ by exploiting the data management system ZEBRA.

HIGZ does not introduce new basic graphics features, but introduces some macroprimitives for frequently used functions (e.g. arcs, axes, boxes, pie-charts, tables). The system provides the following features:

– Basic graphics functions: basic primitives, attributes, space definition.
– Higher-level macroprimitives.
– Data structure management using an interface to the ZEBRA system.
– Interactive picture editing.

These features, which are available simultaneously, are particularly useful during an interactive session, as the user is able to "replay" and edit previously created pictures, without the need to re-run the application program. A direct interface to PostScript is also available.

1.6.4 ZEBRA - The data structure management system

The data structure management package ZEBRA was developed at CERN in order to overcome the lack of dynamic data structure facilities in FORTRAN, the favourite computer language in high energy physics. It implements the **dynamic creation and modification** of data structures at execution time and their transport to and from external media on the same or different computers, memory to memory, to disk or over the network, at an **insignificant cost** in terms of execution-time overheads.

ZEBRA manages any type of structure, but specifically supports linear structures (lists) and trees. ZEBRA input/output is either of a sequential or direct access type. Two data representations, **native** (no data conversion when transferred to/from the external medium) and **exchange** (a conversion to an interchange format is made), allow data to be transported between computers of the same and of different architectures. The direct access package **RZ** can be used to manage hierarchical data bases. In PAW this facility is exploited to store histograms, Ntuples and pictures in a hierarchical direct access directory structure.

1.6.5 MINUIT - Function minimization and error analysis

MINUIT is a tool to find the **minima of a multi-parameter function** and analyse the **shape around the minimum**. It can be used for **statistical analysis** of curve fitting, working on a χ^2 or log-likelihood function, to compute the **best fit** parameter values, their uncertainties and correlations. **Guidance** can be provided in order to find the correct solution, parameters can be kept fixed and data points can be easily added or removed from the fit. An interactive Motif based interface is in preparation.

1.6.6 COMIS - The FORTRAN interpreter

The COMIS interpreter allows the user to execute interactively a set of FORTRAN routines in interpretive mode. The interpreter implements a large subset of the complete FORTRAN language. It is an extremely important tool because it allows the user to specify his own complex data analysis procedures, for example selection criteria or a minimisation function.

1.6.7 SIGMA - The array manipulation language

A scientific computing programming language SIGMA (**S**ystem for **I**nteractive **G**raphical **M**athematical **A**pplications), which was designed essentially for mathematicians and theoretical physicists is integrated into PAW. Its main characteristics are:

– The basic data units are scalars and one or more dimensional rectangular arrays, which are automatically handled.
– The computational operators resemble those of FORTRAN.

1.7 A PAW Glossary

Data Analysis Terminology

DST A "Data Summary Tape" is one basic form of output from a typical physics experiment. A DST is generally not used directly by PAW, but is analyzed by customized user programs to produce Ntuple files, which PAW can read directly.

Ntuple A list of identical data structures, each typically corresponding to a single experimental event. The data structures themselves frequently consist of a row of numbers, so that many Ntuples may be viewed as two-dimensional arrays of data variables, with one index of the array describing the position of the data structure in the list (i.e., the row or event number), and the other index referring to the position of the data variable in the row (i.e., the column or variable number). A meaningful name is customarily assigned to each column that describes the variable contained in that column for each event.

Event A single instance of a set of data or experimental measurements, usually consisting of a sequence of variables or structures of variables resulting from a partial analysis of the raw data. In PAW applications, one typically examines the statistical characteristics of large sequences of similar events.

Variable One of a user-defined set of named values associated with a single event in an Ntuple. For example, the (x, y, z) values of a momentum vector could each be variables for a given event. Variables are typically useful experimental quantities that are stored in an Ntuple; they are used in algebraic formulas to define boolean cut criteria or other dependent variables that are relevant to the analysis.

Cut A boolean-valued function of the variables of a given event. Such functions allow the user to specify that only events meeting certain criteria are to be included in a given distribution.

Mask A set of columns of zeros and ones that is identical in form to a new set of Ntuple variables. A mask is typically used to save the results of applying a set of cuts to a large set of events so that time-consuming selection computations are not repeated needlessly.

Function Sequence of one or more statements with a FORTRAN-like syntax entered on the command line or via an external file.

Statistical Analysis Terminology

Histogram A one- or two-dimensional array of data, generated by HBOOK in batch or in a PAW session. Histograms are (implicitly or explicitly) declared (booked); they can be filled by explicit entry of data or can be derived from other histograms. The information stored with a histogram includes a title, binning and packing definitions, bin contents and errors, statistic values, possibly an associated function vector, and output attributes. Some of these items are optional. The ensemble of this information constitutes an **histogram**.

Booking The operation of declaring (creating) an histogram.

Filling The operation of entering data values into a given histogram.

Fitting Least squares and maximum likelihood fits of parametric functions to histograms and vectors.

Projection The operation of projecting two-dimensional distributions onto either or both axes.

Band A band is a projection onto the X (or Y) axis restricted to an interval along the other Y (or X) axis.

Slice A slice is a projection onto the X (or Y) axis restricted to one bin along the other Y (or X) axis. Hence a slice is a special case of a band, with the interval limited to one bin.

Weight PAW allows the user to include a multiplicative statistical bias for each event which is a scalar function of the available variables. This permits the user to correct for known statistical biases in the data when making histograms of event distributions.

KUIP/ZEBRA User Environment Terminology

Macro A text file containing a set commands and logical constructs to control the flow of execution. Parameters can be supplied when calling a macro.

Vector The equivalent of a FORTRAN array supporting up to three dimensions. The elements of a vector can be stored using a real or an integer representation; they can be entered interactively on a terminal or read from an external file.

Logical Directory The ZEBRA data storage system resembles a file system organized as logical directories. PAW maintains a global variable corresponding to the "current directory" where PAW applications will look for PAW objects such as histograms. The ZEBRA directory structure is a tree, and user functions permit the "current directory" to be set anywhere in the current tree, as well as creating new "directories" where the results of PAW actions can be stored. A special directory called `//PAWC` corresponds to a memory-resident branch of this virtual file system. ZEBRA files may be written to the operating system file system, but entire hierarchies of ZEBRA directories typically are contained in a single binary operating system file.

Graphics Production Terminology

Metafile A file containing graphical information stored in a device independent format, which can be replayed on various types of output devices. (e.g. PostScript).

Picture A graphics object composed of graphics primitives and attributes. Pictures are generated by the HIGZ graphics interface and they can be stored in a picture direct-access database, built with the RZ-package of the data structure manager ZEBRA.

PostScript A high level page description language permitting the description of complex text and graphics using only text commands. Using PostScript representations of graphics makes it possible to create graphics files that can be exchanged with other users and printed on a wide variety of printers without regard to the computer system upon which the graphics were produced. Any graphics display produced by PAW can be expressed in terms of PostScript, written to a file, and printed.

Chapter 2: General principles

2.1 Access to PAW

At CERN the PAW program is interfaced on all systems via a command procedure which gives access to the three release levels of the CERN Program Library (PROduction, OLD and the NEW areas) and sets the proper environment if necessary. Users who are not at CERN or who are using non-central computer systems should contact their system administrator for help on PAW.

2.1.1 VAX/VMS

A command file `CERN_ROOT:[EXE]PAW.COM` is defined system-wide via the logical symbol `PAW`; its interface is:

`PAW/ver` (the default is `PRO`)

You may set the initialization of PAW either as a `PAWLOGON.KUMAC` located in your home directory, or through the logical symbol `DEFINE PAW$LOGON disk:[user.subdir]file.kumac` to be defined usually in your `LOGIN.COM`.

2.1.2 Unix systems

The driver shell script is located in the file `/cern/pro/bin/paw`. In order to access it automatically you could add the directory `/cern/pro/bin` to your command search path. The command syntax is:

`paw -v ver` (the default is `-v PRO`)

2.1.3 Workstation type

PAW needs to know the X-host where graphics must be displayed; this can be specified on each system on the command line:

```
Vax/VMS:   PAW/X11/host=yourhost
Unix:      paw -d X11 -h yourhost
```

or at the "`Workstation`" prompt in PAW: `Workstation type (?=HELP) [CR]=1 :  1.yourhost`
If `yourhost` is not specified, the output is redirected (like for all X11 applications) to the display defined via the environment variable `DISPLAY`.
The workstation type selects which type of workstation has to be opened. It corresponds to a line number in a file `higz_windows.dat`. PAW tries to open this file in your current working directory. If it does not succeed it tries in your HOME directory. If it doesn't succeed once more, it creates the file in your HOME directory as follows:

```
0000 0000 0600 0600
         .
         .
         .
0000 0000 0600 0600
```

where the lines define each of the workstation types (from 1 to 10) with the x-margin (left), y-margin (top), x-size (width) and y-size (height) of the corresponding window in pixels.
For a more complete and up to date description you can refer to the PAW FAQs avaialable from the PAW web home page.

2.1.4 Different modes to start PAW

- A **batch** version of PAW is available (note that batch implies workstation type 0):

```
On Unix   do:  paw -b macroname
On VMS    do:  PAW/BATCH=macroname
```

- One can **disable** the automatic execution of the `PAWLOGON` macro:

```
On Unix   do:  paw -n
On VMS    do:  PAW/NOLOG
```

2.2 Initialising PAW

When PAW is started, a **system** startup procedure is initiated, which indicates the current version of PAW and requests the **workstation type** of the terminal or workstation which you are using.

```
$ PAW
 ******************************************************
 *                                                    *
 *            W E L C O M E    to   P A W             *
 *                                                    *
 *        Version 2.10/01       2 September 1998      *
 *                                                    *
 ******************************************************
 Workstation type (?=HELP) <CR>=1 : ?

 List of valid workstation types:
       0:  Alphanumeric terminal
    1-10:  Describe in file higz_windows.dat
  n.host:  Open the display on host (1 < n < 10)
    7878:  FALCO terminal
    7879:  xterm
```

Note that if you specify 0, PAW will not open a graphics workstation. This may be appropriate if one wants to use PAW on an alphanumeric terminal.

Before passing control to the user, the system looks for a user-supplied file `pawlogon.kumac`. The latter can contain commands which the user wants to be executed at PAW startup, e.g. declaration of files, creation of aliases, definition of HPLOT parameters. A simple version of this PAW initialisation file, displaying date and time, can be:

```
mess '******************************************************'
mess '*                                                    *'
mess '*   Starting PAW session on '//$date//' at '//$time//'     *'
mess '*                                                    *'
mess '******************************************************'
```

In order to only have one version of this file on VAX/VMS the user should define a **logical name** `PAW$LOGON` in his `LOGIN.COM`, as explained on the previous page. The file `pawlogon.kumac` is taken in the current directory.

2.3 Command structure

PAW is based on the KUIP[4] User Interface package, which can provide different types of dialogue styles:

- Command mode, where the user enters a command line via the terminal keyboard.
- Alphanumeric menu mode, where the command is selected from a list.
- Graphics menu modes:
  - Pull-down menus, fixed layout reflecting the command structure;
  - Panels of function keys, interactive user definable multiple layouts.

It is possible to change interactively from one style to another.

The general format of a PAW command line is:

```
command parameters
```

The first part of the **command** has the format:

```
object/verb
```

where the **object** is the item on which the action is performed (e.g. HISTOGRAM, VECTOR, NTUPLE) and the verb is the action to be performed (e.g. CREATE, DELETE, PLOT). In some cases the object needs to be specified further (e.g. GRAPHICS/PRIMITIVE), while in other cases the verb's action needs to be clarified further (e.g. CREATE/1D). All components can be **abbreviated** to their shortest unambiguous form. For example the two following lines will have the same effect of creating a vector A with nine components:

```
VECTOR/CREATE A(9)
```

or

```
VE/CR A(9)
```

In the case that the form is ambiguous all possible interpretations for the given abbreviation are displayed.

The second part of a command are its **parameters** and their meaning is determined by their **position**. Some of these can be **mandatory** with the remaining ones **optional**. If all mandatory parameters are not provided on the command line, PAW will prompt the user to specify them, indicating the default values if defined. If the user wants to assign the default value to a parameter from the command line he can use the **place-holder** character **exclamation mark (!)** to signify this to PAW. In the case of optional parameters, the user **must** provide them in the correct sequence if he wants to **change** their values, otherwise the corresponding defaults are taken. Parameters containing blanks must be enclosed within single quotes.

In the example below we create a one-dimensional histogram, providing the parameters one by one answering the PAW query:

```
PAW > histogram/create/1dhisto
Histogram Identifier (<CR>= ): 10
Histogram title (<CR>= ): title1
Number of channels (<CR>=100): <CR>
Low edge (<CR>=0): 10.
Upper edge (<CR>=100): 20.
```

For the command below we provide all parameters on the command line, including an optional one (1000.), which by default has the value 0. Note that this parameter **must** be specified explicitly, since PAW **does not** prompt for it, as seen in the previous example. Note also the use of the exclamation mark to take the default for the number of channels (100).

```
PAW > hi/cr/1d 20 title2 ! 10. 20. 1000.
```

2.4 Getting help

Once inside PAW, one can start entering commands. An interesting first try would be the HELP command, which displays a list of items, preceded by a number and followed by one line of explanation. In the next example we search for a command to create a one-dimensional histogram.

```
      PAW > help

From  /...

 1:   KUIP          Command Processor commands.
 2:   MACRO         Macro Processor commands.
 3:   VECTOR        Vector Processor commands.
 4:   HISTOGRAM     Manipulation of histograms, Ntuples.
 5:   FUNCTION      Operations with Functions. Creation and plotting.
 6:   NTUPLE        Ntuple creation and related operations.
 7:   GRAPHICS      Interface to the graphics packages HPLOT and HIGZ.
 8:   PICTURE       Creation and manipulation of HIGZ pictures.
 9:   ZEBRA         Interfaces to the ZEBRA RZ, FZ and DZ packages.
10:   FORTRAN       Interface to MINUIT, COMIS, SIGMA and FORTRAN
                    Input/Output.
11:   NETWORK       To access files on remote computers.
12:   OBSOLETE      Obsolete commands.

Enter a number ('Q'=command mode): 4

   /HISTOGRAM
```

```
   Manipulation of histograms, Ntuples.  Interface to the HBOOK package.


From  /HISTOGRAM/...

 1: * FILE          Open an HBOOK direct access file.
 2: * LIST          List histograms and Ntuples in the current directory.
 3: * DELETE        Delete histogram/Ntuple ID in Current Directory
                    (memory).
 4: * PLOT          Plot a single histogram or a 2-Dim projection.
 5: * ZOOM          Plot a single histogram between channels ICMIN and
                    ICMAX.
 6: * MANY_PLOTS    Plot one or several histograms into the same plot.
 7: * PROJECT       Fill all booked projections of a 2-Dim histogram.
 8: * COPY          Copy a histogram (not Ntuple) onto another one.
 9: * FIT           Fit a user defined (and parameter dependent) function
                    to a histogram ID (1-Dim or 2-Dim) in the specified
                    range.
10:   2D_PLOT       Plotting of 2-Dim histograms in various formats.
11:   CREATE        Creation ("booking") of HBOOK objects in memory.
12:   HIO           Input/Output operations of histograms.
13:   OPERATIONS    Histogram operations and comparisons.
14:   GET_VECT      Fill a vector from values stored in HBOOK objects.
15:   PUT_VECT      Replace histogram contents with values in a vector.
16:   SET           Set histogram attributes.

Enter a number ('\'=one level back, 'Q'=command mode): 11

   /HISTOGRAM/CREATE

   Creation ("booking") of HBOOK objects in memory.


From  /HISTOGRAM/CREATE/...

 1: * 1DHISTO       Create a one dimensional histogram.
 2: * PROFILE       Create a profile histogram.
 3: * BINS          Create a histogram with variable size bins.
 4: * 2DHISTO       Create a two dimensional histogram.
 5: * PROX          Create the projection onto the x axis.
 6: * PROY          Create the projection onto the y axis.
 7: * SLIX          Create projections onto the x axis, in y-slices.
 8: * SLIY          Create projections onto the y axis, in x-slices.
 9: * BANX          Create a projection onto the x axis, in a band of y.
10: * BANY          Create a projection onto the y axis, in a band of x.
11: * TITLE_GLOBAL  Set the global title.

Enter a number ('\'=one level back, 'Q'=command mode): 1

 * /HISTOGRAM/CREATE/1DHISTO  ID TITLE NCX XMIN XMAX [  VALMAX ]

   ID         C 'Histogram Identifier' Loop
   TITLE      C 'Histogram title' D=' '
   NCX        I 'Number of channels' D=100
   XMIN       R 'Low edge' D=0.
   XMAX       R 'Upper edge' D=100.
   VALMAX     R 'Maximum bin content' D=0.

   Create a one dimensional histogram.  The contents are set to zero.  If
   VALMAX=0, then a full word is allocated per channel, else VALMAX is used
   as the maximum bin content allowing several channels to be stored into
   the same machine word.

<CR>=continue, 'Q'=command mode, 'X'=execute: q
```

An item preceded by a **star** indicates a **terminal leaf** in the command tree, i.e. an **executable** command. One can also inquire about **creating a one-dimensional histogram** by typing simply:

```
HELP histogram/create/1dhisto
```

or

```
    HELP his/cre/1d
```
or even
```
    HELP 1
```

The system will then display the following information:

```
 * /HISTOGRAM/CREATE/1DHISTO  ID TITLE NCX XMIN XMAX [  VALMAX ]

  ID         C 'Histogram Identifier' Loop
  TITLE      C 'Histogram title' D=' '
  NCX        I 'Number of channels' D=100
  XMIN       R 'Low edge' D=0.
  XMAX       R 'Upper edge' D=100.
  VALMAX     R 'Maximum bin content' D=0.

  Create a one dimensional histogram.  The contents are set to zero.  If
  VALMAX=0, then a full word is allocated per channel, else VALMAX is used
  as the maximum bin content allowing several channels to be stored into
  the same machine word.
```

2.4.1 Usage

Very often a single line description of the usage of a command is sufficient as a reminder. This can be obtained by the USAGE command, e.g.:

```
    PAW > USAGE 1d

    * /HISTOGRAM/CREATE/1DHISTO  ID TITLE NCX XMIN XMAX [  VALMAX ]
```

2.5 Special symbols for PAW

One should pay attention to the fact that, in addition to their common arithmetic meaning, the symbols in table 2.1 have a special connotation when working with PAW .

<table>
<tr><th>Symbol</th><th>Meaning</th></tr>
<tr><td><code>blank</code></td><td>Separator between command and parameter and between different parameters</td></tr>
<tr><td><code>/</code></td><td>Separator between command elements</td></tr>
<tr><td></td><td>Comment line (if first character of the command line)</td></tr>
<tr><td><code>|</code></td><td>Inline comments</td></tr>
<tr><td><code>'</code></td><td>String delimiter</td></tr>
<tr><td><code>_</code></td><td>Line continuation in KUIP commands</td></tr>
<tr><td><code>@</code></td><td>Escape character to be put in front of <code>|</code> and <code>'</code> to interpret them as literal</td></tr>
<tr><td><code>!</code></td><td>Place-holder for command parameter (i.e. default value is taken)</td></tr>
<tr><td></td><td>At beginning of command line: Unix C shell-like history</td></tr>
<tr><td></td><td>(e.g. <code>!!, !number, !-number, !string</code>)</td></tr>
<tr><td><code>[]</code></td><td>Macro argument delimiters</td></tr>
<tr><td><code>#</code></td><td>Separator between macro file and macro member</td></tr>
<tr><td><code>( )</code></td><td>Vector subscript delimiters</td></tr>
<tr><td><code>:</code></td><td>Vector subscript range</td></tr>
<tr><td><code>,</code></td><td>Multi-dimensional vector subscript dimensions delimiter</td></tr>
<tr><td colspan="2">Note: These special characters loose their effect when imbedded in single quotes.</td></tr>
</table>

Table 2.1: Special symbols

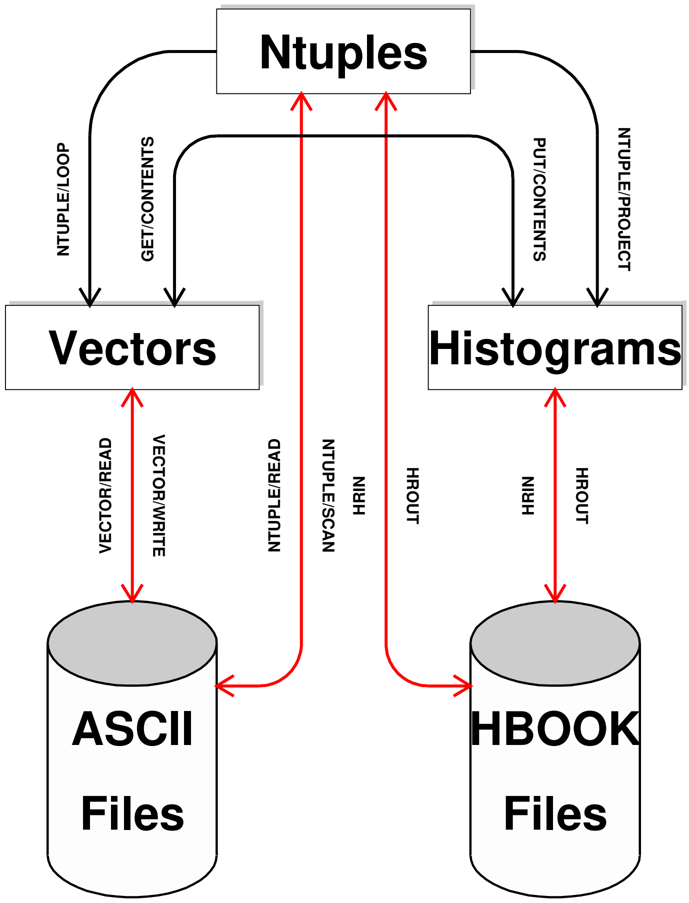


Figure 2.1: PAW entities and their related commands

2.6 PAW entities and their related commands

Relations which exist between various PAW entities as described in section 1.6 on page 6 and the operations which can be performed upon them have been schematically represented in figure 2.1. All commands shown in the picture next to the lines connecting the objects have been abbreviated in a way that they are unambiguous and can be typed to PAW, which will then detail the various parameters to be supplied.

There are three main input/output formats, namely a simple text file (e.g. with data points or commands), a direct access ZEBRA RZ file (used by HBOOK and HIGZ for storing histograms and pictures on a given machine) and a ZEBRA FZ sequential file, which can be used to transfer structured ZEBRA data between various computers. The RZ and FZ representations can be transformed into each other using the TOALFA and FRALFA commands.

The three main PAW objects, Ntuples, histograms and vectors, can be **printed** on an alphanumeric screen (PRINT commands) or they can be plotted on a graphics screen (PLOT commands). The picture can be transformed into a ZEBRA data structure and stored in a HIGZ database for later reference (e.g. editing by the HIGZ editor), or an external presentation can be obtained via the creation of a **metafile**.

Chapter 3: User interface - KUIP

3.1 Command line syntax

The general syntax of a *command line* is a *command path* optionally followed by an *argument list*. The command path and the arguments have to be separated from each other by one or more space characters. Therefore arguments containing spaces or other special characters have to be quoted.

In the following we want to use an appropriate formalism to describe the syntax rules. The notation will be introduced step by step as needed. The verbal explanation given above can be written as:

command-line `::=` *command-path* { *argument* }

The *slanted* symbols are non-terminal, i.e. they are composed of other terminal or non-terminal symbols. The definition of a non-terminal symbol is denoted by "`::=`". Symbols enclosed in braces ("`{...}`") are optional and they can appear zero or more times.

3.1.1 Command structure

The set of commands is structured as an (inverted) tree as shown in figure 3.1.

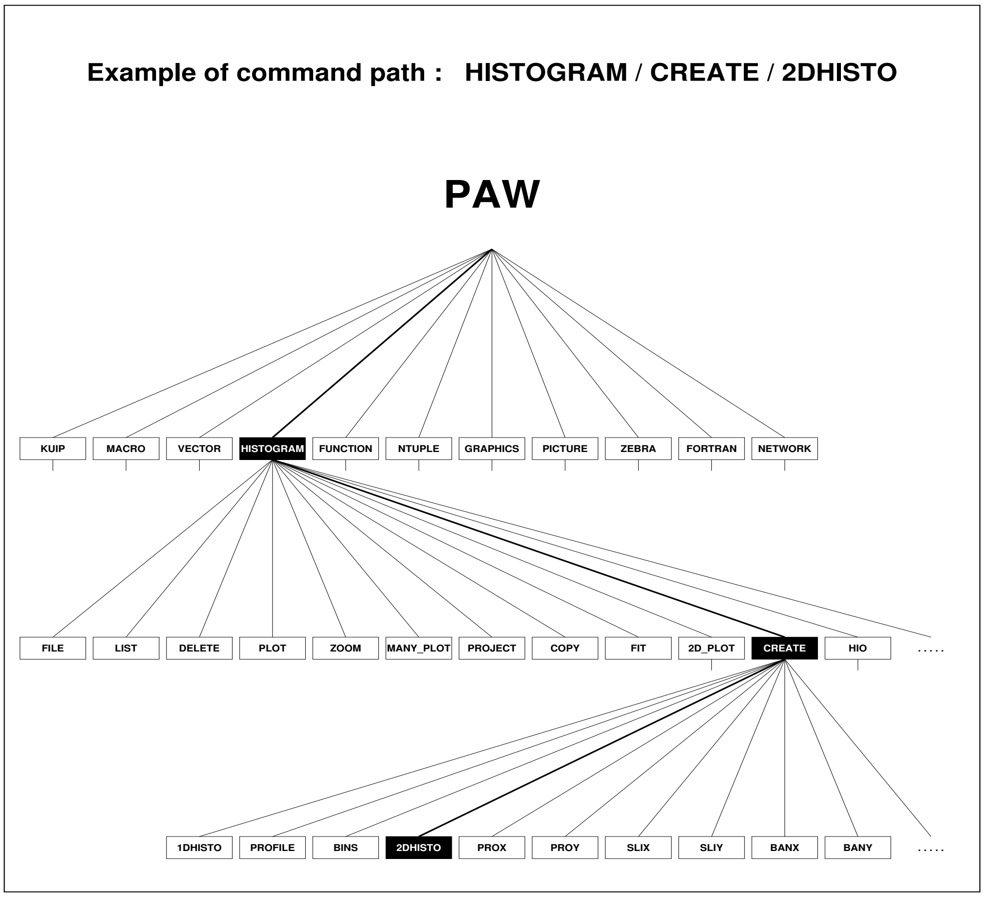


Figure 3.1: Example of the PAW command tree structure

This structure is comparable to a Unix file system. The command set can be dynamically extended by linking new commands or menus into the tree. Compared to a flat list structure the tree allows a cleaner representation through menus, especially when the command set is large. `paw` has more than 200 commands. It would be hard to visualize such a number of command in a single graphics menu.

Abbreviations

A command path consists of a menu path and a command name. The menu path itself consists of a list of menu names up to an arbitrarily deep level of sub-menus.

command-path	`::=`	`[`*menu-path*`/]`*command-name*
menu-path	`::=`	`[/]`*menu-name*`{/`*menu-name*`}`

Here we introduced two more notations. Symbols in teletype mode ("`/`") are literals, i.e. the menu and command names have to be separated by a slash character. Symbols enclosed in brackets ("`[...]`") are optional which can appear zero or one times.

These syntax rules already show that a command path may be abbreviated by omitting part of the leading menu path. For example, if the complete command path is

```
/MENU/SUBMENU/COMMAND
```

valid abbreviations are

```
MENU/SUBMENU/COMMAND
SUBMENU/COMMAND
COMMAND
```

but **not** "`MENU/COMMAND`" or "`/SUBMENU/COMMAND`". Note that the command name matching is case-insensitive, i.e. the following are all valid possibilities:

```
COMMAND
command
Command
```

Furthermore, menu and command names may be abbreviated by omitting trailing parts, i.e.

```
SUB/COMMAND
COMMA
/M/S/C
```

are also valid abbreviations.

The shortest unambiguous abbreviation for any command is not fixed but depends on the whole command set. PAW lists all possible ambiguities if a given abbreviation has no unique match:

```
PAW > LIST
 *** Ambiguous command list. Possible commands are :

 /KUIP/ALIAS/LIST
 /MACRO/LIST
 /VECTOR/LIST
 /HISTOGRAM/LIST
 /NTUPLE/LIST
 /PICTURE/LIST
```

Changing the root menu The command `SET/ROOT` defines the menu from which the search for command name starts. It is not quite comparable to the Unix `cd` or VMS `SET DEFAULT` command. If no matching command is found going downwards from the `SET/ROOT` menu a second attempt is made starting off at the top menu "`/`".

Disabling commands The command `SET/VISIBILITY` allows to disable/enable individual commands. Disabled commands cannot be executed and they do not contribute to name ambiguities. However, the `HELP` information is still available. Note that the `VISIBILITY` command can disable itself which makes it impossible to re-enable any command.

Automatic macro execution The command MACRO/DEFAULT implements two facilities. First it allows to define a directory search path used by the EXEC command for locating .kumac macro files. Second it controls the implicit interpretation of the command name token as a possible macro filename:

-Command This is the default setting which does not try to interpreted cmd as macro name.

-Auto If the search path contains a file cmd.kumac it is executed, i.e. the actual command becomes "EXEC cmd", otherwise the search for a command named cmd starts.

-AutoReverse If cmd is either not a command name or ambiguous and a file cmd.kumac exists the command is transformed into "EXEC cmd".

Command template The command SET/COMMAND allows to define a template which is used whenever the command token does not match any command name. The template can contain "$1",..., "$9" which are substituted with the *n*'th token from the original command line, or "$*" which is replaced by the complete line. For example, PAW can be turned into a calculator by

```
PAW > SET/COMMAND 'mess $sigma($*)'
PAW > 17+2*5
 27
```

"SET/COMMAND 'EXEC $*'" has almost the same effect as "DEFAULT -AutoReverse" but these are two distinct facilities which can be active simultaneously. The difference is that for SET/COMMAND the token in the command name position must not match any command. If does not apply if the token is an ambiguous command name.

Both Auto/AutoReverse and SET/COMMAND logic are ignored during the execution of macro scripts.

3.1.2 Arguments

Most commands have *parameters* for which the user is expected to supply *argument values*. Parameters are either *mandatory* or *optional*. Mandatory arguments which are not specified on the command line are prompted for. If optional arguments are omitted a default value is used instead.

Mandatory parameters always precede the optional parameters. The command USAGE allows to see the number of parameters for a command:

```
PAW > usage manual

 * KUIP/MANUAL ITEM [ OUTPUT OPTION ]
```

The optional parameters are enclosed in square brackets. The default values can be seen from the help text for a command. The STYLE command shown in figure 3.2 has only optional arguments. The corresponding default values are indicated in the help information as "D=*value*".

Mandatory parameters may also have a default value which is used if the prompt is acknowledged by simple hitting the RETURN-key. Otherwise the proposed default is the value used in the previous command execution.

The STYLE command also shows that there are three different kind of parameters: character values indicated by "C" after the parameter name, real values ("R") and integer values ("I").

Numeric (real or integer) parameters may be restricted in the range of acceptable values. In the help text this is indicated as "R=*lower*:*upper*. If the argument value is outside the range PAW prompts the user to enter an acceptable value before the command can be executed. The lower or upper range value may be missing to indicate an unlimited range in one direction. Instead of a simple numeric value the argument may also be an expression.

For both numeric and character parameters the range may also be given as a comma-separated list of values. PAW will accept an argument only if it matches one of the values in the list.

In general the arguments given on the command line are assigned to the command parameters from left to right but there are also ways to change the order. In our syntax notation, using "|" to indicate possible alternatives, we can write:

argument ::= *value* | ! | !! | *name*=*value* | -*value*

An argument given as a simple value is assigned to the next parameter expected. The special values "!" and "!!" are templates for the default value and the value from the previous command execution, respectively.

```
PAW > HELP STYLE

 * KUIP/SET_SHOW/STYLE [ OPTION SGYLEN SGSIZE SGYSPA SGBORD WKTYPE ]

   OPTION     C 'Option' D='?'
   SGYLEN     R 'max Y LENgth of each menu item box' D=0.025 R=0.005:0.25
   SGSIZE     R 'space available for the application' D=0.8 R=0:0.90
   SGYSPA     R 'max Y length of space between menus' D=0.02 R=-0.5:0.50
   SGBORD     R 'X or Y border for menus' D=0.015 R=0:0.25
   WKTYPE     I 'Graphics workstation type' D=0

   Possible OPTION values are:

    ?   show current style
    C   Command line : select Command line input
    AN  Menu with Numbers : select general Alpha menu (with Numbers)
    AL  Menu with Letters : select general Alpha menu (with Letters)
```

Figure 3.2: Parameter types, default values, and range limits

Named arguments

The form "*name=value*" allows to invert the argument order or to skip a list of optional parameters for which the default values should be used. For example,

```
STYLE G SGBORD=0.1
```

is equivalent to

```
STYLE G ! ! ! 0.1
```

A simple argument following a named argument is assigned to the parameter following the named parameter, i.e.

```
STYLE G SGBORD=0.1 1
```

is equivalent to

```
STYLE G ! ! ! SGBORD=0.1 WKTYPE=1
```

Parameter names are case-insensitive but in general they may not be abbreviated. In the help text the abbreviat level is indicated by a "*" inside the parameter name. For example, if the parameter name is shown as

```
LIB*RARY
```

the acceptable abbreviations are "`LIB=`", "`LIBR=`", "`LIBRA=`", "`LIBRAR=`", and "`LIBRARY=`".

PAW does not insist that an argument of the form "*name=value*" matches one of the parameter names. The argument including the "*name=*" part is simply assigned to the next parameter expected.

Option arguments

The last alternative "-*value*" to specify an argument applies only to *option* parameters. (Note the distinction between *option* and *optional*. Option parameters are usually but not necessarily optional.) In the help text option parameters are tagged by the list of possible values (figure 3.3). Frequently these parameters are named "`OPTION`" or "`CHOPT`".

The "-*value*" form allows to specify option arguments out of order, emulating the Unix style of options preceded other command arguments. For example,

```
MANUAL -LATEX /KUIP
```

```
PAW > HELP MANUAL

 * KUIP/MANUAL ITEM [ OUTPUT OPTION ]

   ITEM       C 'Command or menu path'
   OUTPUT     C 'Output file name' D=' '
   OPTION     C 'Text formatting system' D=' '

   Possible OPTION values are:

   ' '     plain text : plain text format
    LATEX  LaTeX format (encapsulated)
    TEX    LaTeX format (without header)
```

Figure 3.3: Example for option parameters

is equivalent to

```
MANUAL /KUIP OPTION=LATEX
```

Note that this is **not** equivalent to "`MANUAL OPTION=LATEX /KUIP`". Unlike to the "`-`*value*" form subsequent simple arguments are still assigned to the next parameter expected, not to the one following the option parameter itself.

Since a leading "`-`" can be part of a valid (non-option) argument the value is checked against a set of rules before it is actually interpreted as an option assignment.

The option argument can be a concatenation of several of the allowed option values. PAW checks that the argument string is exclusively constructed from valid option values. This check is done by removing matches of option values from the argument string, starting with the longest option values first. For example, with the definition

```
   Possible OPTION values are:
    AB
    ABC
    CD
```

the argument "`-ABCD`" is not interpreted as option assignment because after removing the longest match "`ABC`" the remainder "D" is not anymore a valid option value. (This case would have to be written as "`-CDAB`".

Argument values

Since in command line blanks are used to separate the command name and the individual arguments string values containing blanks have to be quoted. The rules are the same as used by Fortran: the quote character is the apostrophe "`'`", and apostroph inside a quoted string have to be duplicated:

```
MESS 'Hello world'
MESS 'Do or don''t'
```

Note that the `MESSAGE` command has only a single parameter:

```
 * KUIP/MESSAGE [ STRING ]

   STRING     C 'Message string' D=' '
...
```

Nevertheless, in most cases quoting the message string is not necessary. If the command line contains more arguments than there are parameters the additional values are concatenated to the argument for the last parameter. In the concatenation each value is separated by a (single) blank character, i.e. the commands

```
MESS 'Hello World'
MESS  Hello World
MESS  Hello      World
```

yield all the same output. Therefore the message text only needs quoting if the words should be separated by more than one space character.
Quoting inhibits the interpretation of the enclosed string as special argument values. Printing an exclamation mark as message text has to written as

```
MESS '!'
```

because “`MESS !`” would mean to take the default value for the parameter `STRING` and yield an empty line only.
Another instance is if an argument of the form “*name=value*” should be taken literally. For example, the command line

```
EXEC mac foo=bar
```

initializes the macro variable “`foo`” to the value “`bar`”. However, if the intention is to pass the string “`foo=bar`” as argument to the macro quotes must be used:

```
EXEC mac 'foo=bar'
```

In addition, some commands, e.g.

```
 * NTUPLE/PLOT IDN [ UWFUNC NEVENT IFIRST NUPD OPTION IDH ]
```

use the form “*name=value*” for equality tests in the cut expression `UWFUNC`. For example, the command

```
NT/PLOT 10.energy year=1998
```

selects all event for which the Ntuple column `YEAR` has the value `1998`. Any name clash between the Ntuple column and one of the command parameters requires quoting. If the column was called `NUPD` instead of `YEAR` the command would have to be written as

```
NT/PLOT 10.energy 'nupd=1998'
```

or alternatively as “`NT/PLOT 10.energy UWFUNC=nupd=1998`”.
Finally, quoted strings are also exempted from any substitutions of aliases, system functions, and macro variables. For example,

```
MESS 'foo'
```

always prints “`foo`” while

```
MESS foo
```

can result in “`bar`” if preceded by the command “`ALIAS/CREATE foo bar`”. Since square brackets denote macro variable substitution and system functions names start with a dollar-sign it is especially recommended to quote VMS file specifications.
The operator “`//`” allows to concatenate several parts to a single argument value. Unquoted strings on either side of the concatenation operator are implicitly treated as literals unless they are subject to a substitution, i.e. the command lines

```
MESS 'abc'//'def'
MESS 'abc'//def
MESS abc//'def'
MESS abc//def
MESS abcdef
MESS 'a'//'b'//'c'//'d'//'e'//'f'
```

are all equivalent (provided that `abc` and `def` are not defined as aliases). The character sequence “`//`” at the beginning or end of an argument is taken literally, e.g. in

```
CD //LUN2//1
```

the command receives the value “`//LUN21`”.

3.1.3 More on command lines

The command line syntax allows to write several commands in one line and also to extend commands with long argument lists over several lines.

Multiple commands on a single line

An input line presented to the PAW command processor may contain several commands separated by "`;`". The commands are executed sequentially as if they were on separate lines:

```
MESS Hello world!; MESS How are you?
```

is equivalent to

```
MESS Hello world!
MESS How are you?
```

Note that the text following the semicolon will not be used to satisfy any prompts emitted by the preceding command, e.g. "`usage; manual`" will not behave as "`usage manual`".

The semicolon is **not** interpreted as line separator if it is immediately followed by a digit or one of the characters

```
  +  - *  ?  [
```

For example, issuing a VMS command with a file version number such as

```
SHELL delete *.tmp;*
```

does not require quoting. Note that this exception rule applies independently of the operating system. In order to avoid surprises we recommend to put always at least one blank after a semicolon intended to be a line separator.

Each command execution returns a status code which is zero for success and non-zero for failure. The sequences "`;&`" and "`;!`" allow to execute the remaining part of an input line depending on the status code of the preceding command. With

```
cmd1 ;& cmd2 ; cmd3
```

the commands `cmd2` and `cmd3` are only executed if `cmd1` succeeded while with

```
cmd1 ;! cmd2 ; cmd3
```

the remaining commands are only executed if the first one failed. Note that the two characters must follow each other immediately without intervening blank.

In some commands, for example `HISTO/PLOT`, one of the parameters is marked in the help text with the attribute "`Loop`". If the corresponding argument is a comma-separated list of values PAW implicitly repeats the command for each value in the list individually:

```
HISTO/PLOT 10,20,30
```

is equivalent to

```
HISTO/PLOT 10
HISTO/PLOT 20
HISTO/PLOT 30
```

Note that "`,`" inside parentheses is not taken as value separator, i.e.

```
HISTO/PLOT 10(1:25,1:25)
```

executes a single command.

`^A/^E`	Move cursor to beginning/end of the line.
`^F/^B`	Move cursor forward/backward one character.
`^D`	Delete the character under the cursor.
`^H, DEL`	Delete the character to the left of the cursor.
`^K`	Kill from the cursor to the end of line.
`^L`	Redraw current line.
`^O`	Toggle overwrite/insert mode. Text added in overwrite mode (including yanks) overwrites existing text, while insert mode does not overwrite.
`^P/^N`	Move to previous/next item on history list.
`^R/^S`	Perform incremental reverse/forward search for string on the history list. Typing normal characters adds to the current search string and searches for a match. Typing `^R/^S` marks the start of a new search, and moves on to the next match. Typing `^H` or `DEL` deletes the last character from the search string, and searches from the starting location of the last search. Therefore, repeated `DEL`'s appear to unwind to the match nearest the point at which the last `^R` or `^S` was typed. If `DEL` is repeated until the search string is empty the search location begins from the start of the history list. Typing `ESC` or any other editing character accepts the current match and loads it into the buffer, terminating the search.
`^T`	Toggle the characters under and to the left of the cursor.
`^U`	Kill from the prompt to the end of line.
`^Y`	Yank previously killed text back at current location. Note that this will overwrite or insert, depending on the current mode.
`TAB`	By default adds spaces to buffer to get to next `TAB` stop (just after every 8th column).
`LF, CR`	Returns current buffer to the program.

Table 3.1: Key-binding for recall style `KSH`

Single commands on multiple lines

For commands with very long argument lists it can become necessary to continue it on the next line. An input line ending with an "`_`" character is joined with the following line.

In the concatenation the underscore itself and all but one of the leading blanks from the next line are removed. Blanks preceding the underscore are left intact. For example,

```
ME_
SS _
'Hello_
       world'
```

is an extravagant way of writing

```
MESS 'Hello world'
```

Note that the interpretation of "`_`" as line continuation cannot be escaped. If the command line should really end with an underscore the last argument must be quoted.

Recalling previous commands

The command lines types during a session are written into a history file. By default the file is called `last.kumac` and is updated every 25 commands. The commands `LAST` and `RECORDING` allow to change the file name and the frequency. At the start of a new session the existing file is renamed into `last.kumacold` (except on VMS) before the new `last.kumac` is created. Comment lines indicate the date and time at which the sessions were started and stopped.

In this way the user can keep track of all commands entered in the previous and in the current session. The command "`LAST -99`" flushes the buffered lines into `last.kumac` and envokes the editor on the file. The user

`BS/^E`	Move cursor to beginning/end of the line.
`^F/^D`	Move cursor forward/backward one character.
`DEL`	Delete the character to the left of the cursor.
`^A`	Toggle overwrite/insert mode.
`^B`	Move to previous item on history list.
`^U`	Delete from the beginning of the line to the cursor.
`TAB`	Move to next `TAB` stop.
`LF,CR`	Returns current buffer to the program.

Table 3.2: Key-binding for recall style `DCL`

can then extract the interactively typed commands and copy them into another `.kumac` file from which they can be re-executed.

The command "`LAST -n`" prints the last *n* commands entered. On a workstation this allows to re-execute command sequences by doing cut-and-paste operations with the mouse.

PAW provides a mechanism similar to the one found in the Unix `csh` shell for re-executing commands:

`!-n` executes the *n*'th last command once more.

`!!` is an short-cut for "`!-1`" re-executing the last command.

`!n` re-executes the *n*'th command entered since the beginning of the session.

`!` prints the commands together with their numbers. The number of lines printed depend on the recording frequency.

`!foo` re-executed the latest command line starting with the string "*foo*".

The command line numbering can also be seen if the prompt string contains "`[]`":

```
PAW > PROMPT 'Paw[] '
Paw[2]
```

On Unix and VMS PAW also provides recalling and editing of command lines for re-executing. The command `RECALL` allows to choose between different key-bindings:

- Recall style `KSH` has an Emacs-like binding (table 3.1) similar to the one used by the `ksh` and `bash` shells. If the terminal returns ANSI escape sequences the arrow keys can be used instead of `^B/^F/^N/^P`.
- Recall style `DCL` implements the key-binding of VMS line editing (table 3.2).
- The style names `KSHO` and `DCLO` allow to switch to overstrike mode instead of the default insert mode.
- Recall style `NONE` directs PAW to do plain reading from the terminal input.

3.2 Aliases

Aliases allow the user to define abbreviations for parts of a command line. There are two types of aliases, *command aliases* and *argument aliases*, which differ in the way they are recognized in a command line. Both alias types can be defined by the `ALIAS/CREATE` command:

```
 * KUIP/ALIAS/CREATE NAME VALUE [ CHOPT ]

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

   Possible CHOPT values are:

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

The alias value may be any string but the alias name can only consist letters, digits, "_", "-", "@", and "$" characters. Command and argument aliases share the same name space. If a command alias with the same name as an existing argument alias is created, the argument alias is deleted first, and vice versa.

3.2.1 Argument aliases

If an argument alias name is recognized anywhere in the command line it is substituted by its value. The name matching is case-insensitive and the substitution is literally, i.e. without case folding or insertion of additional blanks. The replacement is scanned for further occurrences of alias names which in turn will be replaced as well.

The alias name must be separated from the rest of the command line either by a blank or by one of the special characters

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

(not necessarily the same character on both sides). For example, if `foo` and `bar` are alias names, `foot` and `Bar-B-Q` are not affected. If two alias replacements need to be concatenated the "`//`" operator can be used, i.e.

```
ALIAS/CREATE DIR disk$user:[paw]
ALIAS/CREATE FIL file.dat
HISTO/FILE 1 DIR//FIL
```

translates into "`HISTO/FILE 1 disk$user:[paw]file.dat`". Since argument aliases are also recognized in the command position with the definition abbreviations like `HISTO/FIL` cannot be used anymore.

Alias substitution does not take place inside quoted strings. The `ALIAS` commands themselves are treated as a special case. In the command line parsing they are specifically exempted from alias translation in order to allow aliases can be deleted and redefined without quoting. For example,

```
PAW > ALIAS/DELETE *
PAW > ALIAS/CREATE foo bar
PAW > ALIAS/CREATE bar BQ
PAW > ALIAS/CREATE foo tball
PAW > ALIAS/LIST
 Argument aliases:
 BAR        => BQ
 FOO        => tball
 No Command aliases defined.
```

redefines `FOO` rather than creating a new alias name `BQ`. The value part, however, is subject to alias translations. If the aliases are created in reverse order

```
PAW > ALIAS/DELETE *
PAW > ALIAS/CREATE bar BQ
PAW > ALIAS/CREATE foo bar
PAW > ALIAS/LIST
 Argument aliases:
 BAR        => BQ
 FOO        => BQ
 No Command aliases defined.
```

the second alias is created as "`ALIAS/CREATE foo BQ`". In this case quoting the alias value does not avoid the translation. Writing instead

```
ALIAS/CREATE foo 'bar'
```

will yield the same result. Since the `ALIAS` commands bypass part of the command line parsing the translation of the value part has to be applied by the `ALIAS/CREATE` command itself. At that stage the information about quoting is no longer available.

The option "`N`" allows to inhibit the alias expansion in the value. Using this option can lead to an infinite recursion of alias translations which will be detected only when one the alias names involved is actually used.

```
PAW > ALIAS/DELETE *
PAW > ALIAS/CREATE foo bar
PAW > ALIAS/CREATE -N bar foo
PAW > ALIAS/LIST
 Argument aliases:
 BAR        => foo
 FOO        => bar
 No Command aliases defined.
PAW > foo
 *** Recursive command alias in foo
 *** Recursive argument alias in foo
 *** Unknown command: foo
PAW > bar
 *** Recursive command alias in bar
 *** Recursive argument alias in bar
 *** Unknown command: bar
```

Alias substitution happens before the command line is split-up into command name and arguments. Hence, aliases can represent several arguments at once. For example,

```
ALIAS/CREATE limits '100 -1.57 1.57'
FUN1 10 sin(x) limits
```

is equivalent to

```
FUN1 10 sin(x) 100 -1.57 1.57
```

The quotes in the `ALIAS/CREATE` command are necessary but they are not part of the alias value. If an alias value containing blanks is supposed to be treated as a single argument four extra quotes are needed in order that

```
ALIAS/CREATE htitle '''X vs. Y'''
1D 10 htitle 100 0 1
```

is equivalent to

```
1D 10 'X vs. Y' 100 0 1
```

Argument aliases can lead to unexpected interpretations of command lines. For example, a user defining

```
ALIAS/CREATE e EDIT
```

wants "E" to be short-hand for the command `EDIT`. However, the following consequence is probably not intended:

```
PAW > nt/plot 30.e
 ***** Unknown name ---> EDIT
```

For historic reasons the default option for the `ALIAS/CREATE` command is to define an argument alias. However, the use of argument aliases can lead to subtle side-effects and should therefore be restricted as much as possible.

3.2.2 Command aliases

This problem described above does not arise if a command alias is created instead:

```
ALIAS/CREATE -C e EDIT
```

Command aliases are only recognized if they appear at the beginning of a command line (ignoring leading blanks). Hence, there is no need to protect command arguments from inadvertent substitutions. Furthermore the match must be exact (ignoring case differences), i.e. the command

```
/GRAPHICS/HPLOT/ERRORS
```

can still be abbreviated as `HPLOT/E`.
Alias values can also represent several commands by using one of the line separators described in section 3.1.3, e.g.

```
ALIAS/CREATE -C ciao 'MESS Hello world! ; MESS How are you?'
```

3.3 System functions

A set of built-in, so-called system functions is provided. They allow, for example, to inquire the current dialogue style or to manipulate strings. The complete list of available functions can be obtained from "HELP KUIP/FUNCTIONS".

The function name is preceded by a $-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[1]. "$OSMOSIS" will not be recognized as the function "$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 $-sign may also be an environment variable[2]. The replacement value for "$*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 functions, 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.

3.3.1 Inquiry functions

Style inquiries

– $STYLE returns the name of the currently active dialogue style ("C", "G", "GP", etc.). This allows, for example, to a common logon macro containing different default setups depending whether PAW is started in command line mode or in Motif mode:

```
IF $STYLE='XM' THEN
   ...
ELSE
   ...
ENDIF
```

– $LAST returns the previously executed command sequence:

```
PAW > MESS Hello world! ; MESS How are you?
 Hello world!
 How are you?
PAW > MESS $LAST
 MESS Hello world! ; MESS How are you?
```

– $KEYVAL returns the content of the last selected panel box in style GP and
– $KEYNUM returns row/column address in the form "*row.col*". The column address is always given as a two-digit number.

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.

[1]Excluding the colon as separator avoids the substitution of VMS logical name containing a dollar-sign such as in "DISK$OS:[dir]file.dat!''

[2]On VMS there is a distinction between lowercase and uppercase names. Uppercase names (without the $-sign) are searched for first in the logical name tables and then in the symbol table while lowercase names are searched for only in the symbol table. The names HOME, PATH, TERM, and USER have a predefined meaning. In order to avoid conflicts with DCL symbols which are merely defined as abbreviations for running executables and DCL procedures all values starting with a "$" or "@" character are excluded from substitution.

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.

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

Environment inquiries

- `$ARGS` returns the program arguments with which PAW was invoked.
- `$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 PAW is running on, e.g. “UNIX”, “VMS” etc...
- `$MACHINE` returns an identification for the particular hardware platform or Unix brand, e.g. “HPUX”, “IBM”, or “VAX”. Table 3.3 shows the `$OS` and `$MACHINE` values for the different platforms.
  On Unix platforms the operating system version can be obtained by `$SHELL('uname -r')`.
- `$PID` returns the process number or “1” if the operating system does not support the notion of process IDs.
- `$IQUEST(`*i*`)` returns the *i*’th component of the status vector

  ```
  COMMON /QUEST/ IQUEST(100)
  ```

  `IQUEST(1)` always contains the return code of the most recently executed command.
- `$DEFINED(`*name*`)` returns *name* if a variable of that name is defined, or the empty string if the variable is not defined. The argument can contain “*” as wildcard matching any sequence of characters. All matching variable names are returned as a blank separated list.
- `$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*`,`*n*`)` returns the *n*’th line of output from the shell command.
- `$SHELL(`*command*,*sep*`)` returns the output from the shell command, where newlines are replaced by the separator string. The *sep* argument can be omitted and defaults to a single blank character.
  The `$SHELL` function is operational only on Unix systems. The *command* string is passed to the shell set by the `HOST_SHELL` command. Alias definitions etc. in the shell specific startup script (e.g. `.cshrc`) are taken into account.

3.3.2 String manipulations

- `$LEN(`*string*`)` returns the number of characters in *string*.
- `$INDEX(`*string*`,`*substring*`)` returns the position of the first occurence 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

$OS	$MACHINE	Platform
UNIX	ALPHA	DEC Alpha OSF
UNIX	APOLLO	HP/Apollo DomainOS
UNIX	CONVEX	Convex
UNIX	CRAY	Cray Unicos
UNIX	DECS	DECstation Ultrix
UNIX	HPUX	HP/UX
UNIX	IBMAIX	AIX for IBM/370
UNIX	IBMRT	AIX for RS/6000
UNIX	LINUX	Linux for PCs
UNIX	NEXT	NeXT
UNIX	SGI	Silicon Graphics Irix
UNIX	SOLARIS	Sun Solaris
UNIX	SUN	SunOS
VM	IBM	VM/CMS for IBM/370
MVS	IBMMVS	MVS for IBM/370
VMS	ALPHA	VMS for Alpha
VMS	VAX	VMS for Vax
MSDOS	IBMPC	MSDOS for PCs
WINNT	ALPHA	Windows/NT for DEC Alpha
WINNT	IBMPC	Windows/NT for PCs

Table 3.3: Platform identification with `$OS` and `$MACHINE`

- $string(k : k + n - 1)$ if $k > 0$, or
- $string(l + k + 1 : l + k + n)$ if $k \leq 0$, where $l =$ `LEN`$(string)$.

In any case the upper bound is clamped to `LEN`$(string)$. The argument n may be omitted and the result will extend to the end of $string$. Character counting starts with 1; by definition the replacement is empty if $k = 0$ or $n = 0$. If $n < 0$ an error message is emitted.

```
PAW > MESS $SUBSTRING(abcde,2)/$SUBSTRING(abcde,2,3)
 bcde/bcd
PAW > 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.

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

– `$WORD(`*`string,k,n,sep`*`)` returns *n* words starting from word *k*. The last two arguments may be omitted default to blank as separator character and the replacement value extending to the last word in *string*.

```
PAW > MESS $WORD('abc def ghi',2)
 def ghi
PAW > MESS $WORD('abc def ghi',2,1)
 def
```

– `$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:

```
ALIAS/CREATE htitle 'Histogram title'
1d 10 $QUOTE(htitle) 100 0 1
```

Another useful application of `$QUOTE` is to pass the value of an alias or macro variable as a character constant to a `comis` function, for example

```
foo = 'bar'
CALL fun.f($QUOTE([foo]))
```

is equivalent to "`CALL fun.f('bar')`". Since the quotes around "`'bar'`" are not part of the variable value the construct "`CALL fun.f([foo])`" would given the desired result only if the value contains blanks forcing the implicit quoting in the variable substitution.

- `$UNQUOTE(`*`string`*`)` returns a *string* with enclosing quote characters removed. The main use of this function is if a macro variable should be treated as several blank-separated lexical tokens:

```
limits = '100 0 1'
1d 10 'Histogram title' $UNQUOTE([limits])
```

3.3.3 Expression evaluations

- `$EXEC(`*`cmd`*`)` executes a macro command and returns the macro's `EXITM` value. Thus

```
mess $EXEC('mname 5')
```

is equivalent to

```
EXEC mname 5
mess [@]
```

- `$EVAL(`*`expr`*`)` returns the value of a numeric expression. The expression can contain numeric constants and references to vector elements joined by "`+`", "`-`", "`*`", "`/`". Parentheses may be used to override the usual operator precedence. In addition, the functions `ABS(`*`x`*`)` (absolute value), `INT(`*`x`*`)` (truncation towards zero), and `MOD(`*`x,y`*`)` (modulus) are available. Note that all operations, including division of two integer numbers, use floating point arithmetic.

```
PAW > V/CREATE vec(3) R 1.2 3.4 4.5
PAW > MESS $EVAL((2+3)/4) $EVAL(vec(1)+vec(2)+vec(3))
 1.25 9.1
```

Even if *expr* is merely a constant, the result is always in a canonical format with a maximum of 6 non-zero digits. Non-significant zeroes and the decimal point are omitted after rounding the last digit towards $+\infty$ or $-\infty$. A mantissa/exponent notation is used if the absolute value is $\geq 10^6$ or $< 10^{-4}$.

```
PAW > MESS $EVAL(1.500) $EVAL(14.99999) $EVAL(0.000015)
 1.5 15 1.5E-05
```

The explicit use of `$EVAL` is only necessary if the result should be inserted in a place where a string is expected, for example in the `MESSAGE` command. In the instances where a command expects an integer or real argument expressions are implicitly evaluated even without the `$EVAL` function.

- `$SIGMA(`*`expr`*`)` passes the expression to `sigma` for evaluation. `sigma` is an array manipulation package which supports a multitude of mathematical functions (`SQRT`, `EXP`, etc.) operating on scalars and vectors:

```
PAW > V/CREATE v10(10) R 1 2 3 4 5 6 7 8 9 10
PAW > MESS $SIGMA(2*pi) $SIGMA(vsum(v10))
 6.28319 55
```

For a description of the complete `sigma` expression syntax refer to chapter 5.
`sigma` expressions do not follow the syntax rules for PAW expressions. Therefore they cannot contain PAW system functions with arguments. They may, however, contain argument-less system functions, alias names, and macro variables.

- `$RSIGMA` is a slight variation of `$SIGMA`. Both functions return a scalar result in the same canonical format used by `$EVAL`. The only difference is that `$SIGMA` removes the decimal point from integral values while `$RSIGMA` leaves it in. For example, `$RSIGMA` should be used to calculate argument values to be passed to a `comis` routine

```
      SUBROUTINE FUN(X)
      PRINT *,X
      END
```

as floating point constants:

```
PAW > CALL fun.f($SIGMA(sqrt(8)))
  2.828430
PAW > CALL fun.f($SIGMA(sqrt(9)))
  .4203895E-44
PAW > CALL fun.f($RSIGMA(sqrt(9)))
  3.000000
```

If the expression evaluates to a vector result `$SIGMA` (and `$RSIGMA`) return the name of a temporary vector containing the result. `$SIGMA` with a vector result can be used in all places where a vector name is expected, e.g.

```
PAW > V/PRINT $SIGMA(sqrt(array(3,1#3)))
 ?SIG1(1) = 1
 ?SIG1(2) = 1.41421
 ?SIG1(3) = 1.73205
```

The lifetime of these vectors is limited to the current command. Hence, their names should not be assigned to macro variables and not be used in alias definitions:

```
PAW > A/CREATE square_roots $SIGMA(sqrt(array(3,1#3)))
PAW > V/PRINT square_roots
 *** VECTOR/PRINT: unknown vector ?SIG1
```

– `$FORMAT(`*`expr,format`*`)` returns the expression value formatted according to the Fortran *format* specifier. The possible formats are "`F`", "`E`", "`G`", "`I`", and "`Z`" (hexadecimal).

```
PAW > MESS 'x = '//$FORMAT(1.5,F5.2)
 x =  1.50
PAW > MESS 'i = '//$FORMAT(15,I5)
 i =    15
PAW > MESS 'j = '//$FORMAT(15,I5.4)
 j =  0015
```

– `$INLINE(`*`name`*`)` allows to insert the value of an alias or macro variable into an expression which is then treated as being part of the expression. For example,

```
convert = '$UPPER'
foo = $INLINE([convert])('bar')
```

is equivalent to "`foo = $UPPER('bar')`", i.e. "`foo = 'BAR'`". Without `$INLINE` the content of `[convert]` would be treated as a text string with the result that "`foo = '$UPPER(''bar'')'`".

3.3.4 Histograms inquiry functions

– `$HEXIST(id)` returns 1 if histogram `id` exists or 0 otherwise
– `$HINFO(id,'ENTRIES')` returns the number of entries.
– `$HINFO(id,'MEAN')` returns the mean value.
– `$HINFO(id,'RMS')` returns the standard deviation.
– `$HINFO(id,'EVENTS')` returns the number of equivalent event.
– `$HINFO(id,'OVERFLOW')` returns the content of overflow channel.
– `$HINFO(id,'UNDERFLOW')` returns the content of underflow channel.
– `$HINFO(id,'MIN')` returns the minimum bin content.
– `$HINFO(id,'MAX')` returns the maximum bin content.
– `$HINFO(id,'SUM')` returns the total histogram content.
– `$HINFO(id,'XBINS')` returns the number of bins in X direction.
– `$HINFO(id,'XMIN')` returns the lower histogram limit in X direction.
– `$HINFO(id,'XMAX')` returns the upper histogram limit in X direction.
– `$HINFO(id,'YBINS')` returns the number of bins in Y direction.
– `$HINFO(id,'YMIN')` returns the lower histogram limit in Y direction.
– `$HINFO(id,'YMAX')` returns the upper histogram limit in Y direction.
– `$HTITLE(id)` returns the histogram title.

3.3.5 Graphics inquiry functions

- $GRAFINFO('XZONES') returns the number of zones in X direction.
- $GRAFINFO('YZONES') returns the number of zones in Y direction.
- $GRAFINFO('NT') returns the current normalization transformation number.
- $GRAFINFO('WNXMIN') returns the lower X limit of window in current NT.
- $GRAFINFO('WNXMAX') returns the upper X limit of window in current NT.
- $GRAFINFO('WNYMIN') returns the lower Y limit of window in current NT.
- $GRAFINFO('WNYMAX') returns the upper Y limit of window in current NT.
- $GRAFINFO('VPXMIN') returns the lower X limit of viewport in current NT.
- $GRAFINFO('VPXMAX') returns the upper X limit of viewport in current NT.
- $GRAFINFO('VPYMIN') returns the lower Y limit of viewport in current NT.
- $GRAFINFO('VPYMAX') returns the upper Y limit of viewport in current NT.
- $GRAFINFO('?attr') returns the current value of the hplot/higz attribute attr. See the HELP of the command SET to have the list of the valid values of attr.

3.3.6 Cuts manipulations

- $CUT(n) returns the cut expression $n
- $CUTEXPAND(string) replace $n in the (quoted) string by $CUT(n)

3.4 Vectors

PAW provides the facilities to store vectors of integer or real data. These vectors, or rather arrays with up to 3 index dimensions, can be manipulated by PAW commands (see “HELP VECTOR”). Furthermore they are interfaced to the array manipulation package sigma and to the Fortran interpreter comis (see chapter 5).

Vectors are memory resident only, i.e. they are not preserved across program executions. The commands VECTOR/READ and VECTOR/WRITE allow to save and restore vector data from an external file in text format.

Vector names may be composed of letters, digits, underscores and question marks up to a maximum length of 32 characters[3]. Names starting with “?” are reserved for internal use by PAW.

The only exception is the predefined vector simply called “?” which has a fixed size of 100 real elements. Unlike the others the “?” vector is mapped to a fixed memory location (the common block /KCWORK/). It does not show up in VECTOR/LIST output and it is not counted in the result of $NUMVEC.

3.4.1 Creating vectors

Vectors can be created with the VECTOR/CREATE command. The size of the index dimensions is given in Fortran notation, e.g.

```
VECTOR/CREATE v1(100)
VECTOR/CREATE v2(10,10)
```

The lower index bound always starts off at 1, i.e. “V/CREATE v(-10:10)” is not allowed. Omitting the index dimension as in

```
VECTOR/CREATE v
```

is equivalent to

```
VECTOR/CREATE v(1)
```

PAW does not keep track of the actual number of index dimension given in the VECTOR/CREATE command, i.e.

```
VECTOR/CREATE v1(10)
VECTOR/CREATE v3(10,1,1)
```

are equivalent.

[3]Vector names which should be processed by sigma are currently limited to 7 characters.

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,NPLANE)`

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

Figure 3.4: Addressing scheme for vectors

3.4.2 Accessing vectors

Single vector elements can be used in expressions where they are treated as numeric constants. Vectors with a single element only we will refer to as "*scalar vectors*". They have the special property that in expressions it is sufficient to give the name without the "`(1)`" subscript.

Complete vectors and vector subranges can be used in the `$SIGMA` function and as argument to commands expecting a vector name. The subrange notation is the same as in Fortran, e.g. `v(3:5)`. The elements of arrays are stored in column-major order, i.e. the elements `v(1,2)` and `v(2,2)` are adjacent in memory (see figure 3.4).

The vector processing commands are expected to deal only with contiguous vectors. Therefore a subrange referring to a non-contiguous set of elements is copied into a temporary vector and cannot be used as output parameter.

3.5 Expressions

PAW has a built-in parser for different kinds of expressions: arithmetic expressions, boolean expressions, string expressions, and "garbage expressions".

3.5.1 Arithmetic expressions

The syntactic elements for building arithmetic expressions are shown in table 3.4. They can be used

- in the macro statements `DO`, `FOR`, and `EXITM`;
- in macro variable assignments;
- as system function arguments where a numeric value is expected;
- as argument to the `$EVAL` function.

Note that all arithmetic operations are done in floating point, i.e. "`5/2`" becomes "`2.5`". If a floating point result appears in a place where an integer is expected, for example as an index, the value is truncated.

expr	::=	*number*	
	|	*vector-name*	for scalar vectors
	|	*vector-name* (*expr*)	
	|	*vector-name* (*expr* , *expr*)	
	|	*vector-name* (*expr* , *expr* , *expr*)	
	|	[*variable-name*]	if variable value has form of a numeric constant or is the name of a scalar vector
	|	[*variable-name*] (*expr* ...)	if variable value is a vector name
	|	*alias-name*	if alias value has form of a numeric constant
	|	$*system-function* (...)	if function returns a numeric value
	|	- *expr*	
	|	*expr* + *expr*	
	|	*expr* - *expr*	
	|	*expr* * *expr*	
	|	*expr* / *expr*	
	|	(*expr*)	
	|	ABS (*expr*)	
	|	INT (*expr*)	
	|	MOD (*expr* , *expr*)	

Table 3.4: Syntax for arithmetic expressions

bool	::=	*expr rel-op expr*	*rel-op*	::=	.LT.	|	.LE.
	|	*string eq-op string*		|	<	|	<=
	|	*expr eq-op string*		|	.GT.	|	.GE.
	|	.NOT. *bool*		|	>	|	>=
	|	*bool* .AND. *bool*		|	*eq-op*		
	|	*bool* .OR. *bool*	*eq-op*	::=	.EQ.	|	.NE.
	|	(*bool*)		|	=	|	<>

Table 3.5: Syntax for boolean expressions

3.5.2 Boolean expressions

Boolean expressions can only be used in the macro statements IF, WHILE, and REPEAT. The possible syntactic elements are shown in table 3.5.

In addition, a single arithmetic expression is also accepted as boolean expression, interpreting any non-zero value as *true*. This allows, for example, the short-cuts

```
IF $VEXIST(v1) THEN
...
WHILE 1 DO
...
```

instead of the explicit forms

```
IF $VEXIST(v1)<>0 THEN
...
WHILE 1=1 DO
...
```

Note, however, that an arithmetic expression is **not** equivalent to a boolean value. This implies that

string	`::=`	*quoted-string*	
	|	*unquoted-string*	
	|	*string* `//` *string*	concatenation
	|	*expr* `//` *string*	value of expression converted to string representation
	|	`[`*variable-name*`]`	
	|	*alias-name*	
	|	`$`*system-function* `(...)`	

Table 3.6: Syntax for string expressions

```
IF $VEXIST(v1) .and. $VEXIST(v2) THEN   | error
```

is not accepted and has to be written as

```
IF $VEXIST(v1)<>0 .and. $VEXIST(v2)<>0 THEN
```

3.5.3 String expressions

String expressions can be used
- in the macro statements `CASE`, `FOR`, and `EXITM`
- in macro variable assignments
- as system function arguments where a string value is expected
- as argument to the `$EVAL` function

They may be constructed from the syntactic elements shown in table 3.6.

3.5.4 Garbage expressions

Expressions which do not satisfy any of the above syntax rules we want to call "garbage" expressions. For example,

```
s = $OS$MACHINE
```

is not a proper string expression. Unless they appear in a macro statement where specifically only an arithmetic or a boolean expression is allowed, PAW does not complain about these syntax errors. Instead the following transformations are applied:

1 alias substitution
2 macro variable replacement; values containing a blank character are implicitly quoted
3 system function calls are replaced one by one by their value provided that the argument is a syntactically correct expression
4 string concatenation

The same transformations are also applied to command arguments. Therefore the concatenation operator "`//`" can be omitted in many cases. For example,

```
MESS $OS$MACHINE
MESS $OS//$MACHINE
MESS $EVAL($OS$MACHINE)
MESS $EVAL($OS//$MACHINE)
```

give all the same result.

3.5.5 The small-print on expressions

Expressions are evaluated by a `yacc`-generated parser. `Yacc` (“*Yet Another Compiler-Compiler*”) is a standard Unix tool. It produces a C routine to parse an token stream which follows the syntax rules fixed by the grammar definition.

The parser needs as front-end a lexical analyzer which reads the input stream, separates it into tokens, and returns the token type and its value to the parser. There is another Unix tool `lex` which can produce an appropriate lexical analyzer from a set of rules. The PAW lexical analyzer had to be hand-crafted because the interpretation of a symbol depends very much on the global context. For example, if the input stream consists is simply “`foo`” the lexical analyzer has to check consecutively:

- If `foo` is defined as an alias:
  - If the alias value looks like a number, classify it as a *number*.
  - Otherwise classify the alias value as a *string*.
- Otherwise classify it as the *string* “`'foo'`”.

A similar reasoning has to be applied for “`[foo]`”:

- If `foo` is a defined macro variable:
  - If the variable value looks like a number, classify it as a *number*.
  - If the variable value is the name of a scalar vector, classify it as a *number*.
  - Otherwise classify the variable value as a *string*.
- Otherwise classify it as the *string* “`'[foo]'`”.

Macro variables do not have to (and cannot) be declared. The value is always stored as a string and it depends on the context whether the value should be interpreted as a number. Also there is no way to tell in the beginning whether the right-hand side of an assignment is an arithmetic or a string expression.

The lexical analyzer starts off interpreting tokens as a numbers if it can. For example,

```
a = '1'
b = '2'
c = [a]+[b]
```

is tokenized as “*number* + *number*” and gives “`c = 3`” even though the values assigned to `a` and `b` are originally quoted. If we have a string expression

```
[foo]//[bar]
```

this could result in the possible token sequences

```
string // string
number // string
string // number
number // number
```

depending whether the values of `foo` and `bar` look like a number. Accordingly we would have to define four grammar rules to cover these different cases. The same problem occurs in system functions expecting a string argument, e.g.

```
$SUBSTRING([foo],2,3)
```

would need two rules for `foo` being a number or a genuine string.

`Yacc` allows to avoid this inflation of necessary rules by using so-called lexical tie-ins. After having seen “`//`” or “`$SUBSTRING(`” the parser can instruct the lexical analyzer that it should not attempt to classify the next token as a number. Therefore a single rule for each system function is sufficient.

However, a lexical tie-in can only be used after the parser found a unique match between the token sequence and all grammar rules In the case of string concatenation we still have to provide two separate rules for

```
string // string
number // string
```

The grammar rule (see above) actually says that the left-hand side of the “`//`” operator can be either an arithmetic or a string expression. An arithmetic expression is evaluated and then transformed into the result’s string representation. For example,

```
2*3//4
```

gives “`'64'`”. On the other hand,

```
4//2*3
```

gives "'42*3'". It does not become "'46'" because the right-hand side is not consider to be an arithmetic expression. It does also not become "126" because a result of a string operation is never again treated as a number even if it looks like one.

The lexical analyzer forwards numbers in arithmetic expressions as floating point values to the parser. The result is converted back to the string representation when it has to be stored in the macro variable. Since a single numeric value already counts as an arithmetic expression the original string representation can be lost. For example,

```
a = '0123456789'
b = [a]
MESS $LEN([a]) $LEN([b])
```

results in "10 11" because the assignment "b = 0123456789" is taken as an arithmetic expression which is reformatted into 1.23457E+08. The reformatting can be inhibited by using

```
b = $UNQUOTE([a])
```

The `$UNQUOTE` function removes quotes around a string. If the string is already unquoted it does nothing except that in this case the parser will treat the value of `[a]` as a string.

Macros should not depend on this reformatting behavior. We consider it as an obscure side-effect of the present implementation rather than a feature.

3.6 Macros

A macro is a set of command lines stored in a file, which can be created and modified with any text editor. The command `EXEC` invokes the macro and allows for two ways of specifying the macro name:

```
EXEC file
EXEC file#macro
```

The first form executes the first macro contained in *file* while the second form selects the macro named *macro*. The default extension for *file* is "`.kumac`".

Example of macro calls

```
PAW > EXEC abc   | Execute first (or unnamed) macro of file abc.kumac
PAW > EXEC abc#m | Execute macro M of file abc.kumac
```

In addition to all available commands the special "macro statements" in table 3.7 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.

3.6.1 Macro definitions and variables

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

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

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

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

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

```
ENDKUMAC
```

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

The `APPLICATION` statement has the same form and similar functionality as the `SET/APPLICATION` command:

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

Table 3.7: Macro statements

```
APPLICATION  command  marker
   text
marker
```

The text up to the next line containing only the end marker starting in the first column is written to a temporary file and then passed to the application command. The text is not interpreted in any way, i.e. variable substitution etc. does not take place.

Instead of the full spelling `APPLICATION` any valid abbreviation of `/KUIP/SET_SHOW/APPLICATION` be used, e.g. “`APPL`”. A call to `SET/APPLICATION` as a result of an alias expansion, however, is not allowed.

Macro execution

Inside a macro the `EXEC` statement can call other macros. A macro may also call itself recursively. The `EXEC` command allows two different forms for specifying the macro to be executed:

```
EXEC  fname#mname  [ argument-list ]
```

and

```
EXEC  name  [ argument-list ]
```

Between the EXEC *statement* and the EXEC *command* there is a slight difference. The command "EXEC name" executes the first macro in name.kumac while the EXEC statement will try first whether a macro name is defined within the current .kumac file.
Macro execution terminates when one of the statements

```
EXITM  [ expression ]
```

or

```
RETURN  [ expression ]
```

or

```
STOPM
```

is encountered. The EXITM and RETURN statements return to the calling macro. They allow to pass a return value which is stored into the special variable [@] of the calling macro. If no value is given it defaults to "0". Note that the RETURN statement also flags the end of the macro definition, i.e. the construct

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

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

Macro variables

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

```
name = expression
```

The right-hand side of the assignment can be an arithmetic expression, a string expression, or a garbage expression (see section 3.5). The expression is evaluated and the result is stored as a string (even for arithmetic expressions).
The variable value can be used in other expressions or in command lines by enclosing the name in square brackets:

```
[name]
```

For example,

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

If the name enclosed in brackets is not a macro variable then no substitution takes place.
Variable values can also be queried from the user during macro execution. The statement

```
READ  name  [ prompt ]
```

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

Macro using the READ statement

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

Output when executing

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

Macro arguments

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

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

we get the result

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

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

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

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

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

we get the result

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

The macro parameters can also be named, for example:

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

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

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

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

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

yields

```
PAW > EXEC m
 old old
 old new
```

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

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

can be used as

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

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

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

i.e. the second argument "bar" is not assigned to [arg2] or [2] but to [3]. Note that this differs from the behavior for command arguments (see section 3.1.2).

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

```
MACRO m
MESS [foo]
```

yields

```
PAW > EXEC m
 [foo]
PAW > EXEC m foo=bar
 bar
```

Therefore a string containing a "=" must be quoted if it should be passed to the macro literally:

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

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

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

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

Passing a value as argument to a macro is not quite the same as assigning the value to a variable inside the macro. The macro argument is not tried to be evaluated as an arithmetic expression. String operations, however, such as concatenation and alias substitutions, are applied. For example, "EXEC m1 2*3 4//5" with

```
MACRO m1 a=0 b=0
mess [a] [b]
```

yields "2*3 45", while "EXEC m2" with

```
MACRO m1
a = 2*3
a = 4//5
mess [a] [b]
```

yields "6 45". Macro arguments are not tried as arithmetic expressions in order to allow passing of vector names without the use of quotes. Otherwise "EXEC m v1", where v1 is a scalar vector, would pass the value of v1(1) rather than the string 'v1'.

Note that the result "6 45" can also be obtained from the first of the above examples by means of the $INLINE function:

```
MACRO m1 1=0 b=0
a = $INLINE([1])
mess [a] [b]
```

Special variables

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

```
SHIFT
```

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

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

allows to traverse the list of macro arguments.

For each macro the following special variables are always defined:

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

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

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
     V/CREATE V([NUM])
  ENDIF
RETURN
```

Output when executing

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

Variable indirection and arrays

Macro variables can be referenced indirectly by constructing the name using other variables, for example

```
DO i = 1, 10
  a_[i] = [i] * [i]
ENDDO
s = 0
DO i = 1, 10
  s = [s] + [a_[i]]
ENDDO
```

While for PAW we simply created ten variables `a_1`, ..., `a_10`, we can also look at it as an array `a_`*i*. We don't even need to remember the dimension of the array. The system function `$DEFINED` returns all defined variables matching a wildcard, for example

```
s = 0
DO i = 1, $WORDS($DEFINED('a_*'))
  s = [s] + [a_[i]]
ENDDO
```

Instead of `a_`*i* we can also use the more conventional array notation `a(`*i*`)`

```
DO i = 1, 10
  a([i]) = [i] * [i]
ENDDO
s = 0
DO i = 1, $WORDS($DEFINED('a(*)'))
  s = [s] + [a([i])]
ENDDO
```

as long as we have the possibility to match all array elements with a single wildcard expression.

Since for PAW all array elements are just simple variables the indices do not even need to be numeric. We can also construct associative arrays where the indices are names, for example

```
events(mu) = 1000
events(el) = 100
events(tau) = 10
total = 0
names = $DEFINED('events(*)')
DO i = 1, $WORDS([names])
  name = $WORD([names],[i],1)
  total = [total] + [[name]]
ENDDO
```

By the same token we can also create multi-dimensional arrays, for example

```
DO i = 1, 3
  DO j = 1, 3
     a([i],[j]) = [i]*2+[j]
  ENDDO
ENDDO
```

The `$DEFINED` function returns the matching variable names sorted in alphabetical order, i.e.

```
$DEFINED('events(*)') is 'events(el) events(mu) events(tau)'
$DEFINED('a(*)') is 'a(1) a(10) a(2) ... a(9)'
```

and not necessarily in the order in which they were created.

The indirection only allows for variable substitution when constructing the actual variable name. Expression evaluation etc. does not take place and constructs such as

```
  total = [total] + [$WORD([names],[i],1)]   | invalid!
```

are not allowed.

The construct `[[name]]` can also be written as

```
[%name]
```

For example, this is another way to traverse the list of macro arguments:

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

Except for the `[%name]`

Global variables

Global variables can be made visible inside a macro by executing the commands `GLOBAL/CREATE` or `GLOBAL/IMPORT`. Technically these commands create a local variable with the same name initialized to the value of the global variable. When assigning a value to the local variable the change is also propagated to the global variable. Therefore, once they are made visible inside a macro, global variables are assigned to and used in the same way as local variables.

The `GLOBAL/CREATE` command creates a global variable allowing to specify an initial value and a comment text, e.g.

```
GLOBAL/CREATE m_e  0.0005 'Electron mass (GeV)'
GLOBAL/CREATE m_mu 0.106  'Muon mass (GeV)'
```

If executed inside a macro the global variable becomes visible there.

The `GLOBAL/IMPORT` command has an effect only when executed inside a macro. It allows to make global variables visible which have been created elsewhere. The import list may contain "*" as a wildcard for any character sequence, for example

```
GLOBAL/IMPORT m_*
```

Only those global variables existing at the time the `GLOBAL/IMPORT` is executed become visible. Therefore, global variables created in an inferior macro do not become visible even if they match the wildcard. For example, in

```
MACRO a
GLOBAL/IMPORT m_*
EXEC b
...
RETURN

MACRO b
GLOBAL/CREATE m_tau 1.784 'Tau mass (GeV)'
RETURN
```

`m_tau` is not visible in macro `a` unless it is imported after executing `b`.

Deleting a global variable in an inferior macro, on the other hand, also deletes the associated local variables in the macro call stack. For example, in

```
MACRO a
GLOBAL/IMPORT m_*
EXEC b
...
RETURN

MACRO b
GLOBAL/DELETE m_mu
RETURN
```

when returning from macro b the imported variable m_mu will become undefined.

Global variables can also be set and used from the command line, for example,

```
PAW > g/cre x 2
PAW > x=[x]*2
PAW > mess [x]
 4
```

However, the implicit creation when assigning a value to an undefined variables does not apply:

```
PAW > y=0
 *** Unknown command: y=0
```

Global variables are available only since the 95a release.

3.6.2 Flow control constructs

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

The simplest form of flow control is provided by the

```
GOTO label
```

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

```
label:
```

If the jump leads into the scope of a block statement, for example a DO-loop, the result is undefined. The target may be given as an expression evaluating to the actual label name, e.g.

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

In the label definition the colon must follow the label name immediately without any intervening blanks. The label may be followed by a command on the same line, e.g.

```
label: MESS Hello
```

Conditional execution

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

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

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

```
IF expression GOTO label
```

This old-fashioned construct is equivalent to

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

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

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

Each label is a string constant and the comparison with the selection expression is case-sensitive. If a label is followed by another label without intervening statements then a match of the first label will skip to the ENDCASE immediately. In order to execute the same statement sequence for distinct labels a comma-separated list of values can be used. The "*" character in a label item acts as wild-card matching any string of zero or more characters, i.e. "(*)" constitutes the default label.

Example for CASE labels with wild-cards

```
   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
```

Loop constructs

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

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

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

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

where all variables except for `loop` are temporary.

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

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

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

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

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

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

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

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

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

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

```
WHILE expression DO
   statements
ENDWHILE
```

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

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

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

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

```
BREAKL  [ levels ]
```

allows to terminate a loop prematurely. The BREAKL statement continues executing after the end clause of the enclosing DO, FOR, WHILE, or REPEAT block.

```
NEXTL  [ levels ]
```

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

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

Example of using BREAKL and NEXTL

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

Equivalent code using GOTOs

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

Error handling

Each command returns a status code which should be zero if the operation was successful or non-zero if any kind of error condition occurred. The status code is stored in the IQUEST(1) status vector and can be tested as, for example

```
HISTO/FILE 1 foo.hbook
IF $IQUEST(1)<>0 THEN
*-- cannot open file
   ... do some cleanup
   EXITM 1
ENDIF
```

```
ON ERROR GOTO label
```

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

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

and

```
ON ERROR STOPM
```

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

```
ON ERROR CONTINUE
```

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

```
OFF ERROR
```

and

```
ON ERROR
```

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

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

Another way of testing the status code of a command is to use the line separators "`;&`" and "`;!`" (see section 3.1.3). These operators take precedence over the `ON ERROR` setting.

```
cmd1 ;& cmd2 ; cmd3
```

is roughly equivalent to

```
OFF ERROR
cmd1
IF $IQUEST(1)=0 THEN
   cmd2
   ON ERROR
   cmd3
ENDIF
ON ERROR
```

except that the `ON`/`OFF ERROR` statements are virtual and do not overwrite the setting saved by a real `OFF ERROR` statement.

3.7 Motif mode

3.7.1 The Browser Interface

The Browser interface is a general tool to display and manipulate a tree structure of objects. The objects contained in the currently selected directory can be displayed in various forms: big icons, small icons, text only, etc. It is possible to perform actions on these objects or the directories it-selves by accessing pop-up menus directly attached to them: this behavior of the browser gives access to a "direct object manipulation" user interface by opposition to the usual "command mode interface".

Description of the "Main Browser" Window

When PAW++ start one browser is automatically created and displayed: it is called the **"Main Browser"**. Later on it is possible to "clone" this browser (by pressing the corresponding button at the bottom/right) when it is in a certain state. This will give to the user the possibility to have several instances of the browser window, and look at the same time to different kind of objects.

A "browser window" is composed of (Fig. 3.5):

- A menu bar with the menu entries "File" ①, "View" ②, "Options" ③, "Commands" ④ and "Help" ⑤.
- A two lines text/label area (❶ and ❷).
- The middle part of the browser is divided into two scroll-able windows: the "FileList" or **"Browsable window"** ❸ at the left and the "DirList" or **"Object window"** ❹ at the right.
- Two lines of information at the bottom (❺ et ❻), plus a "Clone" ❽ and a "Close" ❾ buttons.

Below follows a description of the middle (and main) part of the browser which is divided into two scroll-able windows on the left and right sides (Fig. 3.5):

- The left hand "FileList" or **"Browsable window"** ❸ shows the list of all the currently connected browsables. Some browsables can also be attached at run time by selecting the corresponding "Open" entry in the menu "File" (e.g. ZEBRA/RZ files for access to histograms and Ntuples).
  Pressing the right mouse button in this window shows a pop-up menu with all the possible actions which have been defined for this browsable.
  Selecting one item (or browsable) in this window with the left mouse button executes by default the "List" action (first entry of the pop-up menu): it displays the content of the browsable in the right hand window ("DirList" or **"Object window"**)
  Note that the first entry of the pop-up menu of actions for one browsable is always "List" and that the last entry is always "Help" : it should give information concerning the selected browsable.
- The right hand "DirList" or **"Object window"** ❹ shows the content of the currently selected browsable for the selected path. E.g. when you select the browsable "Macro", you will get all the macro files and sub-directories which are contained in the selected directory.
  Objects are selected by clicking on them with the left mouse button. Pressing the right mouse button pops up a menu of possible operations depending on the object type ❼.
  An item in a pop-up menu is selected by pointing at the corresponding line and releasing the right mouse button. Double clicking with the left mouse button is equivalent to selecting the first menu item.
  Each menu item executes a command sequence where the name of the selected object is filled into the appropriate place. By default the command is executed immediately whenever possible. (The commands executed can be seen by selecting "Echo Commands" in the "Options" menu of the **"Executive Window"**.) In case some mandatory parameters are missing the corresponding "Command Argument Panel" is displayed, and he remaining arguments have to be filled in. The command is executed then by pressing the "OK" or "Execute" button. (Note that if it is not the last one in the sequence of commands bound to the menu item, PAW is blocked until the "OK" or "Cancel" button is pressed.)

The two lines text/label area at the top displays information about (Fig. 3.5):

- the current path (or directory) for the selected browsable ❶ (entry "Path:"). The directory can be changed by pointing at the tail of the wanted sub-path and clicking the left mouse button. Clicking a second time on the same path segment performs the directory change and updates the "DirList" window with the list of objects.

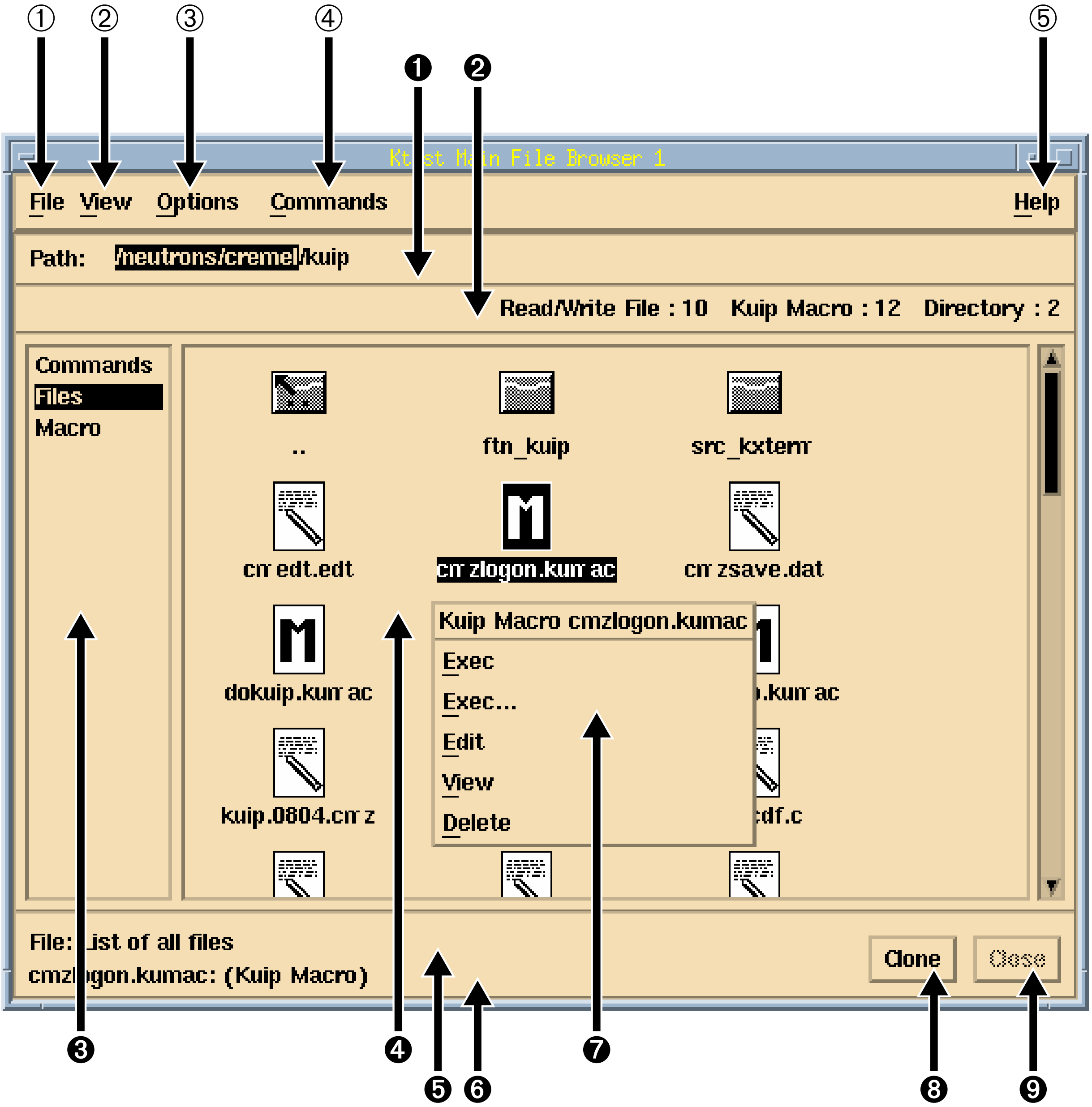


Figure 3.5: **“Main Browser”** Window

– the number of objects of all the different classes defined for the selected browsable in the current directory ❷.

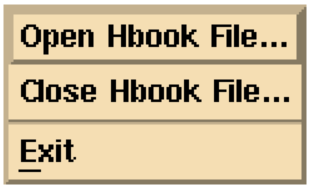
The two lines of information at the bottom are filled with (Fig. 3.5):

– a short description of the browsable which is currently selected ❺ (entry “File:”),
– a short description of the object which is selected in the “object window” for a given browsable ❻.

Below follows a description of the different Browser menus:

File

The **File** menu in the `paw++` **“Main Browser”** is shown below.



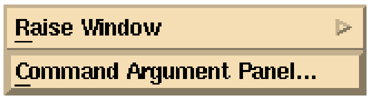
Open Hbook File...	Open one ZEBRA/RZ file.
Close Hbook File...	Close one ZEBRA/RZ file.
Exit	Exit from PAW.

View

The **View** menu allows to change the way objects are displayed or selected.

Icons	display objects with normal size icons and names (default).
Small Icons	display objects with small icons and names.
No Icons	display objects without icons, but names and small titles.
Titles	display objects without icons, but long titles.
Select All	select all the objects.
Filter...	ask for a filter to be put on object names.

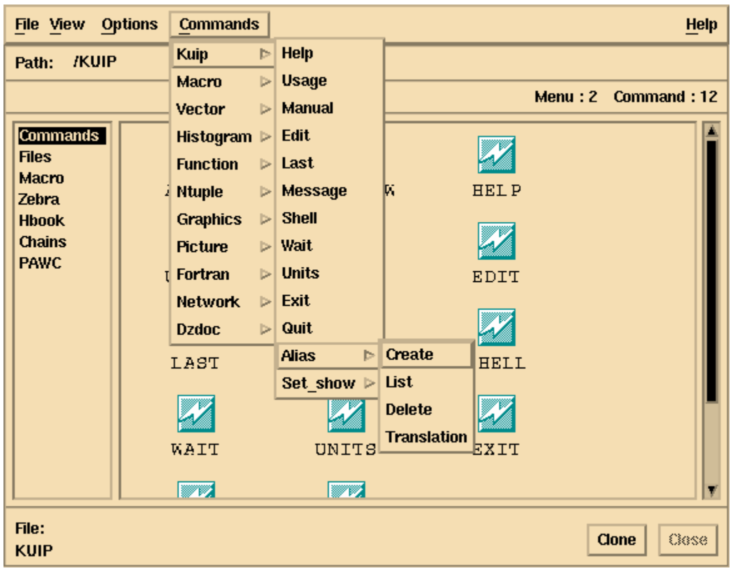
Options



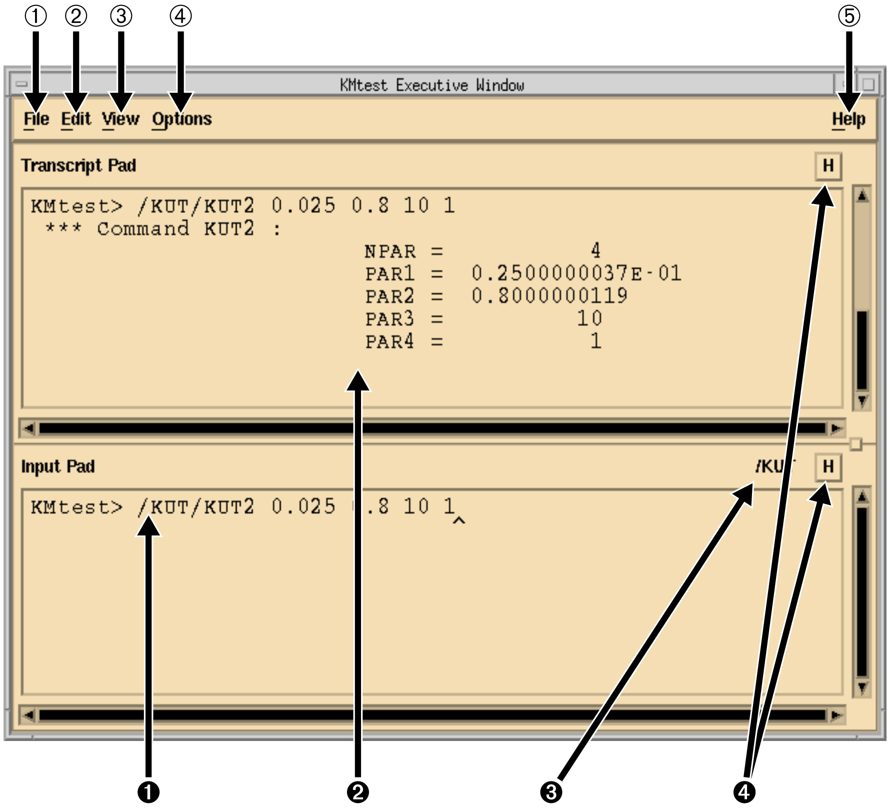
Raise Window "cascade button" with the list of all opened windows. Selecting one of this window will pop-up the window on top of the others.

Command Argument Panel selecting this entry will prompt the user for a command name. If the command is valid then the corresponding "Command Argument Panel" with the list and description of all parameters will be displayed. If the command is ambiguous (e.g. command "list") the user will be proposed a list of all the possible commands. He can then select one and the corresponding "Command Argument Panel" will be displayed. If the command does not exist an error message is displayed.

Commands



This menu gives access to the complete tree of commands. When a terminal item (command) in this menu is selected then the corresponding "Command Argument Panel" is displayed. The functionality of this menu is quite similar to the browsable "Commands" (this is just a matter of taste whether the user prefer to access commands through this pull-down menu or through the "Commands" browser).



① Menu bar entry "File".
② Menu bar entry "Edit".
③ Menu bar entry "View".
④ Menu bar entry "Options".
⑤ Menu bar entry "Help".

❶ **Input Pad**
❷ **Transcript Pad**
❸ Current working directory indicator.
❹ Hold buttons.

Figure 3.6: **"Executive Window"**

Browser Setting or Initialization

The following PAW command can be used to set up the browser in a given state, without having to click with the mouse:

```
/MOTIF/BROWSER browsable [path]
```

- *browsable* is the name of the file (browsable) you want to open (corresponding item is selected in the list of browsables).
- *path* (optional) is the pathname to be used for this browsable.

E.g. If you want to open the browser in the state displayed in Fig. 3.5, without having to click with the mouse, you can execute the PAW command:

```
/MOTIF/BROWSER Files /neutrons/kuip
```

3.7.2 The "Executive Window"

This terminal emulator combines **Input Pad** and **Transcript Pad**, (automatic file backup of **Transcript Pad**, string search in pads, etc.), the Korn shell emacs-style command line editing and command line recall mechanism.

Description and Behavior

The **"Executive Window"** is composed of three main parts (Fig. 3.6):

- A "menu bar" with the menu entries "File" ①, "Edit" ②, "View"③, "Options"④, and "Help"⑤,.
- A **Transcript Pad** ❷ which contains the text output.
- An **Input Pad** ❶ which is an edit-able "scrolled window" where the user can type commands.

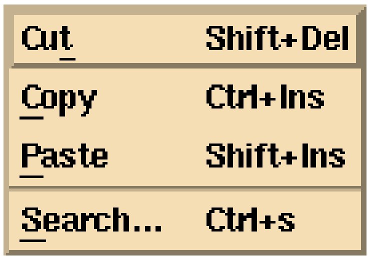
Commands are typed in the input pad behind the PAW prompt. Via the toggle buttons ❹ labeled "H" the **Input Pad** and/or **Transcript Pad** can be placed in hold mode. In hold mode one can paste or type a number of commands into the **Input Pad** and edit them without sending the commands to PAW. Releasing the hold button will causes the **"Executive Window"** to submit all lines, up to the line containing the cursor, to PAW. To submit the lines below the cursor, just move the cursor down. In this way one can still edit the lines just before they are being submitted to PAW.

Commands can be edited in the **Input Pad** using emacs-like key sequences (see section 3.7.2). The **Transcript Pad** shows the executed commands and command output. When in hold mode the **Transcript Pad** does not scroll to make the new text visible.

Every time the current directory is changed, the **Current working directory indicator** ❸ is updated. The current working directory is the one which is currently selected in the **"Main Browser"**.

Below follows a description of the different **"Executive Window"** menus. All **"Executive Window"** menus can be dynamically extended.

Edit



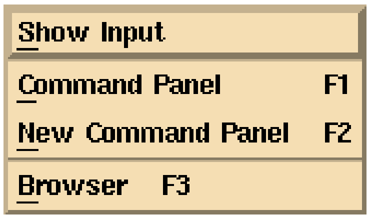
Cut Remove the selected text. The selected text is written to the Cut & Paste buffer. Using the "Paste" function, it can be written to any X11 program. In the **Transcript Pad** "Cut" defaults to the "Copy" function.

Copy Copy the selected text. The selected text is written to the Cut & Paste buffer. Using the "Paste" function, it can be written to any X11 program.

Paste Insert text from the Cut & Paste buffer at the cursor location into the **Input Pad**.

Search... Search for a text string in the **Transcript Pad**.

View



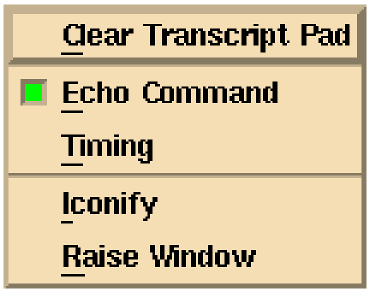
Show Input Show in a window all commands entered via the **Input Pad**.

Command Panel Gives access to the "**PANEL** interface" for a panel which has been predefined in a macro file (see section 3.7.3).

New Command Panel Gives access to the "**PANEL** interface" for setting a new and empty panel to be filled interactively (see section 3.7.3).

Browser Display another instance of the browser.

Options



Clear Transcript Pad Clear all text off of the top of the **Transcript Pad**.

Echo Command Echo executed commands in **Transcript Pad**.

Timing Report command execution time (real and CPU time).

Iconify Iconify **"Executive Window"** and all windows.

Raise Window Display a list of all windows connected. The user can select the window he wants to pop-up.

Edit Key Sequences

Please note that "C-b" means holding down the Control key and pressing the "b"-key. "M-" stands for the Meta or Alt key.

```
C-b:               backward character
M-b:               backward word
Shift M-b:         backward word, extend selection
M-[:               backward paragraph
Shift M-[:         backward paragraph, extend selection
M-<:               beginning of file
C-a:               beginning of line
Shift C-a:         beginning of line, extend selection
C-osfInsert:       copy to clipboard
Shift osfDelete:   cut to clipboard
Shift osfInsert:   paste from clipboard
Alt->:             end of file
M->:               end of file
C-e:               end of line
Shift C-e:         end of line, extend selection
C-f:               forward character
M-]:               forward paragraph
Shift M-]:         forward paragraph, extend selection
C-M-f:             forward word
C-d:               kill next character
M-BS:              kill previous word
C-w:               kill region
C-y:               yank back last thing killed
C-k:               kill to end of line
C-u:               kill line
M-DEL:             kill to start of line
C-o:               newline and backup
C-j:               newline and indent
C-n:               get next command, in hold mode: next line
C-osfLeft:         page left
C-osfRight:        page right
C-p:               get previous command, in hold mode: previous line
C-g:               process cancel
C-l:               redraw display
C-osfDown:         next page
C-osfUp:           previous page
C-SPC:             set mark here
C-c:               send kill signal
C-h:               toggle hold button of pad containing input focus
F8:                re-execute last executed command
Shift F8:          put last executed command in input pad
Shift-TAB:         change input focus
```

3.7.3 User Definable Panels of Commands

The “**PANEL** interface” allows to define command sequences which are executed when the corresponding button in the panel is pressed.

New Panel

It is possible to fill a new and empty panel interactively (see section 3.7.3) giving a label to each button.
In the top menu bar 3 pull-down menus (‘File”, “View” and “Help”) are available. The pull-down menu “File”, whose contents is displayed, contains the 2 items “Save” (to save the actual panel configuration after editing) and “Close” (to close the panel and erase it from the screen). The “View” menu contains various options for displaying the same panel in different ways (see section 3.7.3), and the “Help” menu contains various items to help the user concerning this panel interface.
This new panel definition can also be done with the command `PANEL` using the sequence

```
PANEL 0
PANEL 4.06 ' '
PANEL 0 D 'This is my first panel' 250x200+500+600
```


```
NEWPANEL 4 6 'First panel' _
         250 200 500 600
```

This command creates an empty panel with 4 rows and 6 columns of buttons. The title of this panel will be set to “First panel”. The panel size in pixels is 250 (width) x 200 (height), and the panel position (in pixels) is 500 (along X axis), 600 (along Y axis).

Figure 3.7: New Panel of Commands

You can get automatically access to the command “NEWPANEL” (and its corresponding “Command Argument Panel”) by selecting the menu item “New Command Panel” in the “View” menu of the **“Executive Window”** (Fig. 3.7.2).

Predefined Panel of Commands

The command “PANEL” for a key (or button) definition has to be used if you want to describe your panel in a macro file in order to keep trace of the panel definition, and be able to retrieve it later on. You can predefine as many panels as you want, and you can easily access them by selecting the menu item “Command Panel” in the “View” menu of the **“Executive Window”** (section 3.7.2).

You have to describe in the macro file(s) each button individually. You can also request the macro(s) execution in your “pawlogon.kumac” file so that the panel(s) will be automatically displayed at the beginning of the session.

The general syntax of the command “PANEL” for a key definition is:

```
panel x.y command [label] [pixmap]
```

- *x.y* is the key position (column and row number),
- *command* is the complete command (or list of commands) to be executed when the corresponding button is pressed,
- *label* (optional) is an alias-name for this command. If specified, this alias-name is used for the button label (when the appropriate “View” option is selected) instead of the complete command (which is generally too long for a “user-friendly” button label).
- *pixmap* (optional) has to be specified for graphical keys (fully described in the next section 3.7.3).

An example of a panel definition is given in figure 3.8.

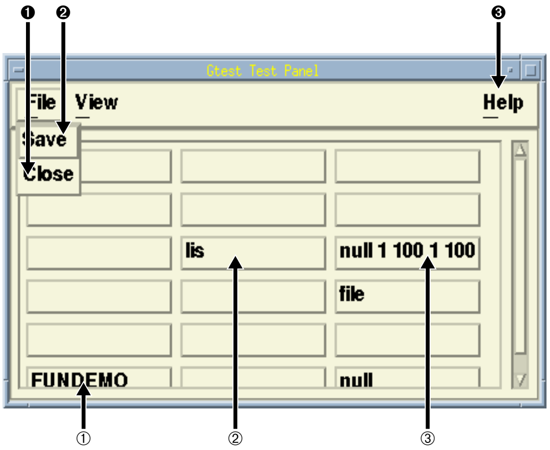
Panel with Graphical keys (Icons) and “View” Selection

As seen in the previous section, the general syntax of the command “PANEL” for a key definition allows the user to define graphical keys (or buttons) where pixmaps are used instead of alpha-numerical labels:

```
panel x.y command [label] [pixmap]
```

The last parameter *pixmap* (optional) is the pixmap to be used for representing the key (button) graphically. If it is specified the graphical representation is displayed by default. It is anyway always possible at run time to ask for an alpha-numerical representation by selecting the appropriate entry in the “View” menu of the panel.

To create a new icon bitmap (or pixmap) one can use the X11 standard bitmap editor “bitmap”. E.g., to get a 20×20 pixel icon called “`m1`”, one can type: `bitmap m1.bm 20x20`. The output file `m1.bm` containing “`#define`



Macro for panel definition

```
*
* MOTIF_PANEL panel_test.kumac
*
panel 0
panel 3.02 'list'
panel 3.03 'null 1 100 1 100'
panel 4.03 'file'
panel 6.01 'FUNDEMO'
panel 6.03 'null'
panel 0 d 'Test Panel' 450x250+600+600
```

❶ Close button (to close panel)
❷ Save button (to save panel into a macro file)
❸ Access to various “helps” on the “**PANEL** interface”

① ② ③ User defined buttons

Figure 3.8: Predefined Panel of Commands

`m1_width 20` ...” has to be referred in the command “/MOTIF/ICON” (with the correct path for the filename), e.g. `/MOTIF/ICON m1 /user/.../.../m1.bm`

The following macro is a general example for a panel definition with graphical keys.

```
*****************************************************
*                                                   *
*              *** panel.kumac ***                  *
*                                                   *
* General example for a panel with icons definition *
*                                                   *
*                                                   *
*****************************************************
*
* Icon bitmaps
*
/motif/icon m1 mk1.bm
/motif/icon m2 mk2.bm
/motif/icon m3 mk3.bm
/motif/icon m4 mk4.bm
/motif/icon m5 mk5.bm
*
* Panel keys definition
* N.B. General syntax:
*      panel r.c command [label] [pixmap]
*      label --> command alias
*                (written in the panel and executed for <Button press>).
*                if <label> (optional) is defined then:
*                   /KUIP/ALIAS/CREATE <label> <command>
*                is automatically generated.
*                if <label> is not defined then "command" is used
*                for button label.
*
panel 0
panel 2.01 null
```

```
panel 2.02 tex_1
panel 3.01 '/example/general kuip.tex tex 1' 'tex_1' m1
panel 3.02 '/example/general kuip.tex tex 2' 'tex_2' m2
panel 3.03 '/example/general kuip.tex tex 3' . m3
panel 3.04 '/example/general kuip.tex tex 4' . m4
panel 4.01 ' ' . m5
panel 4.02 'tex_5' . m5
panel 5.01 '/example/general kuip.tex tex 6' . sm_menu
panel 5.02 '/example/general kuip.tex tex 6' . big_menu
panel 6.01 '/example/general kuip.tex tex 7' 'tex_7'
panel 6.02 '/example/general kuip.tex tex 7' 'tex_7' m1
panel 0 d 'Marker Types' 300x300+500+500
```

Figure 3.9 shows the panel defined in the macro listed above with different "View(ing)" options. In the first window (top/right) the "View" menu is displayed, with the different possibilities which are offered to the user to see the same panel in different ways.

Panel Edition and Saving

All the panels (new or predefined) can be edited interactively. Clicking with the left mouse button on a panel button removes its definition. Clicking with the right mouse button on an empty panel button the user will be asked to give a definition to this button (figure 3.10).

The PANEL commands needed to recreate a panel can be automatically saved into a macro file by pressing the "Save" button ❷ (Fig. 3.8). The panel configuration with its current size and position (which can be modified interactively) is kept into the macro. Panels can be reloaded either by executing the command 'PANEL 0 D' or by pressing the "Command Panel" button in the "View" menu of the **"Executive Window"** and entering the corresponding macro file name.

Some characters in the panel keys/buttons have a special meaning:

– The dollar sign inside a key is replaced by additional keyboard input. For example:

```
'V/PRINT V($)'    | entering 11:20 will execute V/PRINT V(11:20)
```

– Keys ending with a double minus sign make an additional request of keyboard input. For example:

```
V/PRINT V--'     | entering AB will execute V/PRINT VAB
```

"Multi˙panel" or Palette of panels Definition

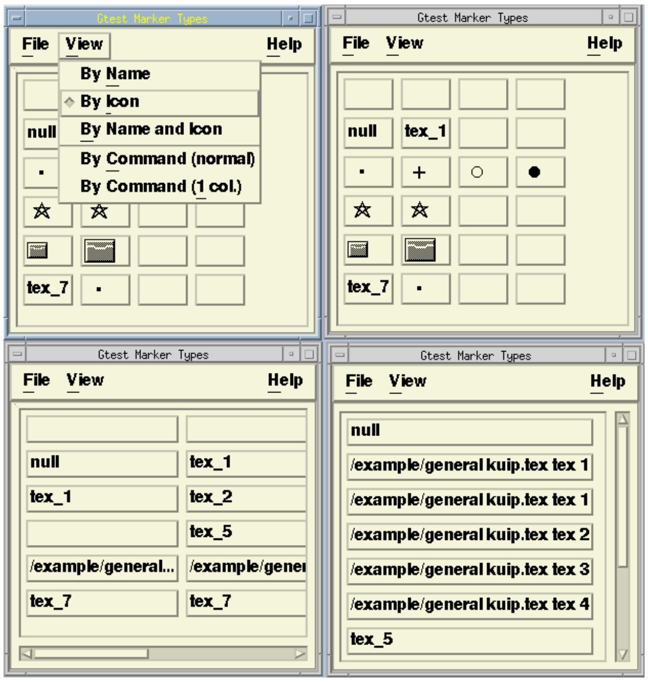
It may be nice or more user-friendly to group a certain number of panels (related to similar actions or objects to be manipulated) in a so-called "palette" of panels. This is possible with the command "`MULTI_PANEL`" which opens such a widget. [4]

```
/MOTIF/MULTI_PANEL [title] [geometry]
```

E.g. `MULTI_PANEL 'My Palette' '200x100+0+0'` will display a "multi˙panel" widget with title "My Palette" and geometry "200x100+0+0" (Position=0,0 in X and Y, width=200, height=100). When this command is executed all panel definitions and executions will go into this "multi˙panel" (or palette) widget. This can be done simply by executing macro(s) containing your panel definition(s), or by selecting the "Add button" entry in the menu "File" available in the "multi˙panel" widget. To terminate a "multi-panel" setting one just have to type: `MULTI_PANEL end`. This means that the following panel definitions and executions will be displayed as individual panels and will not go into this "palette" anymore, unless another palette is opened (by executing again the command "MULTI˙PANEL"). Then the panels will go into that new palette.

The following sequence of commands (which can be put inside a macro) can be used to set up a palette:

[4] For those who are familiar with the "UIMX" User Interface Management System, this is an emulation of the "Palette" widget which is built-in inside this program.



``By Name'' (bottom left): The panel is displayed with alphanumeric labels. If the alias-name "label" is specified in the "panel" command it is used for the button label, otherwise the complete command is displayed.

``By Icon'' (top right): The panel is displayed with graphical labels (icons), if "pixmap" is specified in the "panel" command. Otherwise "label" or the complete command are used instead (no graphical representation). This "view" setting is the default one (the setting can be changed interactively at run time, and the default setting can be changed with the appropriate resource in the ".Xdefaults" for each user individually).

``By Name and Icon'': The panel is displayed with both alphanumeric and graphical (if any) labels. (Not yet implemented ...).

``By Command (normal)'' The panel is displayed with the complete command names. The arrangement of the buttons stay the same (which might not be very convenient ... See below).

``By Command (1 col.) (bottom right): The panel is displayed with the complete command names BUT the arrangement of the buttons is modified: all buttons are displayed on one column, and "blank" buttons are suppressed (this can save a lot of space, and is more user-friendly, for this kind of viewing option).

Figure 3.9: Panel "View" Selection

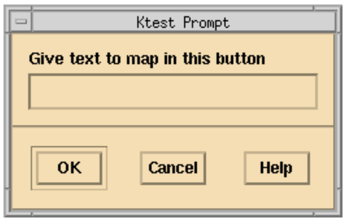
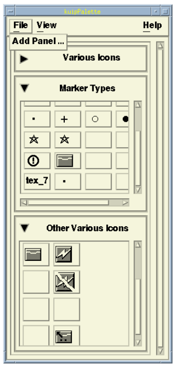


Figure 3.10: Interactive panel button definition



User-defined "palette" with 3 panels :

"Various Icons" : this panel is not displayed (arrow turned left to right) at the moment. One would just have to press the arrow button to make it visible ...

"Marker Types" : this user-defined panel is visible (arrow turned top to down). One can turned it off by pressing the arrow button.

"Other Various Icons" : this user-defined panel is also visible.

Figure 3.11: Multi˙panel (or Palette)

```
MULTI_PANEL
EXEC PANEL1.KUMAC
EXEC PANEL2.KUMAC
EXEC PANEL3.KUMAC
MULTI_PANEL end
```

N.B. `panel1.kumac`, `panel2.kumac`, and `panel3.kumac` are macro files with “usual” panel setting and definition.

Figure 3.11 shows an example of a user-defined palette (with some predefined panels). The “arrow buttons” can be pressed either to reduce the panel to a label containing the panel title (arrow button is then turned left to right) or to display it (arrow button turned up to down). One can see that the “palette” is a good way to have many panels defined and save space on the screen.

3.7.4 X-Windows Resources

X-Windows resources control the appearance and behavior of an application. PAW resources be can redefine them by specifying new values in the standard X11 way : i.e. by editing the “.Xdefaults” file or the system wide “`/usr/lib/X11/app-defaults/<appl_class>`”.

Each new resource has to be specified on a separate line. The syntax for editing one specific resource is always the following:

`<appl._class>*<resource_name>: <resource_value>`

where:

- “appl.˙class” has to be replaced by “Paw++”.
- “resource˙value” is the value to be given to the corresponding “resource˙name”. It can be an integer, a boolean value, a color, a font, or any kind of predefined syntax (e.g. for geometry).

The following is a non exhaustive list of the most important or frequently used X-Windows resources. The default values are put inside “[]”.

- Background and foreground color for all windows (except KXTERM):

  `...*background: ...`

  `...*foreground: ...`
- Geometry ([width]x[height]+[xpos]+[ypos]) of the **“Executive Window”** (KXTERM):

  `...*kxtermGeometry: ... [650x450+0+0]`
- Geometry of the Browser(s):

  `...*kuipBrowser_shell.geometry: ... [-0+0] (1) or [+0+485] (2)`

  (1) without any graphics window - (2) with graphics window(s) managed by HIGZ.
- Geometry of the Graphics Window(s) (if any):

  `...*kuipGraphics_shell.geometry: ... [600x600-0+0]`
- Character font for menus, buttons and dialog area:

  `...*fontList: ... [-adobe-helvetica-bold-r-normal--12-120-75-75-p-70-iso8859-1]`
- Character font for the **Input Pad** and **Transcript Pad** (KXTERM):

  `...*kxtermFont: ... [*-courier-medium-r-normal*-120-*]`
- Character font for the “HELP” windows:

  `...*helpFont: ... [*-courier-bold-r-normal*-120-*]`
- Character font for all “Text” widgets:

  `...*XmText*fontList: ...`

  `...*XmTextField*fontList: ...`
- Character font for the icon labels in the browser(s) **“Object window”**:

  `...*dirlist*fontList: ...`

– Background and foreground colors for the **“Object window”** in browser(s):

```
...*dirlist*background:  ...
...*dirlist*foreground:  ...
```

– Background and foreground colors for the icons associated to the object class “objclass”:

```
...*dirlist*<objclass>*iconBackground:  ...  [white]
...*dirlist*<objclass>*iconForeground:  ...  [black]
```

– Background and foreground colors for the icon-labels associated to the object class “objclass”:

```
...*dirlist*<objclass>*iconLabelBackground:  ...  [white]
...*dirlist*<objclass>*iconLabelForeground:  ...  [black]
```

– Possibility to turn on/off the zooming effect when traversing directories structures inside the browser(s):

```
...*zoomEffect:  ...  [on]
```

– Speed of the zooming effect in the browser(s) when turned on:

```
...*zoomSpeed:  ...  [10]
```

– Double click interval in milliseconds (time span within which 2 button clicks must occur to be considered as a double click rather than two single clicks):

```
...*doubleClickInterval:  ...  [250]
```

– Background and foreground colors for the **“Browsable window”** in browser(s):

```
...*fileList*background:  ...
...*fileList*foreground:  ...
```

– Focus policy:

```
...*keyboardFocusPolicy:  ...
```

If “explicit” focus is set by the mouse or a keyboard command. If “pointer” focus is determined by the mouse pointer position.

The appearance and behavior of the **“Executive Window”** are managed by “KXTERM” whose class-name is “KXterm”. It means that, for instance, to change the background and foreground color of the **“Executive Window”**, one has to override the following resources:

```
KXterm*background:  ...
KXterm*foreground:  ...
```

Concerning the appearance of the built-in icons (browsers for “Commands”, “Files” and “Macro”), the classes of objects which are currently predefined are:

```
Cmd        -- Command
InvCmd     -- Deactivated command
Menu       -- Menu tree
MacFile    -- Macro File
RwFile     -- Read-write file
RoFile     -- Read-only file
NoFile     -- No access file
ExFile     -- Executable file
DirFile    -- Directory
DirUpFile  -- Up directory (..)
```

3.8 Nitty-Gritty

3.8.1 System dependencies

PAW tries to provide as far as possible a homogeneous environment across different operating systems and hardware platforms. Here we want to summarize the remaining system-dependencies. To a large extend the comments made on Unix apply also to the MS-DOS and Windows/NT implementations.

SHELL command

The `SHELL` command allows to pass a command line to the underlying operating system for execution. If used without arguments the `SHELL` command suspends PAW and allows to enter OS commands interactively. When leaving the subprocess, either with the command `return` or `exit` depending on the system, PAW resumes execution.

Unix The command `HOST_SHELL` defines the shell to be invoked. The start-up value is taken from the environment variable `SHELL` or set to an appropriate default such as `/bin/sh`. On some Unix implementations the `SHELL` command can fail if there is not enough free swap space to duplicate the current process.

VMS The `SHELL` command spawns a subprocess with a DCL command processor. This is notoriously slow and there is no way to combine several DCL commands into one `SHELL` command.

EDIT command

The `EDIT` command allows to edit a file without leaving the PAW. The command `HOST_EDITOR` defines the editor to be invoked. The start-up value is taken from the environment variables `KUIPEDITOR`, `EDITOR`, or set to a system dependent default.
`HOST_EDITOR` sets the shell command (sans filename) for starting the editor. Some values have a system dependent special meaning.

Unix The default editor is `vi`. The shell command containing a "`&`" does not necessarily mean that the editor will run as a background process (see section 3.8.2).

VMS The special names `EDT` and `TPU` use the callable interface to these two editors. The startup time is much less than, for example `EDIT/TPU` which spawns a subprocess. However, there is a problem with the callable `EDT`. If any error condition occurs (invalid filename etc.) the callable `EDT` will be unusable for the rest of the session.

Exception handling

PAW installs a signal handler in order to catch exceptions and return to the command input prompt. The command "`BREAK OFF`" disables the signal handler, i.e. PAW aborts in case of an exception. For some systems "`BREAK ON`" allows to request a traceback of where the exception has happened.
There are two major types of exceptions caught by the signal handler. Program exceptions indicate either a bug in PAW or insufficient protection against invalid user input:

Floating point exceptions are caused by divide by zero, floating point overflow, square root of negative numbers etc. Floating point underflows are usually silently ignored and the result is treated as being zero.

Segmentation violation indicates an attempt to read or write a memory location outside the address space reserved by the process, e.g. if an array index is out of bounds. In C code it is most often caused by dereferencing a `NULL` pointer which is prohibited on many systems.

Bus error is usually caused by an unaligned access. Most RISC processors have strict requirements for properly aligned data.

Illegal instruction can mean that PAW tries to executed data as code, for example if the return address on the stack has been overwritten.

Don't be surprised if PAW shows irregular behavior after an exception!
The second type of exceptions handled by the PAW signal handler are user breaks. Hitting the break key (usually `Ctrl-C`) aborts a running command and returns to the input prompt.

Unix The actual break key can be changed with the Unix command `stty`. The default setup usually is "`stty intr ^C`". Unix provides a second kind of keyboard interrupt which is intentionally **not** caught by the PAW signal handler to allow killing run-away processes. A convenient setting is "`stty quit '\\'`"
User break interception does not work for Windows/NT. Tell Microsoft that signal handlers are pretty useless if they are not allowed to use `printf` and `longjmp`.

VMS The user break key is `Ctrl-C`. `Ctrl-Y` is treated like `Ctrl-C`, i.e. it does not bring up the DCL prompt.

3.8.2 The edit server

By default editing from within a PAW is synchronous, i.e. PAW is suspended until the editor terminates. On a workstation this is an inconvenient restriction because the editor can run in a separate window while PAW continues to accept commands.

To take care of this problem PAW provides a facility called the "edit server". Instead of calling the editor directly, PAW starts the editor server as a background process which leaves PAW ready to accept more commands. The server invokes the editor and waits for it. When the editor terminates the server informs PAW about the file which is ready.

The processing routine cannot be called at the very instant the file is ready. PAW waits until the user hits the `RETURN`-key to execute the next command. The file is then checked in *before* the command just entered is executed.

As a protection especially for users working alternately on a terminal or on a workstation PAW does not try asynchronous editing if one of the following conditions is missing:

- The edit server module `kuesvr` must be found in the search path.
- The editor command set by `HOST_EDITOR` must end with an ampersand ("`&`").
- The environment variable `DISPLAY` must be set.

Note that the editor command must create its own window, possibly by wrapping the editor into a terminal window. For convenience "`HOST_EDITOR 'vi &'`" is interpreted automatically as "`xterm -e` *`cmd`* `&`".

Some Unix windowing editors tend to fork themselves as a detached process by default. For example the `jot` editor found on Silicon Graphics systems requires a special option "`-noFork`". Otherwise the edit server and PAW think that the editor has already terminated leaving the file unchanged.

In Paw++ it is essential to use the edit server mechanism. Otherwise invoking the editor from a pop-up menu freezes the screen when the right-hand mouse button is pressed before the subprocess terminates.

The screen can only be unlocked by logging in remotely and killing the PAW.

For asynchronous editing on VMS either the Motif version of TPU must be used or the hosteditor command must create its own terminal window, e.g.

```
HOST_EDITOR TPU/DISPLAY=MOTIF
HOST_EDITOR 'CREATE/TERM/WAIT EDT'
```

Chapter 4: Vectors

Vectors are named arrays of numerical data, memory resident, which can be created during a session, loaded from HBOOK objects, typed in by hand, read from disk files, operated upon using the full functionality of SIGMA or COMIS. Vectors can be used to produce graphics output, and, if necessary, stored away on disk files for further usage. Vectors provide a very convenient mechanism to transport numerical information between different PAW objects, and to manipulate mathematically their content. 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). They can be handled by using `VECTOR/...` commands.

Simple arithmetic operations can be applied to vectors. In addition, as SIGMA is part of PAW, powerful array manipulation operations are available, through the SIGMA, `$SIGMA` and `APPLICATION SIGMA` commands (see section 5.1 on page 67).

4.1 Vector creation and filling

A vector is **created** either by the **PAW command** `VECTOR/CREATE`, by the **SIGMA function** `ARRAY`. or by the **COMIS statement** `VECTOR`.

Example of vector creation

```
VECTOR/CREATE  X(100)      will create a 100-components vector, values = 0.
SIGMA X=ARRAY(100,1#100)   will create a 100-components vector and assign
                            to each element the values 1,2,...100
VECTOR X(100)              in a COMIS routine creates a 100-components vector
                           and initialises each element to zero
```

Once the vector is created, it can be manipulated using the following PAW commands:

`VECTOR/INPUT vlist`	Input from the terminal values into the vector elements specified by the list `vlist`.
`VECTOR/READ vlist`	Values can be **read in** from a file into the vector elements specified by the list vlist.
`VECTOR/COPY v1 v2`	Values in `v1` are copied into `v2`.
`VECTOR/WRITE vlist`	Values in the vector elements specified by the list vlist can be **saved** on a file.
`VECTOR/PRINT vlist`	Values of the vector elements specified in `vlist` will be **printed** on the terminal.
`VECTOR/LIST`	A **list** of existing vectors and their **characteristics** is printed on the terminal.
`VECTOR/DELETE`	Allows global or selective deletion of vectors.

4.2 Vector addressing

Indexing of vectors is possible. The indexing permitted in PAW can be considered as a superset of that permitted by FORTRAN.

Example of vector indices

Vec	`for all elements`
Vec(13)	`for element 13`
Vec(12:)	`for elements 12 up to the last`
Vec(:10)	`for elements 1 to 10`
Vec(5:8)	`for elements 5 to 8`

Sub-elements of the two-dimensional vector `Vec(3,100)` (3 columns by 100 rows) may be addressed by:

Using two-dimensional vectors

Vec(2,5:8)	`for elements 5 to 8 in column 2`
Vec(2:3,5:8)	`for elements 5 to 8 columns 2 to 3`
Vec(2,5)	`for element 5 in column 2`
Vec(:,3)	`for all elements in row 3`
Vec(2)	`for all elements in the 2-nd column (SPECIAL CASE)`

4.3 Vector arithmetic operations

A number of basic vector arithmetic operations is available:

```
VBIAS v1 bias v2       v2(I) = bias + v1(I)
VSCALE v1 scale v2     v2(I) = scale * v1(I)
VADD v1 v2 v3          v3(I) = v1(I) + v2(I)
VMULTI v1 v2 v3        v3(I) = v1(I) * v2(I)
VSUBTR v1 v2 v3        v3(I) = v1(I) - v2(I)
VDIVID v1 v2 v3        v3(I) = v1(I) / v2(I), if v2(I)<>0
```

In all operations only the minimum vector length is considered, i.e. an operation between a vector `A` of dimension 10 and a vector `B` of dimension 5 will involve the first 5 elements for both vectors. If the destination vector does not exist, it is created with the same length as specified in the source vector.

4.4 Vector arithmetic operations using SIGMA

A more complete and convenient mechanism for the mathematical manipulation of entire vectors is provided by SIGMA. SIGMA-generated arrays are stored as PAW vectors and therefore are accessible to PAW commands, and PAW vectors are accessible to SIGMA. The facilities available via SIGMA are described in the next chapter.

4.5 Using vectors in a COMIS routine

The declaration `VECTOR vector_name` may be used inside a COMIS routine to address a PAW vector. If the vector does not exist, it is created with the specifications provided by the declared dimension.

4.6 Usage of vectors with other PAW objects

Vectors can be used to transport numerical information between different PAW objects, and to manipulate mathematically their content.

`VECTOR/HFILL VNAME ID`	Each vector element of `VNAME` is used to fill the existing histogram `ID`.
`HISTOGRAM/GET_VECTOR/CONTENT`	Provides an interface between vectors and histograms.
`HISTOGRAM/PUT_VECTOR/CONTENT`	Provides an interface between histograms and vectors.

4.7 Graphical output of vectors

`VECTOR/DRAW VNAME`	Interprets the content of the vector `VNAME` as a histogram contents and draw a **graph**.
`VECTOR/PLOT VNAME`	Vector elements are considered as individual values to be entered into a histogram and a graph is produced. If `VNAME` is the name of a vector, then each vector element of `VNAME` is used to fill a histogram which is automatically booked with 100 channels and plotted. If `VNAME` has the form `VNAME1%VNAME2` then a scatter-plot of vector `VNAME1` versus `VNAME2` is plotted.

A number of graphical primitives are available in PAW. Those directly related to the graphical output of vectors are:

`GRAPH N X Y`	Draw a curve through a set of points defined by arrays `X` and `Y`.
`HIST N X Y`	Draw an histogram defined by arrays `X` and `Y`.
`PIE X0 Y0 RAD N VAL`	Draw a pie chart, of `N` slices, with size of slices given in `VAL`, of a radius `RAD`, centered at `X0`, `Y0`.

4.8 Fitting the contents of a vector

A user defined (and parameter dependent) function can be fitted to the points defined by the two vectors `X` and `Y` and the vector of associated errors `EY`. The general syntax of the command to fit vectors is:

`VECTOR/FIT x y ey func [ chopt np par step pmin pmax errpar ]`

For more information have a look at the online help of this command in PAW.

Chapter 5: SIGMA

5.1 Access to SIGMA

The SIGMA array manipulation package can be accessed in three different ways in PAW:

Precede the statement by the prefix SIGMA

Example

```
PAW > SIGMA xvec=array(100,-pi#pi*2)
PAW > SIGMA y=sin(xvec)*xvec
```

Note the use of the predefined constant `PI` in SIGMA with the obvious value.

The PAW command: APPLication SIGMA

All commands typed in after this command will be directly processed by SIGMA. The command `EXIT` will return control to PAW, e.g.

```
PAW > APPLication SIGMA
SIGMA > xvec=array(100,-pi#pi*2)
SIGMA > sinus=sin(xvec)*xvec
SIGMA > cosinus=cos(xvec)*xvec
SIGMA > exit
PAW > vector/list
 Vector Name                  Type    Length    Dim-1    Dim-2    Dim-3

 XVEC                            R       100      100
 SINUS                           R       100      100
 COSINUS                         R       100      100

 Total of  3 Vector(s)
```

The PAW system function $SIGMA

The expression to be evaluated must be enclosed in parentheses. The function will return the numerical value of the expression (if the result is a scalar) or the name of a temporary vector (if the result is a vector).
Assuming that the computation of the function `sin(x)*x` in the above example would be only for the purpose of producing a graph, (i.e. the result is not needed for further calculations), then one could just have typed the following commands:

```
PAW > SIGMA xvec=array(100,-pi#pi*2)
PAW > GRAph 100 xvec $SIGMA(SIN(XVEC)*XVEC)
```

5.2 Vector arithmetic operations using SIGMA

A complete and convenient mechanism for the mathematical manipulation of vectors is provided by SIGMA. In the following, we use the words “array” and “vector” as synonyms. In both cases, we refer to PAW vectors, in the sense that SIGMA offers an alternative way to generate and to manipulate PAW vectors (see section 4 on page 64).
The notation of SIGMA is similar to that of FORTRAN, in the sense that is based upon formulae and assignment statements.
The special operator `ARRAY` is used to generate vectors:

```
vname  =  ARRAY  (arg1,arg2)
```

`vname` Name of the vector (array) being created.

`arg1` Defines the array **structure**, i.e. the Number of COmponents (`NCO`) of the array.

`arg2` Provides the **numerical values** filling the array row-wise.
If `arg2` is absent (or does not provide enough values) the array is filled with 1.

5.2.1 Basic operators

`+`	Add
`-`	Subtract
`*`	Multiply
`/`	Divide
`**`	Exponentiation
`&`	Concatenation

Note that ill defined operations will give `0.` as result. For instance: a division by zero gives zero as result.

5.2.2 Logical operators

Logical operators act on entities that have **Boolean** values `1` (true) or `0` (false). The result is Boolean.

`AND`	Logical operation AND
`NOT`	Logical operation NOT
`OR`	Logical operation OR
`EQ`	EQual to
`GE`	Greater or Equal to
`GT`	Greater Than
`LE`	Less or Equal to
`LT`	Less Than
`NE`	Not Equal

5.2.3 Control operators

`!PRINT`	Provides the automatic printing of every newly created array or scalar.
`!NOPRINT`	Suppresses the automatic printing of every newly created array or scalar.

Examples

```
A=ARRAY (6,1#6)             1  2  3  4  5  6
A=ARRAY (4)                 1  1  1  1
A=ARRAY (5,2&3&-1&2&1.2)    2  3  -1  2  1.2
A=ARRAY (3)*PI              3.1415927  3.1415927  3.1415927
A=ARRAY (1,123E4)           1230000.0
```

5.3 SIGMA functions

SIGMA provides some functions which perform a task on a whole array. These functions have no analogues in FORTRAN because all FORTRAN functions operate on one or more single numbers. Presently available SIGMA functions are listed in table 5.1 below.

5.3.1 SIGMA functions - A detailed description.

In the following description of the SIGMA functions, the letter `R` always denotes the **result** and `arg` denotes one or more **arguments**. Any argument may itself be an expression. In that case `arg` means the result of this expression. Let `OP` denote any of the above array functions, then the statement:

```
R = OP  (arg1,arg2,...)
```

produces `R` without doing anything to the contents stored under the names appearing in `arg1,arg2,...`. Thus, although in the description we may say "...`OP` does such and such to `arg` ...", in reality it leaves `arg` intact and works on the argument to produce `R`.

Name	Result	Explanation
ANY	Scalar	The result is a Boolean scalar of value 1 (true) if at least one component of the argument is true and 0 (false) otherwise.
DEL	Vector	Analog to the **Dirac-DELta Function**. V1=DEL(V) sets each element of V1 to 0.0 (if corresponding element in V is **non-zero**) or to 1.0 (if corresponding element is **zero**).
DIFF	Vector	V2=DIFF(V) **forward difference** of V. The rightmost value in V1 is obtained by quadratic extrapolation over the last three elements of V.
LS	Vector	V1=LS(V,N) **shifts** index of V to the left by N steps (cyclic).
LVMAX	Scalar	S1=LVMAX(V1) sets S1 equal to the **index** (location) of the **maximum** value in vector V1.
LVMIM	Scalar	S1=LVMIN(V1) sets S1 equal to the **index** (location) of the **minimum** value in vector V1.
MAX	Vector	V3=MAX(V1,V2) sets each element of V3 equal to the **maximum** of the corresponding elements in V1 and V2.
MAXV	Vector	V1=MAXV(V) sets each element of V1 equal to the **maximum** value in V.
MIN	Vector	V3=MIN(V1,V2) sets each element of V3 equal to the **minumum** of the corresponding elements in V1 and V2.
MINV	Vector	V1=MINV(V) sets each element of V1 equal to the **minimum** value in V.
NCO	Scalar	V1=NCO(V) **Number of COmponents** of vector of V.
ORDER	Vector	V1=ORDER(V,V2) finds a permutation that brings V2 in a non-descending order and applies it to V to generate V1.
PROD	Vector	V1=PROD(V) V1 is the **running product** of V.
QUAD	Vector	V2=QUAD(V1,H) The quadrature function QUAD numerically integrates each row of V1 with respect to the scalar step size H.
SUMV	Vector	V2=SUMV(V1) **running sum** of V.
VMAX	Scalar	S1=VMAX(V1) sets S1 equal to the **maximum** value in vector V1.
VMIN	Scalar	S1=VMIN(V1) sets S1 equal to the **minimum** value in vector V1.
VSUM	Scalar	S1=VSUM(V) **sum** of all components of V.

Table 5.1: SIGMA functions

```
R  =  ANY  (arg)
```

The function ANY considers the result of the argument expression as a Boolean array. SIGMA represents “true” by 1 and “false” by 0. Thus the components of arg must be either 0 or 1, otherwise an error is generated.

If at least one component of the result of the argument expression is 1, than ANY returns the scalar 1. If all components of the result of the argument expression are 0 then ANY returns the scalar 0. If arg is a Boolean scalar, R = arg.

Example of the ANY command

```
PAW > APPL SIGMA
SIGMA > !PRINT                             | Print newly created vectors and scalars
SIGMA > W=(-2)**ARRAY(10,1#10)
NCO(W       )=   10
W       =
 -2.000      4.000     -8.000      16.00     -32.00      64.00
 -128.0      256.0     -512.0      1024.
SIGMA > X=W GT 0
NCO(X       )=   10
X       =
 0.0000      1.000     0.0000      1.000     0.0000      1.000
```

```
 0.0000      1.000      0.0000      1.000
SIGMA > R=ANY(X)
NCO(R       )=    1
R         1.000
```

R = **DEL** (arg)

DEL is a discrete analogue of a **Dirac delta function**. DEL works independently on each row of the argument array. If the elements of any row of the argument are denoted by $X_1,\ X_2,\ \ldots,\ X_i,\ \ldots,\ X_n$ then the corresponding row of the result of the delta function operation will be $Z_1,\ Z_2,\ \ldots,\ Z_i,\ \ldots,\ Z_n$ where all $Z_i = 0$ except in three cases, in which $Z_i = 1$, namely:

1 When the component X_i is itself zero.

2 When $X_{i-1},\ X_i$ are of opposite sign and $|X_i| < |X_{i-1}|$ If $i = 1$ then linear extrapolation to the left is used.

3 When $X_i,\ X_{i+1}$ are of opposite sign and $|X_i| \leq |X_{i+1}|$ If $i = 1$ then linear extrapolation to the right is used.

If arg is a scalar, the value of DEL(arg) will be 1 if arg is zero, and 0 otherwise.

Example of the del command

```
SIGMA > W=array(11,-1#1)
NCO(W       )=   11
W        =
 -1.000     -0.8000     -0.6000     -0.4000     -0.2000     -0.2980E-07
 0.2000      0.4000      0.6000      0.8000       1.000

SIGMA > X= (W+1.01)*W*(W-.35)*(W-.42)
NCO(X       )=   11
X        =
-0.1917E-01 -0.2357    -0.2384    -0.1501    -0.5524E-01-0.4425E-08
 0.7986E-02 -0.5640E-03 0.4347E-01 0.2476     0.7578

SIGMA > R=del(x)
NCO(R       )=   11
R        =
  1.000      0.0000      0.0000      0.0000      0.0000       1.000
 0.0000       1.000      0.0000      0.0000      0.0000
```

R = **DIFF** (arg)

The DIFF function generates the **forward difference** of each row of the argument array, say $X_1,\ X_2,\ \ldots,\ X_i,\ \ldots,\ X_n$ and creates an array with components equal to the forward difference of X: $X_2 - X_1,\ X_3 - X_2,\ \ldots,\ X_n - X_{n-1}, X_0$ where the rightmost value X_0 is obtained by quadratic extrapolation over the last three elements of the result of arg. Applied to a scalar DIFF gives a zero result.

Example of the DIFF command

```
SIGMA > x=array(6,5#0)
NCO(X       )=    6
X        =
  5.000      4.000      3.000      2.000      1.000      0.0000
SIGMA > Y=x*x
NCO(Y       )=    6
Y        =
  25.00      16.00      9.000      4.000      1.000      0.0000
SIGMA > Z=Diff(Y)
NCO(Z       )=    6
Z        =
 -9.000     -7.000     -5.000     -3.000     -1.000       1.000
```

R = **LS** (arg1,arg2)

The `LS` rearrangement function performs a **left shift**. `arg1` is the array to be shifted; `arg2` must be a scalar value (rounded if necessary by the system), interpreted as the number of places the array has to be shifted to the left. The scalar `arg2` can be negative, in which case `LS` shifts to the right a number of places equal to the absolute value of `arg2`.

It should be noted the the shift is performed circularly `modulo N`, where `N` is the number of components in the rows of the array to be shifted. Hence, `LS(X,N+1)` shifts the `N` component rows of `X` by `1` to the left, and `LS(X,-1)` shifts the rows by `N-1` to the left (or by `1` to the right). If `arg1` is a scalar, `R = arg1`.

Example of the left shift command

```
SIGMA > X=array(4&5,array(20,1#20))
NCO(X       )=    4    5
X       =
  1.000       2.000       3.000       4.000
  5.000       6.000       7.000       8.000
  9.000       10.00       11.00       12.00
  13.00       14.00       15.00       16.00
  17.00       18.00       19.00       20.00

SIGMA > y=ls(x,1)

NCO(Y       )=    4    5
Y       =
  2.000       3.000       4.000       1.000
  6.000       7.000       8.000       5.000
  10.00       11.00       12.00       9.000
  14.00       15.00       16.00       13.00
  18.00       19.00       20.00       17.00

SIGMA > y=ls(x,-3)
NCO(Y       )=    4    5
Y       =
  2.000       3.000       4.000       1.000
  6.000       7.000       8.000       5.000
  10.00       11.00       12.00       9.000
  14.00       15.00       16.00       13.00
  18.00       19.00       20.00       17.00

SIGMA > X=array(5,1#5)
NCO(X       )=    5
X         1.000       2.000       3.000       4.000       5.000
SIGMA > z=ls(x,3)
NCO(Z       )=    5
Z         4.000       5.000       1.000       2.000       3.000
SIGMA > z1=ls(x,-4)
NCO(Z1      )=    5
Z1        2.000       3.000       4.000       5.000       1.000
```

R = **LVMAX** (arg1) and R = **LVMIN** (arg1)

The functions `LVMAX` and `LVMIN` returns as a scalar result the index (position) of the largest or smallest element, respectively, in the argument array.

Example of using the `LVMAX` and `LVMIN` commands

```
SIGMA >  x=sin(array(10,1#10))
NCO(X       )=   10
X       =
```

```
 0.841     0.909     0.141    -0.757    -0.959    -0.279     0.657
 0.989     0.412    -0.544

SIGMA > r=lvmax(x)
NCO(R       )=    1
R          8.00
```

R = **MAX** (arg1,arg2) and R = **MIN** (arg1,arg2)

The functions `MAX` and `MIN` work independently on each element of their arguments. `arg2` can be a scalar. The result has the same dimension as the argument array `arg1` and each element of the result is set equal to the largest or smallest element, respectively, of the corresponding element of the argument arrays.

Example of using the `MAX` and `MIN` commands

```
SIGMA >  x=sin(array(10,1#10))
NCO(X       )=   10
X       =
 0.841     0.909     0.141    -0.757    -0.959    -0.279     0.657
 0.989     0.412    -0.544

SIGMA >   y=cos(array(10,1#10))
NCO(Y       )=   10
Y       =
 0.540    -0.416    -0.990    -0.654     0.284     0.960     0.754
-0.146    -0.911    -0.839

SIGMA > z=min(x,y)
NCO(Z       )=   10
Z       =
 0.540    -0.416    -0.990    -0.757    -0.959    -0.279     0.657
-0.146    -0.911    -0.839
```

R = **MAXV** (arg) and R = **MINV** (arg)

The extrema functions `MAXV` and `MINV` work on each element of their argument and the result has the same dimension as the argument array `arg1`. Each element of of the result is set equal to the largest or smallest element, respectively, of the corresponding row of the argument array.

All these functions, if applied to a scalar argument, yield `R=arg`.

Example of using the `MAX` and `MIN` commands

```
SIGMA > x=array(10,0#10)
NCO(X       )=   10
X       =
 0.0000      1.111      2.222      3.333      4.444      5.556
  6.667      7.778      8.889      10.00

SIGMA > s=sin(x)*x
NCO(S       )=   10
S       =
 0.0000     0.9958      1.767    -0.6352     -4.286     -3.695
  2.494      7.755      4.539     -5.440

SIGMA >  x=minv(s)
NCO(X       )=   10
X       =
 -5.440     -5.440     -5.440     -5.440     -5.440     -5.440
 -5.440     -5.440     -5.440     -5.440
```

```
R  = NCO  (arg)
```

The “Number of COmponents” (`NCO`) control function obtains the `NCO` vector of the `arg`. The `NCO` vector of a scalar is the scalar `1`. For any argument the `NCO(NCO(arg))` gives the number of dimensions of the `arg`.

Using the `NCO` command

```
SIGMA >  x=array(4&3&2,array(24,2#48))
NCO(X       )=    4    3    2
X        =
  2.000      4.000      6.000      8.000
  10.00      12.00      14.00      16.00
  18.00      20.00      22.00      24.00

  26.00      28.00      30.00      32.00
  34.00      36.00      38.00      40.00
  42.00      44.00      46.00      48.00

SIGMA >  r=nco(x)
NCO(R       )=    3
R         4.000      3.000      2.000
SIGMA >  ndim=nco(nco(x))
NCO(NDIM    )=    1
NDIM      3.000
```

```
R = ORDER  (arg1,arg2)
```

The ordering function `ORDER` acts independently on each row of `arg1`. `arg2` must have the same row length as `arg1`.
`ORDER` finds the permutation that brings `arg2` into a non-descending sequence (row-wise) and constructs the result by applying this permutation to `arg1`. It may in some cases be expanded to that structure by using the techniques of the topological arithmetic. This is particularly useful if `arg2` is a single vector with the length of the rows of `arg1`.

Using the `ORDER` command

```
SIGMA > X = 1&1&2&4&-3&1&3
NCO(X       )=    7
X        =
  1.00      1.00      2.00      4.00     -3.00      1.00      3.00
SIGMA > P = ORDER(X,X)
NCO(P       )=    7
P        =
 -3.00      1.00      1.00      1.00      2.00      3.00      4.00
SIGMA > P = ORDER(X,-X)
NCO(P       )=    7
P        =
  4.00      3.00      2.00      1.00      1.00      1.00     -3.00
SIGMA > Y = ARRAY(7,1# 7)
NCO(Y       )=    7
Y        =
  1.00      2.00      3.00      4.00      5.00      6.00      7.00
SIGMA > P = ORDER(Y,X)
NCO(P       )=    7
P        =
  5.00      1.00      2.00      6.00      3.00      7.00      4.00
```

```
R = PROD  (arg)
```

The `PROD` function generates the **running product** of each row of the argument array, say $X_1,\ X_2, \ldots,\ X_n$ and creates an array with components equal to the running product of the component of the argument: $X_1, X_2, \ldots, X_n$ $X_1, X_1 \times X_2,\ \ldots,\ X_1 \times X_2 \times \ldots X_n$

Using the `TIMES` command

```
SIGMA > x=array(6&4,array(24,1#24))
NCO(X       )=    6    4
X        =
   1.000       2.000       3.000       4.000       5.000       6.000
   7.000       8.000       9.000       10.00       11.00       12.00
   13.00       14.00       15.00       16.00       17.00       18.00
   19.00       20.00       21.00       22.00       23.00       24.00

SIGMA > y=prod(x)
NCO(Y       )=    6    4
Y        =
   1.000       2.000       6.000       24.00       120.0       720.0
   7.000       56.00       504.0       5040.     0.5544E+05 0.6653E+06
   13.00       182.0       2730.     0.4368E+05 0.7426E+06 0.1337E+08
   19.00       380.0       7980.     0.1756E+06 0.4038E+07 0.9691E+08
```

`R =` **QUAD** `(arg1,arg2)`

The **quadrature function** `QUAD` numerically integrates each row of `arg1` with respect to the scalar step size `h` defined by `arg2`.

The result `R` has the same dimension as `arg1` and the integration constant is fixed by choosing the first point of the result to be zero.

The method uses a four-point forward and backward one-strip-formula based on Lagrange interpolation. We have for the first point of the result:

$$R_1 = \int_{x_1}^{x_1} (arg1)\mathrm{d}x = 0$$

for the second and third points

$$R_{i+1} = R_i + \frac{h}{24}(9f_i + 19f_{i+1} - 5f_{i+2} + f_{i+3})$$

and for all subsequent points

$$R_i = R_{i-1} + \frac{h}{24}(f_{i-3} - 5f_{i-2} + 19f_{i-1} + 9f_i)$$

where the f_i are elements of `arg1` and are assumed to be values of some functions evaluated at equidistant intervals with interval width equal to `h` (`h` being equal to the value of `arg2`).

`R =` **SUMV** `(arg)`

The `SUMV` function generates the **running summation** of each row of the argument array, say $X_1,\ X_2,\ \ldots,\ X_i,\ \ldots,\ X_n$ and creates an array with components equal to the running sum of the X_i namely: $X_1,\ X_1 + X_2,\ \ldots,\ X_1 + X_2 + \ldots X_i,\ \ldots,\ X_1 + X_2 + \ldots X_n$.

Using the `SUM` function

```
SIGMA >  x=array(6&4,array(24,1#24))
NCO(X       )=    6    4
X        =
   1.000       2.000       3.000       4.000       5.000       6.000
   7.000       8.000       9.000       10.00       11.00       12.00
   13.00       14.00       15.00       16.00       17.00       18.00
   19.00       20.00       21.00       22.00       23.00       24.00
```

```
SIGMA > ********************
SIGMA > * SIGMA application *
SIGMA > *  showing use of   *
SIGMA > *   QUAD numeric    *
SIGMA > *   integration     *
SIGMA > ********************
SIGMA > x=array(101,0#2*pi)
SIGMA > * Function value array
SIGMA > y=sin(x)
SIGMA > * Step size
SIGMA > dx=0.6283186E-01
SIGMA > print dx
 NCO(DX      )=    1
 DX       0.6283186E-01
SIGMA > * Integration of SIN(X)
SIGMA > * in interval 0<=X<+2*PI
SIGMA > f=quad(y,dx)
SIGMA > * Analytical result
SIGMA > * is   1-COS(X)
SIGMA > g=1-cos(x)
SIGMA > * Compute the difference
SIGMA > erro=(g-f)*10**6
SIGMA > * Plot the difference
SIGMA > *  in units of 10^-6
SIGMA > exit
PAW > opt GRID
PAW > gra 101 x erro
```

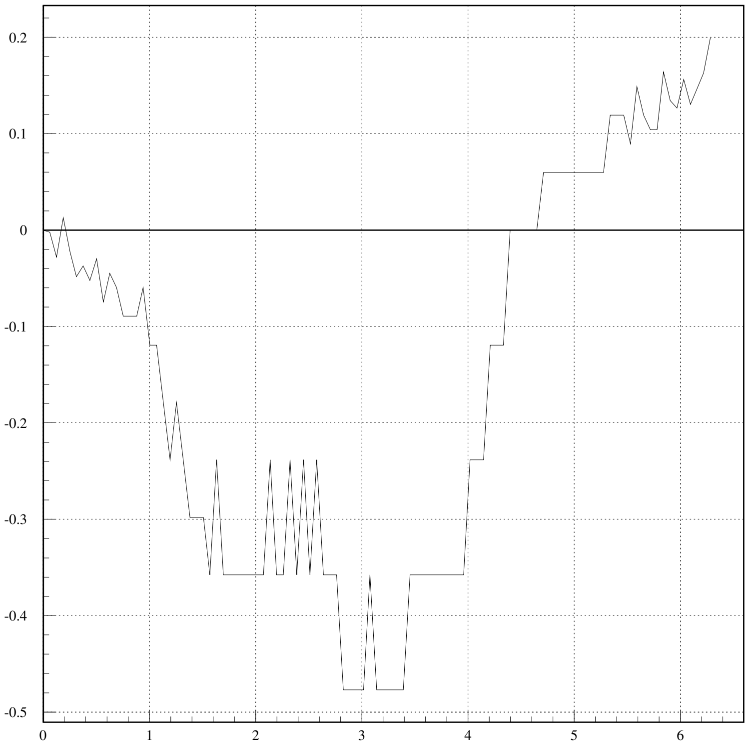


Figure 5.1: Using numerical integration with SIGMA

```
SIGMA > y=sumv(x)
NCO(Y       )=    6    4
Y       =
   1.000      3.000      6.000      10.00      15.00      21.00
   7.000      15.00      24.00      34.00      45.00      57.00
   13.00      27.00      42.00      58.00      75.00      93.00
   19.00      39.00      60.00      82.00      105.0      129.0
```

R = **VMAX** (arg) and R = **VMIN** (arg)

The functions `VMAX` and `VMIN` return a scalar equal to the largest or smallest element of the array `arg`.

R = **VSUM** (arg1)

The `VSUM` function generates the **sum** of each element of the argument array, say $X_1,\ X_2,\ \ldots,\ X_i,\ \ldots,\ X_n$ and creates a scalar whose value is equal to the sum of all the components of X namely: $X_1 + X_2 + X_3,\ \ldots,\ X_n$

Using the `VSUM` function

```
SIGMA > x=array(10)
NCO(X       )=   10
X       =
  1.00      1.00      1.00      1.00      1.00      1.00      1.00
  1.00      1.00      1.00

SIGMA > r=vsum(x)
NCO(R       )=    1
R         10.0
```

5.4 Available library functions

The library functions available under SIGMA are listed below. All these functions have a single argument, unless otherwise indicated. The number indicated between parentheses corresponds to the number of the same function in the CERN program library.

`ABS`	ABSolute value
`ACOS`	ArCOSine
`ALOGAM`	LOGarithm of the GAMma Function (C341)
`ASIN`	ArcSINe
`ATAN`	ArcTANgent
`ATAN2`	ArcTANgent2 (2 arguments)
`BESI0`	Mod. Bessel Function I0 (C313)
`BESI1`	Mod. Bessel Function I1 (C313)
`BESJ0`	Bessel Function J0 (C312)
`BESJ1`	Bessel Function J1 (C312)
`BESK0`	Mod. Bessel Function K0 (C313)
`BESK1`	Mod. Bessel Function K1 (C313)
`BESY0`	Bessel Function Y0 (C312)
`BESY1`	Bessel Function Y1 (C312)
`COS`	COSine
`COSH`	Hyperbolic COSine
`COSINT`	COSine INTegral (C336)
`DILOG`	DILOGarithm Function (C304)
`EBESI0`	$\exp(-\|x\|)I_0(x)$ (C313)
`EBESI1`	$\exp(-\|x\|)I_1(x)$ (C313)
`EBESK0`	$\exp(x)K_0(x)$ (C313)
`EBESK1`	$\exp(x)K_1(x)$ (C313)
`ELLICK`	Complete Elliptic Integral K (C308)
`ELLICE`	Complete Elliptic Integral E (C308)
`ERF`	Error Function ERF (C300)
`ERFC`	Error Function ERFC (C300)
`EXP`	EXPonential
`EXPINT`	EXPonential INTegral (C337)
`FREQ`	Normal Frequency Function FREQ (C300)
`GAMMA`	GAMMA Function (C305)
`INT`	Takes INTegral part of decimal number
`LOG`	Natural LOGarithm
`LOG10`	Common LOGarithm
`MOD`	Remaindering
`RNDM`	Random Number Generator: `V1=RNDM(V)`, with `NCO(V1)=NCO(V)` generates random numbers between `0` and `1`.
`SIGN`	Transfer of SIGN: `V2=SIGN(V,V1)`, `V2=|V|*V1/|V1|`
`SIN`	SINe Function
`SINH`	Hyperbolic SINe
`SININT`	SINe INTegral (C336)
`SQRT`	SQuare RooT
`TAN`	TANgent
`TANH`	Hyperbolic Tangent

Ill defined functions will return `0.` as result. (e.g. `SQRT` of a negative number is taken as `0`).

Chapter 6: HBOOK

6.1 Introduction

Many of the ideas and functionality in the area of data presentation, manipulation and management in PAW find their origin in the HBOOK subroutine package [2], which handles statistical distributions (histograms and Ntuples). HBOOK is normally run in a batch environment, and it produces generally graphics output on the line printer or, optionally, via the HPLOT [7] package on a high resolution graphic output device.

The HBOOK system consists of a few hundred FORTRAN subroutines which enable the user to symbolically define, fill and output one- and two-dimensional density estimators, under the form of **histograms, scatter-plots** and **tables**.

Furthermore the analysis of large data samples is eased by the use of **Ntuples**, which are two-dimensional arrays, characterised by a **fixed** number N, specifying the number of entries per element, and by a **length**, giving the total number of elements. An element of a Ntuple can be thought of as a physics "event" on e.g. a Data Summary Tape (micro-DST). **Selection criteria** can be applied to each "event" or element and a complete Ntuple can be statistically analysed in a fast, efficient and interactive way.

6.1.1 The functionality of HBOOK

The various user routines of HBOOK can be subdivided by functionality as follows:

Booking	Declare a one- or two-dimensional histogram or a Ntuple
Projections	Project two-dimensional distributions onto both axes
Ntuples	Way of writing micro data-summary-files for further processing. This allows to make later projections of individual variables or correlation plots. Selection mechanisms may be defined
Function representation	Associates a real function of 1 or 2 variables to a histogram
Filling	Enter a data value into a given histogram, table or Ntuple
Access to information	Transfer of numerical values from HBOOK-managed memory to Fortran variables and back
Arithmetic operations	On histograms and Ntuples
Fitting	Least squares and maximum likelihood fits of parametric functions to histogramed data
Smoothing	Splines or other algorithms
Random number generation	Based on experimental distributions
Archiving	Information is stored on mass storage for further reference in subsequent programs
Editing	Choice of the form of presentation of the histogramed data

6.2 Basic ideas

The basic data elements of HBOOK are the **histogram** (one- and two-dimensional) and the **Ntuple**. The user identifies his data elements using a **single integer**. Each of the elements has a number of **attributes** associated with it.

The HBOOK system uses the ZEBRA [6] data manager to store its data elements in a COMMON block `/PAWC/`, shared with the KUIP [4] and HIGZ [8] packages, when the latter are also used (as is the case in PAW). In fact the first task of a HBOOK user is to declare the length of this common to ZEBRA by a call to `HLIMIT`, as is seen in the programs shown in Section 6.3[1].

In the `/PAWC/` data store, the HBOOK, HIGZ and KUIP packages have all their own **division** (see [6] for more details on the notion of divisions) as follows (figure 6.1):

[1] This is of course not necessary in PAW, which is already precompiled when it is run. However when treating very large data samples or in other special applications, it might be necessary to specify a different value for the length of the dynamic store, which is defined by a call to `PAWINT` from the main initialisation routine `PAMAIN`. The "default" value for the length of `/PAWC/` is 500000 (Apollo), 200000 (IBM) or 300000 (other systems), with respectively 10000 and 68000 words initially reserved for HIGZ and KUIP.

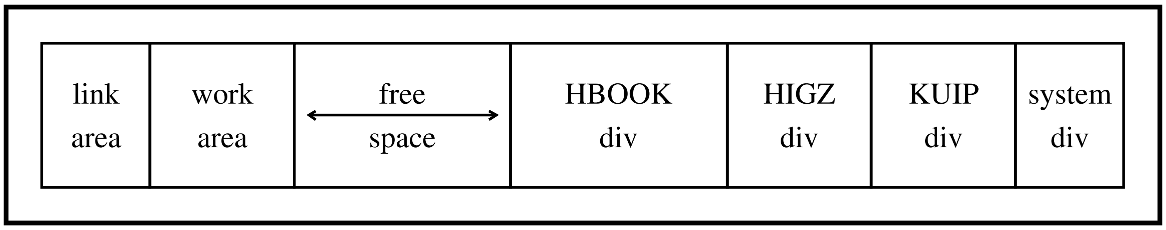


Figure 6.1: The layout of the /PAWC/ dynamic store

`LINKS` Some locations at the beginning of /PAWC/ for ZEBRA pointers.

`WORKS` Working space (or division 1) used by the various packages storing information in /PAWC/

`HBOOK` Division 2 of the store. Reserved to HBOOK

`HIGZ` A division reserved for the HIGZ graphics package.

`KUIP` A division reserved for the KUIP user interface package.

`SYSTEM` The ZEBRA system division. It contains some tables, as well as the Input/Output buffers for HRIN and HROUT.

6.2.1 RZ directories and HBOOK files

An advantage of using ZEBRA in HBOOK is that ZEBRA's direct access **RZ package** is available. The latter allows data structures to be uniquely addressed via **pathnames**, carrying a mnemonic meaning and showing the relations between data structures. Related data structures are addressed from a **directory**. Each time a RZ file is opened via a call to HRFILE a supplementary top directory is created with a name specified in the calling sequence. This means that the user can more easily keep track of his data and also the **same** histogram identifiers can be used in various files, what makes life easier if one wants to study various data samples with the same program, since they can be addressed by changing to the relevant file by a call to HCDIR first.

Example of using directories

```
CALL HRFILE(1,'HISTO1',' ')     ! Open first  HBOOK RZ file (read only)
CALL HRFILE(2,'HISTO2','U')     ! Open second HBOOK RZ file (update)
CALL HCDIR('//HISTO1',' ')      ! Make HISTO1 current directory
CALL HRIN(20,9999,0)            ! Read ID 20 on file 1
  ....
CALL HCDIR('//HISTO2',' ')      ! Make HISTO2 current directory
CALL HRIN(10,9999,0)            ! Read ID 10 on file 2
  ....
CALL HROUT(20,ICYCLE,' ')       ! Write ID 20 to file 2
CALL HREND('HISTO1')            ! Close file 1
CALL HREND('HISTO2')            ! Close file 2
```

In the previous example (and also in the code presented in section 6.3) it is shown how an external file is available via a directory name inside HBOOK (and PAW), and that one can change from one to the other file by merely **changing directory**, via the PAW command CDIR, which calls the HBOOK routine HCDIR.

6.2.2 Changing directories

One must pay attention to the fact that **newly** created histograms go to **memory** in the //PAWC directory (i.e. the /PAWC/ common). As an example suppose that the current directory is //LUN1, and an operation is performed on two histograms. These histograms are first copied to memory //PAWC, the operation is performed and the result is **only** available in //PAWC,

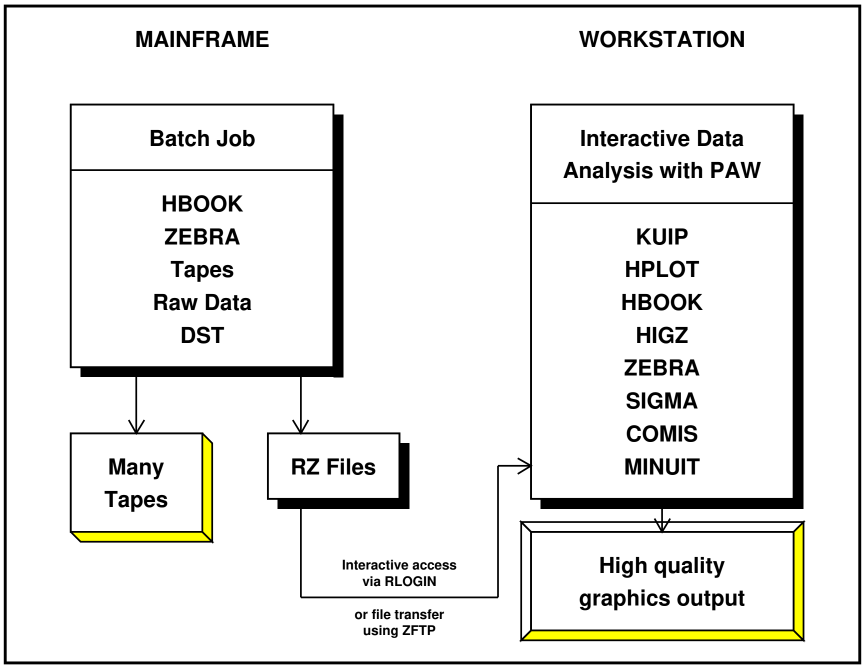


Figure 6.2: Schematic presentation of the various steps in the data analysis chain

```
PAW > CDIR //LUN1            | Set current directory to //LUN1
PAW > ADD 10 20 30           | Add histograms 10 and 20 into 30
                             | Histogram 30 is created in //PAWC
PAW > Histo/Plot //PAWC/30   | Show the result of the sum
PAW > CD //PAWC              | Set the current directory to memory
PAW > Histo/plot 30          | Show the result once more
```

Similarly when histograms or Ntuples are plotted (e.g. by the `HISTO/PLOT` command), they are copied to memory possibly replacing an old copy of the same ID. As long as the copy in memory is not changed, each time the ID is read from the **external** file. This is because in a **real time** environment, e.g. using **global sections** on VMS or **modules** with OS9, the data base on the external medium can be changed by concurrent processes. However if the HBOOK data structure, associated with the histogram or Ntuple in memory is **altered** (e.g. by a `MAX, IDOPT, FIT` command), then it becomes the **default** for subsequent operations. If one wants the **original copy** one first must delete the copy from memory or **explicitly** use the pathname for the external file.

```
PAW > Histo/file 1 his.dat   | The file contains ID=10
PAW > Histo/Plot 10          | ID=10 read from file and plotted
PAW > H/plot 10              | ID=10 read again from file and plotted
PAW > H/fit 10 ! G           | Read from file, make a Gaussian fit on //PAWC/10
PAW > H/plot 10              | ID=10 read from memory since it changed
PAW > H/del 10               | Delete histogram 10 from memory
PAW > H/plot 10              | ID=10 read again from file and plotted
```

6.3 HBOOK batch as the first step of the analysis

Although it is possible to define histograms interactively in a PAW session, and then read the (many thousands of) events, in general for large data samples the relevant variables are extracted from the **Data Summary Files** or **DST**s and stored in **histograms** or an **Ntuple**. The histogram needs already that a certain choice has to be made as to the range of values for the plotted parameter, because the **binning**, or the coarseness, of the distribution has to be specified when the histogram is defined (**booked**). Also only one- and two-dimensional histograms are possible, hence the correlations between various parameters can be difficult to study. Hence it seems in many cases more appropriate to store the value of the important parameters for each event in an **Ntuple**. This approach preserves the correlation between the parameters and allows selection criteria to be applied on the (reduced) data sample at a later stage.

In general, the time consuming job of analysing all events available on tape is run on a mainframe or CPU server, and the important event parameters are stored in a Ntuple to allow further detailed study. For convenience the Ntuple can be output to disk for each run, and then at a later stage the Ntuples can be **merged** in order to allow a global interactive analysis of the complete data sample.

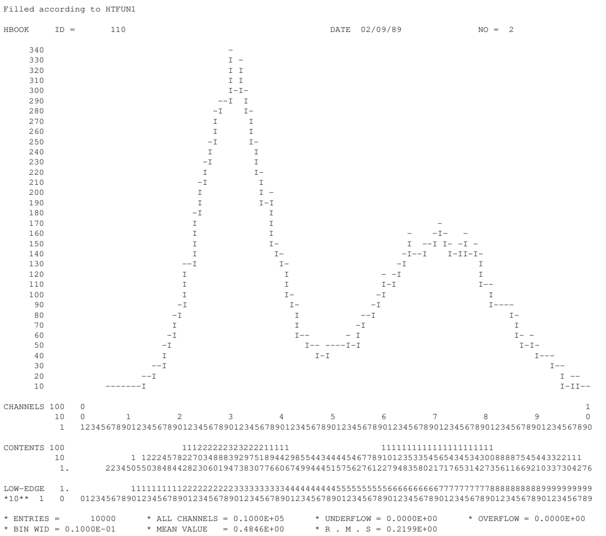
A typical batch job in which data are analysed offline and some characteristics are stored in HBOOK is like shown below.

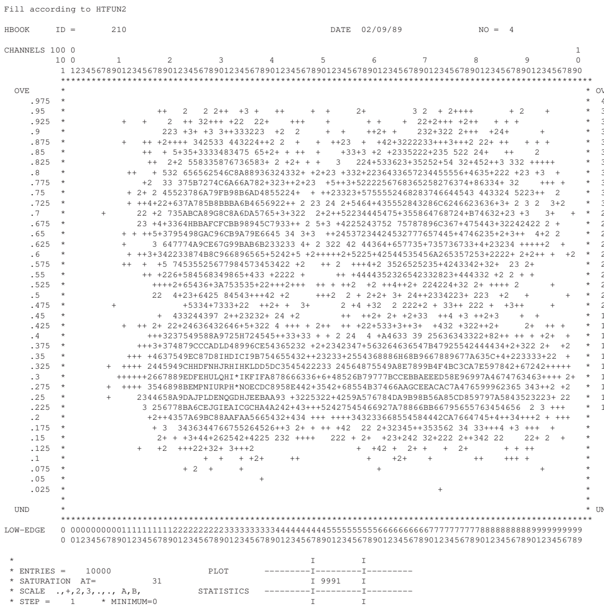
```
      PROGRAM HTEST
      PARAMETER (NWPAWC=20000)
      COMMON/PAWC/H(NWPAWC)
      EXTERNAL HTFUN1,HTFUN2
*.------------------------------------------------------------
      CALL HLIMIT(NWPAWC)
*             Book histograms and declare functions
      CALL HBFUN1(100,'Test of HRNDM1',100,0.,1.,HTFUN1)
      CALL HBOOK1(110,'Filled according to HTFUN1',100,0.,1.,1000.)
      CALL HBFUN2(200,'Test of HRNDM2',100,0.,1.,40,0.,1.,HTFUN2)
      CALL HSCALE(200,0.)
      CALL HBOOK2(210,'Fill according to HTFUN2',100,0.,1.,40,0.,1.,30.)
*             Fill histograms
      DO 10 I=1,10000
         X=HRNDM1(100)
         CALL HFILL(110,X,0.,1.)
         CALL HRNDM2(200,X,Y)
         CALL HFILL(210,X,Y,1.)
   10 CONTINUE
*             Save all histograms on file HTEST.HBOOK
      CALL HRPUT(0,'HTEST.HBOOK','N')
      CALL HDELETE(100)
      CALL HDELETE(200)
      CALL HPRINT(0)
      END
```

```
      FUNCTION HTFUN2(X,Y)
*             Two-dimensional gaussian
      HTFUN2=HTFUN1(X)*HTFUN1(Y)
      END

      FUNCTION HTFUN1(X)
*             Constants for gaussians
      DATA C1,C2/1.,0.5/
      DATA XM1,XM2/0.3,0.7/
      DATA XS1,XS2/0.07,0.12/
*             Calculate the gaussians
      A1=-0.5*((X-XM1)/XS1)**2
      A2=-0.5*((X-XM2)/XS2)**2
      X1=C1
      X2=C2
      IF(ABS(A1).GT.0.0001)X1=C1*EXP(A1)
      IF(ABS(A2).GT.0.0001)X2=C2*EXP(A2)
*             Return function value
      HTFUN1=X1+X2
      END
```

After opening the RZ HBOOK file, HBOOK is initialised by a call to `HLIMIT`, which declares a length of 20000 words for the length of the `/PAWC/` dynamic store. Then the one- and two- dimensional histograms 110 and 210 are filled respectively according to the functions `HTFUN1` and `HTFUN2`. The output generated by the program is shown below





6.3.1 Adding some data to the RZ file

The second run using program `HTEST1` shows how to add some data to the HBOOK RZ file created in the job `HTEST`. After opening the file in question in update mode (`'U'` option) with the name `EXAM2`, a new directory `NTUPLE` is created, known as `//EXAM2/NTUPLE` as seen in the output of `HLDIR` command at the end of the output. A one- and a two-dimensional histogram and a Ntuple with identifiers of respectively 10, 20 and 30 are booked. Each Ntuple element or "event" is characterised by three **variables** (labelled `'X'`, `'Y'` and `'Z'`). The Ntuple data, when the initial size of `1000` words is exhausted, will be written to the directory specified in the call to `HBOOKN`, i.e. `//EXAM2/NTUPLE`, and the data in memory are replaced with those newly read. A one- and a two-dimensional projection of `X` and `X Y` are then made onto histograms 10 and 20 respectively, before they are printed and written on the HBOOK RZ file. At the end the **current** and **parent** directories are listed. The contents of the latter shows that the data written in the first job (`HTEST`) are indeed still present in the file under the top directory `//EXAM2`. The call to `RZSTAT` shows usage statistics about the RZ file.

Example of adding data to a HBOOK RZ file

```
      PROGRAM HTEST1
      PARAMETER (NWPAWC=20000)
      COMMON/PAWC/H(NWPAWC)
      DIMENSION X(3)
      CHARACTER*8 CHTAGS(3)
      DATA CHTAGS/'   X   ','   Y   ','   Z   '/
*.------------------------------------------------------
      CALL HLIMIT(NWPAWC)
*             Reopen data base
      CALL HROPEN(1,'EXAM2','HTEST.HBOOK',0,'U')
      CALL HMDIR('NTUPLE','S')
      CALL HBOOK1(10,'TEST1',100,-3.,3.,0.)
      CALL HBOOK2(20,'TEST2',30,-3.,3.,30,-3.,3.,250.)
      CALL HBOOKN(30,'N-TUPLE',3,'//EXAM2/NTUPLE',1000,CHTAGS)
*
      DO 10 I=1,10000
         CALL RANNOR(A,B)
         X(1)=A
         X(2)=B
         X(3)=A*A+B*B
         CALL HFN(30,X)
  10  CONTINUE
*
      CALL HPROJ1(10,30,0,0,1,999999,1)
      CALL HPROJ2(20,30,0,0,1,999999,1,2)
      CALL HPRINT(0)
      CALL HROUT(0,ICYCLE,' ')
      CALL HLDIR(' ',' ')
      CALL HCDIR('\',' ')
      CALL HLDIR(' ',' ')
      CALL RZSTAT(' ',999,' ')
      CALL HREND('EXAM2')
      END
```

6.4 Using PAW to analyse data

After transferring the HBOOK RZ file, which was created in the batch job as explained in the previous section, we start a PAW session to analyse the data which were generated. The PAW session below shows that the file `HTEST.HBOOK` is first opened via a call to `HISTO/FILE`. The data on the file are now accessible as the top directory `//LUN1`. When listing with the `LDIR` command the contents of the top directory `//LUN1` and its `NTUPLE` subdirectory, the same information (histograms and Ntuples) is found as in the batch job (figure 6.3)

6.4.1 Plot histogram data

The analysis of the data can now start and we begin by looking at the histograms in the top directory. Figure 6.4 shows the commands entered and the corresponding output plot. They should be compared with the lineprinter output in Section 6.3.

6.5 Ntuples: A closer look

We now turn our attention to the `NTUPLE` directory to show the functionality and use of Ntuples. After making `NTUPLE` the **current** directory the available HBOOK objects are listed. The structure of the Ntuple with identifier `30` is `PRINTed`. The contents of the various Ntuple elements ("events") can be viewed by the `NTUPLE/SCAN` command. As with most Ntuple commands a **selection criterion** can be given to treat only given "selected" subsamples of the Ntuple (two examples are seen with the further `NTUPLE/SCAN` commands (see figure 6.5).

6.5.1 Ntuple plotting, variables and selection mechanisms

The general format of the command `NTUPLE/PLOT` to project and plot a Ntuple as a (1-Dim or 2-Dim) histogram with automatic binning, possibly using a selection algorithm is:

NTUPLE/PLOT `idn [ uwfunc nevent ifirst nupd chopt idh]`

`IDN` Ntuple Identifier and variable(s) (see table 6.1)

`UWFUNC` Selection function (see table 6.2) - Default no function

```
TEST1

HBOOK     ID =       10                                         DATE  02/09/89                               NO =  1

     280
     270                                                      - -
     260                                                      I I  -
     250                                                   -  I I  I
     240                                                 - I  I-I- I -
     230                                                 I-I--I  I I-I-
     220                                                -I       I I  I-
     210                                                I        I I   I-
     200                                                I        I-I    I-
     190                                          - - --I                I --
     180                                          I-I-I                  I-II--
     170                                          I                           I
     160                                          I                           I--
     150                                       - -I                             I --
     140                                      -I-I                              I II
     130                                     -I                                 I-II-
     120                                    -I                                      I-
     110                                  --I                                        I--
     100                                --I                                            I
      90                                I                                              I
      80                                I                                              I----
      70                              --I                                                  I-
      60                             -I                                                     I--
      50                          ---I                                                        I--
      40                     -----I                                                             I--
      30                     I                                                                    I-----
      20               - ----I                                                                         I---
      10       --------I-I                                                                                I--------

CHANNELS 100   0                                                                                                  1
          10   0         1         2         3         4         5         6         7         8         9         0
           1   1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890

CONTENTS 100                             11111111111111112222222222212222221111111111111
          10           1 111133333444666900012343487888813252263749623310978877552442100777765544332222211 1
           1.  1266487877127932587516069303434644322909949809367004036056844525243975324963516782565365312194856211

LOW-EDGE       --------------------------------------------------
           1.  3222222222222222211111111111111111                                 11111111111111111222222222222222
           0   0988776554432211099887665543322100998776654433211000112334456677899001223345566788990112234455677889
           0   0482604826048260482604826048260482604826048260482606284062840628406284062840628406284062840628406284

* ENTRIES =      10000      * ALL CHANNELS = 0.9969E+04      * UNDERFLOW = 0.1200E+02      * OVERFLOW = 0.1900E+02
* BIN WID = 0.6000E-01      * MEAN VALUE   =-0.3907E-02      * R . M . S = 0.9857E+00
```

```
TEST2

HBOOK     ID = 20        DATE  02/09/89         NO =  2

CHANNELS   10 U 0        1         2         3 O
            1 N 123456789012345678901234567890 V
           **************************************
  OVE      *        + ++ +232++2+ +++           * OVE
    2.8    *      ++ 2    +2 + 2  +             *  30
    2.6    *           2 2+  +34+++ ++   +      *  29
    2.4    *          2+ 3322343+ 3++ +         *  28
    2.2    *    + 2    247236663524+23++   +    *  27
    2      *    +    2+23769597A75 6+2+ 2       *  26
    1.8    *       + 5598576EBCDAA53357  2+ +   *  25
    1.6    *      ++3278CC9JFO8F98C86643+2+     *  24
    1.4    *      344686AAGJJMEMIDFG964232+   + *  23
    1.2    *    ++++44BBJGMQOPWNICCGI97322++  + *  22
    1      *     2+545BGOMTSX*VYTJMCFA755++2    *  21
     .8    *    2+4799DHSRUX****VXRQJC57635+    *  20
     .6    *   + +25CBEKLZ********MXGGCI4322  3 *  19
     .4    * 2   4+779BN*U*********YOIFB862     *  18
     .2    * 2 ++266CCLR************OIHA464+2 4 *  17
           * +  3238ECX*T***********YKPC772   + *  16
-    .2    * + +423D6LDS**X********ZUMGC543+  2 *  15
-    .4    * +  2347CAHSSX*********UMK75D2 3  + *  14
-    .6    *   2334AAKML*V**********IIH9773++ + *  13
-    .8    *   +22565CLJL*X******Z*TL9H948+ +   *  12
-   1      * 2 2 32666EMLN****Q*ULLQMABB342+  2 *  11
-   1.2    *   + 22377BDIUS*P***TTUNBDA545+2    *  10
-   1.4    * + + 2 +689E7KKNWUNRIHJCEA472+++  + *   9
-   1.6    *     2+3+74BCMJIGOIKEIAAD6643++   2 *   8
-   1.8    * + + +2222856AA8HGJACB6786+2+2++    *   7
-   2      *   +   2 +273598EDC5977634++        *   6
-   2.2    * +   + ++2+274977548883+++2 +++     *   5
-   2.4    *         +  +3367558445+442+   +    *   4
-   2.6    *       +2 +  2224+6++7234 +    +    *   3
-   2.8    *          +  33+3+322++ +           *   2
-   3      *       ++ ++ 22 2 +4+2 2            *   1
  UND      *          + +  23 +2+++      +      * UND
           **************************************
LOW-EDGE       ---------------
           1.  32222211111         1111122222
           0   086420864208642024680246802468

*                                                   I    19  I
* ENTRIES =    10000            PLOT         -------I--------I-------
* SATURATION  AT=          255                 12   I  9936  I    19
* SCALE  .,+,2,3,.,., A,B,      STATISTICS   -------I--------I-------
* STEP =    1     * MINIMUM=0                       I    14  I
```

```
********************************************************
* NTUPLE ID=   30  ENTRIES=  10000   N-TUPLE           *
********************************************************
*  Var numb  *   Name    *    Lower     *    Upper     *
********************************************************
*      1     *    X      * -.422027E+01 * 0.386411E+01 *
*      2     *    Y      * -.411076E+01 * 0.378366E+01 *
*      3     *    Z      * 0.485187E-04 * 0.179518E+02 *
********************************************************


===> Directory : //EXAM2/NTUPLE
        30 (N)   N-TUPLE
        10 (1)   TEST1
        20 (2)   TEST2

===> Directory : //EXAM2
       100 (1)   Test of HRNDM1
       110 (1)   Filled according to HTFUN1
       200 (2)   Test of HRNDM2
       210 (2)   Fill according to HTFUN2


     NREC    NWORDS    QUOTA(%)  FILE(%)    DIR. NAME
      34      34064      0.85      0.85   //EXAM2/NTUPLE
      41      40431      1.02      1.02   //EXAM2
```

```
PAW > histo/file 1 htest.hbook    | open the HBOOK RZ file
PAW > ldir                        | list current directory

************** Directory ===> //LUN1 <===

                 Created 890902/1955  Modified 890902/1958

===> List of subdirectories
NTUPLE           Created 890902/1958 at record     9

===> List of objects
    HBOOK-ID  CYCLE   DATE/TIME   NDATA   OFFSET    REC1    REC2
       100       1   890902/1955    153       1       3
       110       1   890902/1955     88     154       3
       200       1   890902/1955   4335     242       3       4 ==>     7
       210       1   890902/1955    767     481       7       8

 NUMBER OF RECORDS =    7  NUMBER OF MEGAWORDS =  0 +  6367 WORDS
 PER CENT OF DIRECTORY QUOTA USED =    0.175
 PER CENT OF FILE USED            =    0.175
 BLOCKING FACTOR                  =   74.540
PAW > ldir ntuple               | list directory in NTUPLE

************** Directory ===> //LUN1/NTUPLE <===

                 Created 890902/1958  Modified 890902/1958

===> List of objects
    HBOOK-ID  CYCLE   DATE/TIME   NDATA   OFFSET    REC1    REC2
        30       2   890902/1958   1082     215      41      42
                 1   890902/1958   1082     725      39      40
        10       1   890902/1958    151     783      40
        20       1   890902/1958    305     934      40      41

 NUMBER OF RECORDS =   34  NUMBER OF MEGAWORDS =  0 + 34064 WORDS
 PER CENT OF DIRECTORY QUOTA USED =    0.851
 PER CENT OF FILE USED            =    0.850
 BLOCKING FACTOR                  =   94.899
```

Figure 6.3: Adding and reading data on a HBOOK RZ direct access file

Plotting histogram data

```
PAW > zon 1 2          | Divide picture into 2 vertically
PAW > set htyp -3      | Set hatch style for histogram
PAW > hi/pl 110        | Plot 1-dimensional histogram 110
PAW > hi/pl 210        | Plot 2-dimensional histogram 210
```

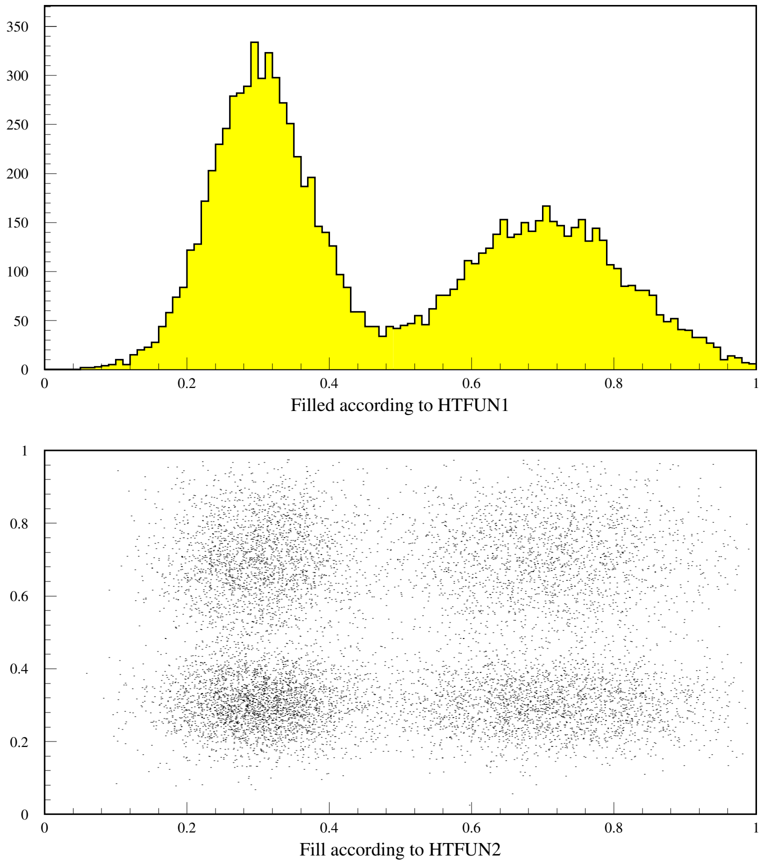


Figure 6.4: Plot of one- and two-dimensional histograms

`NEVENT` Number of events to be processed (default is `999999`)

`IFIRST` First event to be processed (default is `1`)

`NUPD` Frequency with which to update histogram (default is `1000000`)

`OPTION` Options

`IDH` Identifier of histogram to fill

With most Ntuple operations a "selection function" `UWFUNC` of a form described in table 6.2 can be used, i.e. it can take the form of a simple or composed **expression** or an **external FORTRAN function**, executed by COMIS [1], a **cut** or a **mask**. The selection function also acts as a **weighting** factor.

6.5.2 Masks

The mask facility allows the user to specify up to **32** selection criteria associated with a Ntuple. These criteria are defined like cuts, but their value for each event are written to an external direct access file, from which the information can be readily retrieved at a later stage, without recalculating the condition value in question. In the

```
PAW > cd ntuple                                  | move to NTUPLE directory
PAW > hi/li                                      | list HBOOK objects

===> Directory : //LUN1/NTUPLE
        30 (N)   N-TUPLE
        10 (1)   TEST1
        20 (2)   TEST2

PAW > nt/print 30                                | print summary for Ntuple 30

********************************************************
* NTUPLE ID=   30  ENTRIES=  10000   N-TUPLE           *
********************************************************
*  Var numb  *   Name    *    Lower     *    Upper     *
********************************************************
*      1     *    X      * -.422027E+01 * 0.386411E+01 *
*      2     *    Y      * -.411077E+01 * 0.378365E+01 *
*      3     *    Z      * 0.485188E-04 * 0.179518E+02 *
********************************************************

PAW > nt/scan 30                                 | scan the first elements
+-------+--------------+--------------+--------------+
| Event |   X          |   Y          |   Z          |
+-------+--------------+--------------+--------------+
|     1 | -1.06459     | -1.82194     |  4.45282     |
|     2 | -1.15619     |  0.106067    |  1.34802     |
|     3 |  0.923492    |  0.943671    |  1.74335     |
|     4 | -0.145332    | -0.57672     |  0.353727    |
|     5 | -1.18289     |  1.50525     |  3.66501     |
|     6 | -0.658942    |  1.17934     |  1.82504     |
|     7 | -0.071134    |  0.216755    |  0.0520428   |
|     8 | -1.45944     |  0.869828    |  2.88655     |
|     9 |  2.2881      | -0.103207    |  5.24604     |
|    10 | -0.70103     | -0.238115    |  0.548141    |
|    11 |  1.27792     | -0.633723    |  2.03468     |
|    12 |  0.046591    |  0.45629     |  0.210371    |
|    13 | -0.966939    |  0.441924    |  1.13027     |
|    14 |  0.299147    |  1.72798     |  3.07542     |
|    15 |  1.35417     |  0.425711    |  2.015       |
|    16 |  2.51372     | -1.17377     |  7.69653     |
|    17 |  0.974036    | -0.677181    |  1.40732     |
|    18 |  0.299531    | -1.10509     |  1.31094     |
|    19 |  0.407014    |  0.236156    |  0.22143     |
+-------+--------------+--------------+--------------+
More...? ( <CR>/N/G ) n

PAW > nt/sc 30 z>16                              | example of a condition on the Z variable
+-------+--------------+--------------+--------------+
| Event |   X          |   Y          |   Z          |
+-------+--------------+--------------+--------------+
|  1945 | -0.08474     |  4.00098     |  16.015      |
|  7664 |  0.81875     |  3.9523      |  16.291      |
+-------+--------------+--------------+--------------+
==> 2 events satisfied the imposed cuts

PAW > nt/sc 30 abs(x)>4.or.abs(y)>4              | example of a more complex selection criterion
+-------+--------------+--------------+--------------+
| Event |   X          |   Y          |   Z          |
+-------+--------------+--------------+--------------+
|  1945 | -0.08474     |  4.00098     |  16.015      |
+-------+--------------+--------------+--------------+
==> 1 event satisfied the imposed cuts
```

Figure 6.5: Print and scan Ntuple elements

example session below first a **new** mask file MNAME.MASK is defined. Next we define event selection criteria and store their result at various bit positions in the mask vector MNAME.

Defining cuts and masks

```
PAW > NT/CUT $4 Z>X**2                  | Define cut 4
PAW > MASK/FILE MNAME N
PAW > NT/PLOT 30.X X**2+Y**2>2>>MNAME(1)
PAW > NT/PLOT 30.X $4.AND.Y>1>>MNAME(2)
PAW > NT/PLOT 30.Y SIN(Z).GT.SIN(Y)>>MNAME(3)
```

Format	Explanation	Example
IDN.CHNAME	The variable **named** "CHNAME"	30.x variable x
IDN.expression	Expression is any numerical expression of Ntuple variables. It may include a call to a COMIS function.	30.X**2+Y**2 30.X*COMIS.F
IDN.B%A	Scatter-plot of variable B versus A for each event.	30.Y%X Y versus X
IDN.expr1%expr2	expr1 and expr2 can be any numerical expression of the Ntuple variables. They can be COMIS functions.	30.SQRT(X**2+Y**2)%SIN(Z) 30.COMIS1.F%COS(Z)
	Any combination of the above	30.3%COMIS2.F*SIN(X)

Table 6.1: Syntax for specifying Ntuple variables

Format	Explanation	Example
0 or missing	No selection is applied (weight is 1).	NT/PLOT 30.X
Combination of cuts	A CUT or combination of CUTs, each created by the command NTUPLE/CUTS	NT/PLOT 30.X $1 (use cut $1) NT/PLOT 30.X $1.AND.$2 NT/PLOT 30.X .NOT.($1.AND.$3).OR.$2
Combination of masks	A MASK or combination of MASKs, each created by the command NTUPLE/MASK/FILE	Assuming there is a mask vector MSK: NT/PLOT 30.X MSK(4) (bit 4) NT/PLOT 30.X MSK(1).OR.MSK(6)
Logical expression	Any **logical** combination of conditions between Ntuple variables, cuts and masks.	NT/PLOT 30.X X>3.14.AND.(Y<Z+5.) NT/PLOT 30.X $1.AND.MASK(3).OR.Z<10
Numerical expression	Any **numerical** combination of constants and Ntuple variables. In this case the value of the expression will be applied as a **weight** to the element being plotted.	NT/PLOT 30.X Y weight X by Y NT/PLOT 30.X X**2+Y**2 weight X by X^2+Y^2
Selection function	Name of a selection function in a text file of the form fun.f (Unix), FUN.FOR (VAX). The function value is applied as a **weight**	NTUPLE/PLOT 30.X SELECT.F For each event the plotted value of X will be multiplied by the value of the selection function SELECT calculated for that event.
	Any combination of the above	NT/PL 30.Y%F1.F*SIN(X) $1.OR.F2.F

Table 6.2: Syntax of a selection function used with a Ntuple

```
 PAW > MASK/LIST MNAME                    | Print mask definitions

MNAME      Events:  10000  (file MNAME.mask, read/write)
            # select  Description
    bit  1:     3577  X**2+Y**2>2
    bit  2:     1567  $4.AND.Y>1
    bit  3:     7050  SIN(Z).GT.SIN(Y)

 PAW > MASK/CLOSE MNAME                   | close MNAME.MASK file
```

Of course doing this kind of gymnastics makes sense only if a **time consuming** selection mechanism is used and only a few events are selected. In a subsequent run the mask file can then be read to display the information much more quickly.

Using a mask file of a previous run

```
PAW > MASK/FILE MNAME                    | open the mask file for read
PAW > NT/PLOT 30.X MNAME(1)              | plot using bit 1
PAW > NT/PLOT 30.X MNAME(2)              | plot using bit 2
PAW > NT/PLOT 30.Y MNAME(3)              | plot using bit 3
PAW > MASK/CLOSE MNAME                   | close MNAME.MASK file
```

Cuts

A **cut** is identified by an integer (between 0 and 100) preceded by a $ sign and is a **logical** expression of Ntuple elements, other cuts, masks or functions.

Example of cuts

```
PAW > NT/CUT $1 4<X                       | variable
PAW > NT/CUT $2 0.4<X<0.8.AND.Y<SQRT(Z)   | ditto
PAW > NT/CUT $3 FUN.F                     | external function
PAW > NT/CUT $4 FUN.F.AND.Z>X**2          | ditto plus variable
PAW > NT/CUT $5 ($1.AND.$2).OR.$4       | combination of cuts
PAW > NT/CUT $6 $1.AND.Z<0               | cut and variable
PAW > NT/CUT $7 X                         | event weight
PAW > NT/CUT $8 SQRT(Y)                   | ditto
PAW > NT/CUT $9 MASK(23).AND.$8          | mask and cut
```

Cut definitions can be written to a file and later re-read.

```
PAW > NT/CUT $0 W cuts.dat                | write all cuts to file
PAW > NT/CUT $4 R cuts.dat                | read cut 4 from file
PAW > NT/CUT $4 P                         | print cut 4
$4  =  FUN.F.AND.Z>X**2
```

Graphical cut

One can also define a cut on the screen in a **graphical** way, by pointing out the upper and lower limits (1-dimensional case) or an area (2-dimensional case) by using the mouse or arrow keys (see figure 6.6).

Using graphical cuts

```
  PAW > gcut 1 30.x%y                                  | graphical cut 1
  PAW > zon 1 2                                        | define picture layout
  PAW > title 'Graphical cuts'                         | title for picture
  PAW > 2d 211 'X versus Y' 50 -2.5 2.5 50 -2.5 2.5 0.  | user binning
  PAW > 1d 212 'X - Before and after cut' 60 -3. 3. 0.  |    ditto
  PAW > 1d 213 'Y - Before and after cut' 60 -3. 3. 0.  |    ditto
  PAW > nt/pl 30.x%y idh=211                           | plot y versus x in histogram 211
  PAW > cut $1 d                                       | draw graphical cut 1
  PAW > zon 2 2 3 s                                    | redefine the picture layout
  PAW > nt/pl 30.x idh=212                             | plot x BEFORE cut in histogram 212
  PAW > set htyp -3                                    | use hatch for plot after cut
  PAW > nt/pl 30.x $1 option=s idh=212                 | plot x AFTER cut on same plot
  PAW > set htyp 0                                     | no hatch for plot without cut
  PAW > nt/pl 30.y idh=213                             | plot y BEFORE cut in histogram 213
  PAW > set htyp -3                                    | use hatch for plot after cut
  PAW > nt/pl 30.y $1 option=s idh=213                 | plot y AFTER cut on same plot
```

COMIS selection function

In the definition of a selection criterion an external function (in the sense that it has not been compiled and linked together with PAW) can be used. This function is interpreted by the COMIS [1] package. The CERNLIB functions which are callable from within such a function are given in the online help of the command CALL.

The command NTUPLE/UWFUNC allows a selection function for a Ntuple to be prepared more easily. It generates a function with a name specified by the user and with code making available the variables corresponding to the given Ntuple identifier via a COMMON block. As an example consider the Ntuple number 30 used previously.

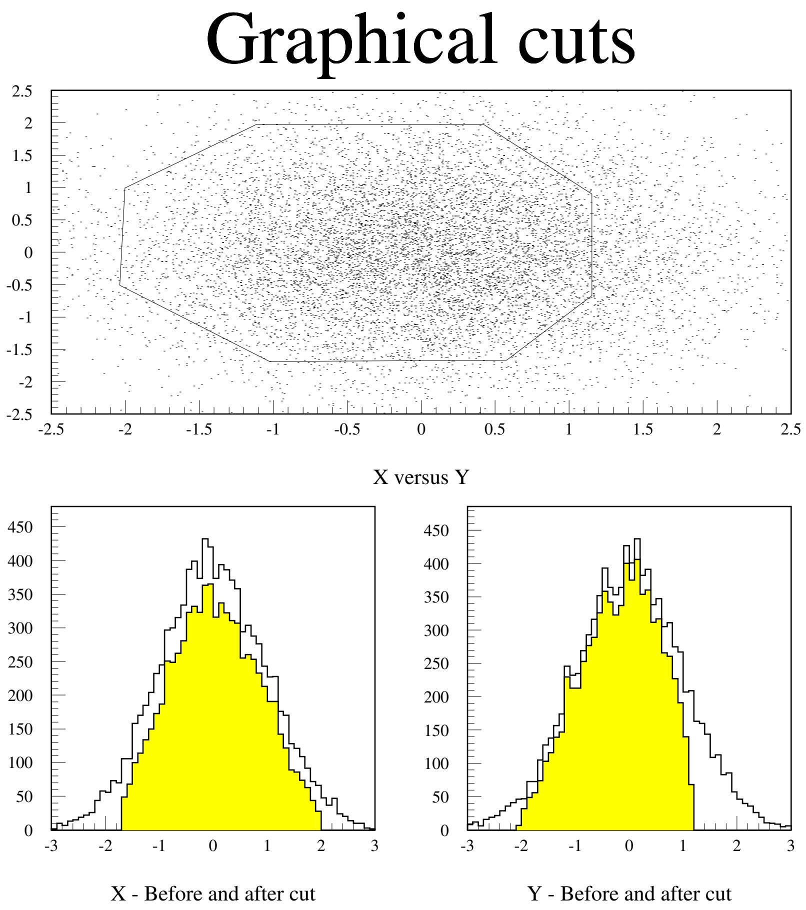


Figure 6.6: Graphical definition of cuts

Specifying a user selection function

```
 PAW > NTUPLE/UWFUNC 30 SELECT.F EPT    | Generate and edit SELECT.F
      REAL FUNCTION SELECT(XDUMMY)
      REAL X     ,    Y     ,    Z
      COMMON/PAWIDN/IDNEVT,OBS(13),
     +    X     ,    Y     ,    Z
      DIMENSION XDUMMY(  3)
      CHARACTER*8 CHTAGS(  3)
      DATA CHTAGS/'    X     ','    Y     ','    Z     '/
*
      SELECT=1.
      PRINT 1000,IDNEVT
      DO 10 I=1,  3
         PRINT 2000,I,CHTAGS(I),XDUMMY(I)
  10  CONTINUE
*
 1000 FORMAT(8H IDNEVT=,I5)
 2000 FORMAT(5X,I3,5X,A,1H=,G14.7)
      END
```

The user can add further FORTRAN code with the command `EDIT`. Remember that the value of the function can be used for weighting each event.

Plotting Ntuples

```
PAW > ZONE 1 2                       | 2 histograms one above the other
PAW > OPTION STAT                    | Write statistics on plot
PAW > NT/PLOT 30.Z                   | plot variable Z of Ntuple 30
PAW > 1d 300 'Z recalculated and user binning' 100 0. 10.
PAW > NT/PLOT 30.X**2+Y**2 IDH=300 | Recalculate variable Z + plot with user binning
```

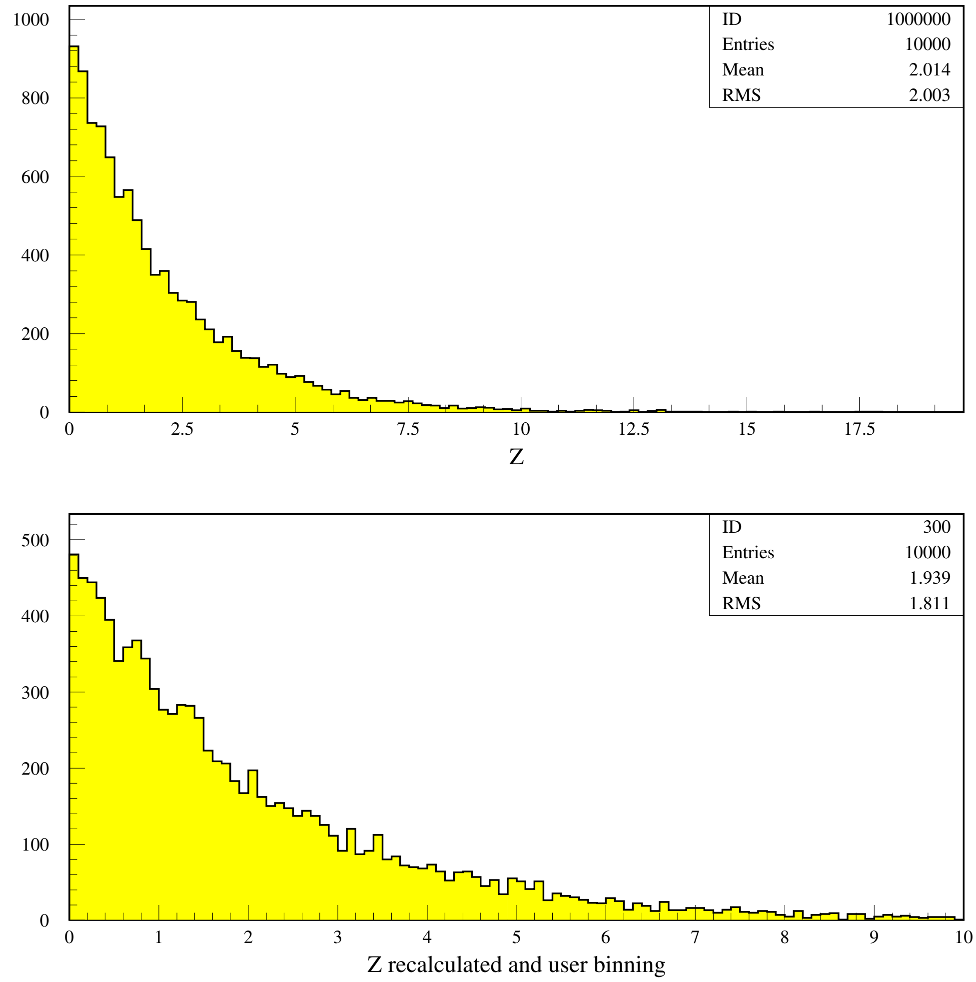


Figure 6.7: Read and plot Ntuple elements

6.5.3 Examples

To put into practice the syntax explained above let us consider figure 6.7. We first plot variable Z with the binning automatically calculated by HBOOK. Then we define a histogram with identifier 300 into which we want HBOOK to plot the squared sums of the elements X and Y. This corresponds to the definition of the Z variable as can be seen in the FORTRAN listing in figure 6.3. As the MEAN and RMS are only calculated on the events within the histogram boundaries, they differ slightly between the top and bottom plot in figure 6.7.

6.6 Fitting with PAW/HBOOK/MINUIT

Minuit[5][2] is conceived as a tool to find the minimum value of a multi-parameter function and analyze the shape of the function around the minimum. The principal application is foreseen for statistical analysis, working on

[2]The following information about Minuit has been extracted from the Minuit documentation.

More complex Ntuple presentations

```
PAW > zone 2 2                                     | Divide plot in 4 zones
PAW > option STAT                                  | Select option to write statistics on plot
PAW > set HTYP -3                                  | Define histogram hatch type
PAW > 1d 401 'NT/PL - X' 100. -2.5 2.5             | Book 1 dim histogram
PAW > nt/pl 30.1 idh=401                           | Plot variable 1 (x) using histogram 401
PAW > 1d 402 'NT/PL E option - Y' 100. -2.5 2.5    | 1 dim histogram (different title)
PAW > set MTYP 21                                  | Select market type for points on plot
PAW > nt/pl 30.y option=E idh=402                  | Plot y variable with Error bar option
PAW > 1d 403 'NT/PL B option - X' 40. -2.5 2.5     | 1 dim histogram (different title + binning)
PAW > set BARW 0.4                                 | Define bar width for bar chart
PAW > set BARO 0.3                                 | Define bar origin for bar chart
PAW > csel NB 0.33                                 | Print selection criterion on plot
PAW > set HCOL 1001                                | Histogram colour black
PAW > nt/pl 30.x y>0 option=B idh=403              | Plot x variable as bar chart
PAW > 1d 404 'NT/PL PL option - Y' 100. -2.5 2.5   | 1 dim histogram (different title)
PAW > max 404 160                                  | Fix maximum for plotting hist 404
PAW > nt/pl 30.y sqrt(z)>1 -404 option=pl          | Plot y variable with PL option
```

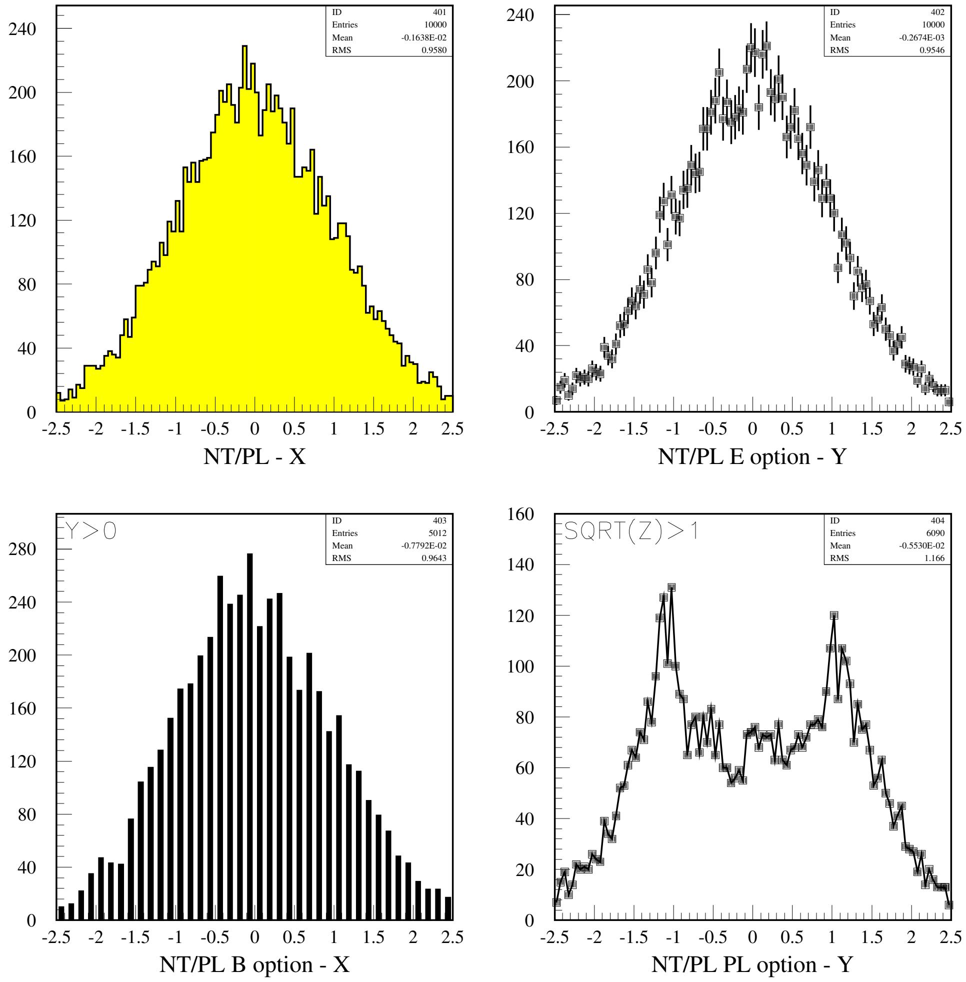


Figure 6.8: Selection functions and different data presentations

chisquare or log-likelihood functions, to compute the best-fit parameter values and uncertainties, including correlations between the parameters. It is especially suited to handle difficult problems, including those which may require guidance in order to find the correct solution.

6.6.1 Basic concepts of MINUIT.

The MINUIT package acts on a multiparameter FORTRAN function to which one must give the generic name `FCN`. In the PAW/HBOOK implementation, the function `FCN` is called `HFCNH` when the command `Histo/Fit` (PAW) or the routine `HFITH` are invoked. It is called `HFCNV` when the command `Vector/Fit` or the routine `HFITV` are invoked. The value of `FCN` will in general depend on one or more variable parameters.

To take a simple example, suppose the problem is to fit a polynomial through a set of data points with the command Vector/Fit. Routine `HFCNV` called by `HFITV` calculates the chisquare between a polynomial and the data; the variable parameters of `HFCNV` would be the coefficients of the polynomials. Routine `HFITV` will request MINUIT to minimize `HFCNV` with respect to the parameters, that is, find those values of the coefficients which give the lowest value of chisquare.

6.6.2 Basic concepts - The transformation for parameters with limits.

For variable parameters with limits, MINUIT uses the following transformation:

$$P_{\mathrm{int}} = \arcsin\left(2\frac{P_{\mathrm{ext}} - a}{b - a} - 1\right) \qquad\qquad P_{\mathrm{ext}} = a + \frac{b - a}{2}\left(\sin P_{\mathrm{int}} + 1\right)$$

so that the internal value P_{int} can take on any value, while the external value P_{ext} can take on values only between the lower limit a and the upper limit b. Since the transformation is necessarily non-linear, it would transform a nice linear problem into a nasty non-linear one, which is the reason why limits should be avoided if not necessary. In addition, the transformation does require some computer time, so it slows down the computation a little bit, and more importantly, it introduces additional numerical inaccuracy into the problem in addition to what is introduced in the numerical calculation of the `FCN` value. The effects of non-linearity and numerical roundoff both become more important as the external value gets closer to one of the limits (expressed as the distance to nearest limit divided by distance between limits). The user must therefore be aware of the fact that, for example, if he puts limits of $(0, 10^{10})$ on a parameter, then the values 0.0 and 1.0 will be indistinguishable to the accuracy of most machines.

The transformation also affects the parameter error matrix, of course, so MINUIT does a transformation of the error matrix (and the "parabolic" parameter errors) when there are parameter limits. Users should however realize that the transformation is only a linear approximation, and that it cannot give a meaningful result if one or more parameters is very close to a limit, where $\partial P_{\mathrm{ext}}/\partial P_{\mathrm{int}} \approx 0$. Therefore, it is recommended that:

- Limits on variable parameters should be used only when needed in order to prevent the parameter from taking on unphysical values.
- When a satisfactory minimum has been found using limits, the limits should then be removed if possible, in order to perform or re-perform the error analysis without limits.

6.6.3 How to get the right answer from MINUIT.

MINUIT offers the user a choice of several minimization algorithms. The `MIGRAD` (Other algorithms are available with Interactive MINUIT, as described on Page 96) algorithm is in general the best minimizer for nearly all functions. It is a variable-metric method with inexact line search, a stable metric updating scheme, and checks for positive-definiteness. Its main weakness is that it depends heavily on knowledge of the first derivatives, and fails miserably if they are very inaccurate. If first derivatives are a problem, they can be calculated analytically inside the user function and communicated to PAW via the routine `HDERIV`.

If parameter limits are needed, in spite of the side effects, then the user should be aware of the following techniques to alleviate problems caused by limits:

Getting the right minimum with limits.

If MIGRAD converges normally to a point where no parameter is near one of its limits, then the existence of limits has probably not prevented MINUIT from finding the right minimum. On the other hand, if one or more parameters is near its limit at the minimum, this may be because the true minimum is indeed at a limit, or it may be because the minimizer has become "blocked" at a limit. This may normally happen only if the parameter is so close to a limit (internal value at an odd multiple of $\pm\frac{\pi}{2}$ that MINUIT prints a warning to this effect when it prints the parameter values.

The minimizer can become blocked at a limit, because at a limit the derivative seen by the minimizer $\partial F/\partial P_{\mathrm{int}}$ is zero no matter what the real derivative $\partial F/\partial P_{\mathrm{ext}}$ is.

$$\frac{\partial F}{\partial P_{\mathrm{int}}} = \frac{\partial F}{\partial P_{\mathrm{ext}}}\frac{\partial P_{\mathrm{ext}}}{\partial P_{\mathrm{int}}} = \frac{\partial F}{\partial P_{\mathrm{ext}}} = 0$$

Getting the right parameter errors with limits.

In the best case, where the minimum is far from any limits, MINUIT will correctly transform the error matrix, and the parameter errors it reports should be accurate and very close to those you would have got without limits. In other cases (which should be more common, since otherwise you wouldn't need limits), the very meaning of parameter errors becomes problematic. Mathematically, since the limit is an absolute constraint on the parameter, a parameter at its limit has no error, at least in one direction. The error matrix, which can assign only symmetric errors, then becomes essentially meaningless.

6.6.4 Interpretation of Parameter Errors:

There are two kinds of problems that can arise: the **reliability** of MINUIT's error estimates, and their **statistical interpretation**, assuming they are accurate.

Statistical interpretation:

For discussion of basic concepts, such as the meaning of the elements of the error matrix, or setting of exact confidence levels, see [9, 10, 11].

Reliability of MINUIT error estimates.

MINUIT always carries around its own current estimates of the parameter errors, which it will print out on request, no matter how accurate they are at any given point in the execution. For example, at initialization, these estimates are just the starting step sizes as specified by the user. After a `MIGRAD` or `HESSE` step, the errors are usually quite accurate, unless there has been a problem. MINUIT, when it prints out error values, also gives some indication of how reliable it thinks they are. For example, those marked `CURRENT GUESS ERROR` are only working values not to be believed, and `APPROXIMATE ERROR` means that they have been calculated but there is reason to believe that they may not be accurate.

If no mitigating adjective is given, then at least MINUIT believes the errors are accurate, although there is always a small chance that MINUIT has been fooled. Some visible signs that MINUIT may have been fooled are:

- Warning messages produced during the minimization or error analysis.
- Failure to find new minimum.
- Value of `EDM` too big (estimated Distance to Minimum).
- Correlation coefficients exactly equal to zero, unless some parameters are known to be uncorrelated with the others.
- Correlation coefficients very close to one (greater than 0.99). This indicates both an exceptionally difficult problem, and one which has been badly parameterized so that individual errors are not very meaningful because they are so highly correlated.
- Parameter at limit. This condition, signaled by a MINUIT warning message, may make both the function minimum and parameter errors unreliable. See the discussion above "*Getting the right parameter errors with limits*".

The best way to be absolutely sure of the errors, is to use "independent" calculations and compare them, or compare the calculated errors with a picture of the function. Theoretically, the covariance matrix for a "physical" function must be positive-definite at the minimum, although it may not be so for all points far away from the minimum, even for a well-determined physical problem. Therefore, if MIGRAD reports that it has found a non-positive-definite covariance matrix, this may be a sign of one or more of the following:

A non-physical region: On its way to the minimum, MIGRAD may have traversed a region which has unphysical behavior, which is of course not a serious problem as long as it recovers and leaves such a region.

An underdetermined problem: If the matrix is not positive-definite even at the minimum, this may mean that the solution is not well-defined, for example that there are more unknowns than there are data points, or that the parameterization of the fit contains a linear dependence. If this is the case, then MINUIT (or any other program) cannot solve your problem uniquely, and the error matrix will necessarily be largely meaningless, so the user must remove the underdeterminedness by reformulating the parameterization. MINUIT cannot do this itself.

Numerical inaccuracies: It is possible that the apparent lack of positive-definiteness is in fact only due to excessive roundoff errors in numerical calculations in the user function or not enough precision. This is unlikely in general, but becomes more likely if the number of free parameters is very large, or if the parameters are badly scaled (not all of the same order of magnitude), and correlations are also large. In any case, whether the non-positive-definiteness is real or only numerical is largely irrelevant, since in both cases the error matrix will be unreliable and the minimum suspicious.

An ill-posed problem: For questions of parameter dependence, see the discussion above on positive-definiteness.

Possible other mathematical problems are the following:

Excessive numerical roundoff: Be especially careful of exponential and factorial functions which get big very quickly and lose accuracy.

Starting too far from the solution: The function may have unphysical local minima, especially at infinity in some variables.

6.6.5 Fitting histograms

The general syntax of the command to fit histograms is:

HISTOGRAM/FIT `id func [ chopt np par step pmin pmax errpar ]`

Only the parameters, which are of more general use, are described in detail. For an up to date description of this command have a look in the online help or in the reference manual.

`ID` A histogram identifier (1-dim or 2-dim)
A bin range may be specified, e.g. `Histo/Fit 10(25:56) ...`

`FUNC` Name of a function to be fitted to the histogram.
This function can be of various forms:

1 The name of a file which contains the user defined function to be minimized. Function name and file name must be the same. For example file `FUNC.FOR` is:

```
FUNCTION FUNC(X)   or FUNC(X,Y) for a 2-Dim histogram
COMMON/PAWPAR/PAR(2)
FUNC=PAR(1)*X +PAR(2)*EXP(-X)
END
```

2 One of the keywords below **(1-dim histograms only)**, which will use the parameterization described at the right for the fit.

```
G    Func=par(1)*exp(-0.5*((x-par(2))/par(3))**2)
E    Func=exp(par(1)+par(2)*x)
Pn   Func=par(1)+par(2)*x+par(3)*x**2...+par(n+1)*x**n, 0<n<20
```

3 A combination of the keywords above with the 2 operators `+` or `*`.

Note that in this case, the order of parameters in PAR must correspond to the order of the basic functions. Blanks are not allowed in the expression.

`CHOPT` All options of the `HISTO/PLOT` command plus the following additional ones:

- `0` Do not plot the result of the fit. By default the fitted function is drawn unless the option “N” below is specified.
- `B` Some or all parameters are bounded. In this case vectors `STEP,PMIN,PMAX` must be specified. Default is: All parameters vary freely.
- `D` The user is assumed to compute derivatives analytically using routine `HDERIV`. By default, derivatives are computed numerically.
- `L` Use Log Likelihood method. Default is χ^2 method.
- `M` Invokes interactive Minuit (See on Page 96)
- `N` Do not st ore the result of the fit bin by bin with the histogram. By default the function is calculated at the centre of each bin and the fit results stored with the histogram data structure.
- `Q` Quiet mode. No output printed about the fit.
- `V` Verbose mode. Results are printed after each iteration. By default only final results are printed.
- `W` Sets weights equal to 1.

`NP` Number of parameters in fit ($0 \leq$ `NP` ≤ 34)

`PAR` Vector containing the fit parameters.
Before the fit: Vector containing the initial values
After the fit: Vector containing the fitted values.

`STEP` Vector with step size for fit parameters

`PMIN` Vector with lower bounds for fit parameters

`PMAX` Vector with upper bounds for fit parameters

`ERRPAR` Vector with errors on the fitted parameters

When using predefined functions (case 2 for the `FUNC` parameter) initial values need not be specified when `NP=0`. In this case the parameter vector `PAR`, if specified, is only filled with the fitted parameters on **output**.

6.6.6 A simple fit with a gaussian

Example of simple fit with gaussian in PAW

```
PAW > opt stat    | Select option to show histogram statistics on plot
PAW > opt fit     | Select option to show fitted parameters on plot
PAW > hi/fit 10 G | Fit histogram 10 with a single gaussian
    *********************************************
    *                                           *
    * Function minimization by SUBROUTINE HFITGA *
    * Variable-metric method                    *
    * ID =         10  CHOPT = T                *
    *                                           *
    *********************************************
Convergence when estimated distance to minimum (EDM) .LT.  0.10E-03

FCN=   96.97320     FROM MIGRAD    STATUS=CONVERGED  CALLS=  549 EDM=  0.26E-03
                    STRATEGY= 1    ERROR DEF=    1.0000

INT EXT  PARAMETER                                   STEP         FIRST
NO. NO.    NAME         VALUE          ERROR         SIZE      DERIVATIVE
  1  1  Constant      239.83        2.8178       0.00000       0.57627E-02
  2  2  Mean        -0.53038E-02   0.77729E-04   0.00000        22.025
  3  3  Sigma        0.98766       0.70224E-02   0.00000      -0.88534

CHISQUARE = 0.1021E+01  NPFIT =   98
```

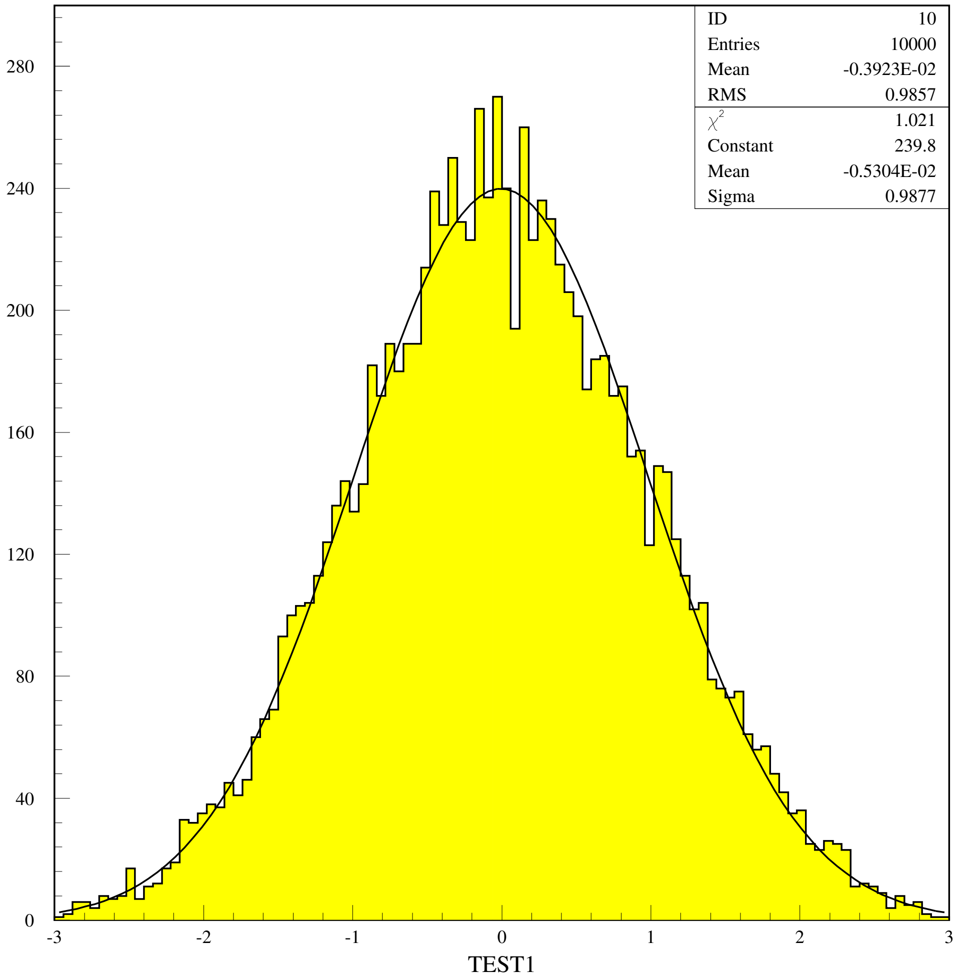


Figure 6.9: Example of a simple fit of a one-dimensional distribution

Fit parts of histogram separately

```
PAW > opt NSTA                          | Turn off option showing statistics on plot
PAW > ve/cr par(6)                      | Create a vector with 6 elements
PAW > set fit 111                       | Show fitted parameters + errors on plot
PAW > hi/fit 110(1:50) G ! 0 par        | Fit first half with a gaussian and plot

    **********************************************
    *                                            *
    * Function minimization by SUBROUTINE HFITGA *
    * Variable-metric method                     *
    * ID =        110  CHOPT = TR                *
    *                                            *
    **********************************************
Convergence when estimated distance to minimum (EDM) .LT.  0.10E-03

FCN=   90.66560     FROM MIGRAD   STATUS=CONVERGED   CALLS=  152 EDM=  0.68E-05
                    STRATEGY= 1    ERROR DEF=    1.0000

INT EXT  PARAMETER                                  STEP          FIRST
NO. NO.    NAME         VALUE         ERROR          SIZE       DERIVATIVE
  1   1  Constant       300.28       5.0681       0.13342       0.97075E-04
  2   2  Mean          0.30698      0.10511E-02  -0.13885E-04  -0.57797
```

Parameter	Input value	Result of Figure 6.10	Result of Figure 6.11
First Gaussian:			
Height	1. (normalised)	$300. \pm 5.$	$308. \pm 5.$
Mean value	0.3	0.307 ± 0.001	0.303 ± 0.001
Width (sigma)	0.07	0.074 ± 0.001	0.070 ± 0.001
Second Gaussian:			
Height	0.5 (normalised)	$153. \pm 3.$	$154. \pm 4.$
Mean value	0.7	0.702 ± 0.002	0.703 ± 0.002
Width (sigma)	0.12	0.120 ± 0.002	0.119 ± 0.002

Table 6.3: Results for the fitted parameters of the gaussian distributions as compared to the initial values which the gaussian distributions were generated in the "batch" job in Section 6.3. The table also includes the result of the double gaussian fit in section 6.11.

```
  3  3  Sigma        0.73832E-01   0.67896E-03  -0.57602E-04   -4.6407

CHISQUARE = 0.2159E+01  NPFIT =   45

PAW > hi/fit 110(50:99) G 0 0 par(4)  | Fit second half with gaussian, do not plot

    *********************************************
    *                                           *
    * Function minimization by SUBROUTINE HFITGA *
    * Variable-metric method                    *
    * ID =        110  CHOPT = TR               *
    *                                           *
    *********************************************
Convergence when estimated distance to minimum (EDM) .LT.  0.10E-03

FCN=   30.16534     FROM MIGRAD    STATUS=CONVERGED  CALLS=  221 EDM=  0.87E-04
                    STRATEGY= 1    ERROR DEF=    1.0000

INT EXT  PARAMETER                                  STEP         FIRST
NO. NO.    NAME        VALUE          ERROR          SIZE      DERIVATIVE
  1  1  Constant      153.27        3.0227       0.65005E-01   0.36877E-02
  2  2  Mean         0.70186       0.19599E-02   0.40388E-03    4.8103
  3  3  Sigma        0.11965       0.18242E-02  -0.25292E-03    6.9011

CHISQUARE = 0.6418E+00  NPFIT =   50

PAW > hi/plot 110 SFUNC     | Plot result of fit on Same plot
PAW > ve/pr par(1:6)        | Print the fitted parameters in PAR
PAR (    1 ) =   300.2846
PAR (    2 ) =  0.3069752
PAR (    3 ) =  0.7383241E-01
PAR (    4 ) =   153.2716
PAR (    5 ) =  0.7018576
PAR (    6 ) =  0.1196475
```

Example of a more complex fit

```
PAW > * Create vector of 6 elements and give initial values for combined fit of two gaussians
PAW > ve/cr par2(6) r 200 0.3 0.1 100 0.7 0.1  | initial values for the 6 fit parameters
PAW > set fit 111                              | display fitted parameters plus errors
PAW > hi/fit  110(2:99) G+G  ! 6 par2          | perform the fit (sum of 2 gaussians)
```

```
 *********************************************
 *                                           *
 * Function minimization by SUBROUTINE HFITH *
 * Variable-metric method                    *
 * ID =        110  CHOPT = R                *
 *                                           *
 *********************************************
 Convergence when estimated distance to minimum (EDM) .LT.  0.10E-03

 FCN=   57.41251     FROM MIGRAD    STATUS=CONVERGED  CALLS=  597 EDM=  0.10E-03
                     STRATEGY= 1    ERROR DEF=    1.0000

 INT EXT  PARAMETER                                  STEP         FIRST
 NO. NO.    NAME        VALUE          ERROR          SIZE      DERIVATIVE
   1   1  P1            307.86        5.3896        1.3393      -0.51814E-03
   2   2  P2           0.30265       0.10750E-02   0.18577E-03    3.5622
   3   3  P3           0.70029E-01   0.86285E-03   0.19967E-03    11.689
   4   4  P4            153.62        3.0170       0.73111       0.30406E-02
   5   5  P5           0.70303       0.20652E-02   0.43051E-03   -1.2694
   6   6  P6           0.11865       0.18645E-02   0.39360E-03    3.2237

 CHISQUARE = 0.6524E+00  NPFIT =   94
```

6.7 Doing more with Minuit

When the HISTO/FIT or VECTOR/FIT command is invoked, PAW/HBOOK will set a default environment for Minuit. Control may be given to Minuit if the option "M" is specified in the command. In this case, the user may enter Minuit control statements.

Overview of available MINUIT commands

CLEar

Resets all parameter names and values to undefined. Must normally be followed by a PARAMETER command or equivalent, in order to define parameter values.

CONtour par1 par2 [devs] [ngrid]

Instructs MINUIT to trace contour lines of the user function with respect to the two parameters whose external numbers are **par1** and **par2**. Other variable parameters of the function, if any, will have their values fixed at the current values during the contour tracing. The optional parameter [**devs**] (default value 2.) gives the number of standard deviations in each parameter which should lie entirely within the plotting area. Optional parameter [**ngrid**] (default value 25 unless page size is too small) determines the resolution of the plot, i.e. the number of rows and columns of the grid at which the function will be evaluated.

EXIT

End of Interactive MINUIT. Control is returned to PAW.

FIX parno

Causes parameter **parno** to be removed from the list of variable parameters, and its value will remain constant (at the current value) during subsequent minimizations, etc., until another command changes its value or its status.

HELP [SET] [SHOw]

Causes MINUIT to list the available commands. The list of SET and SHOw commands must be requested separately.

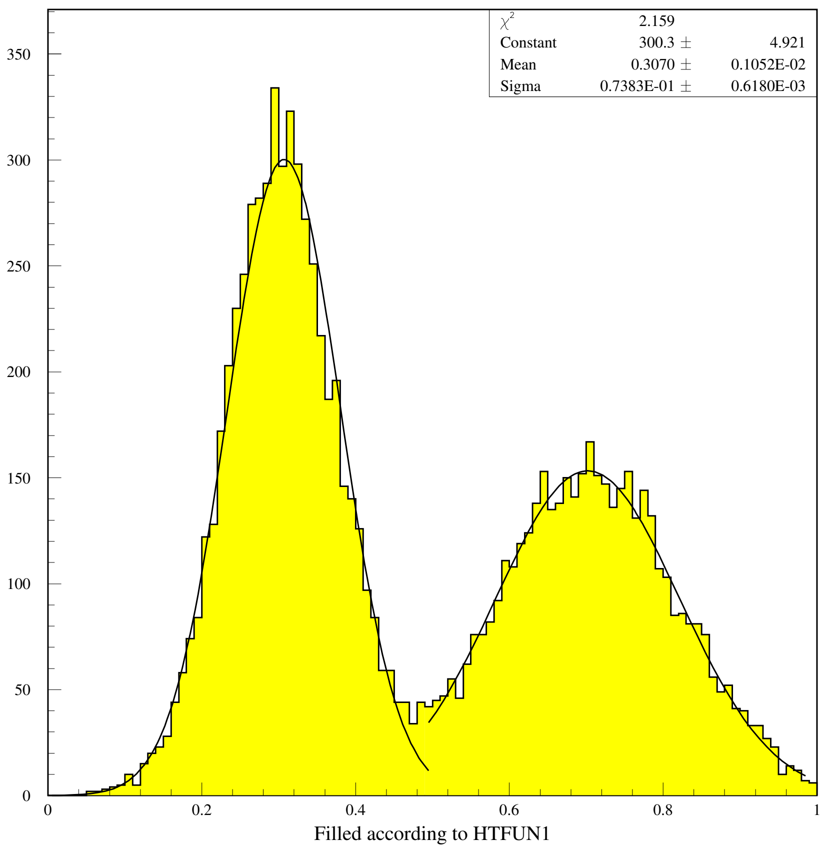


Figure 6.10: Example of a fit using sub-ranges bins

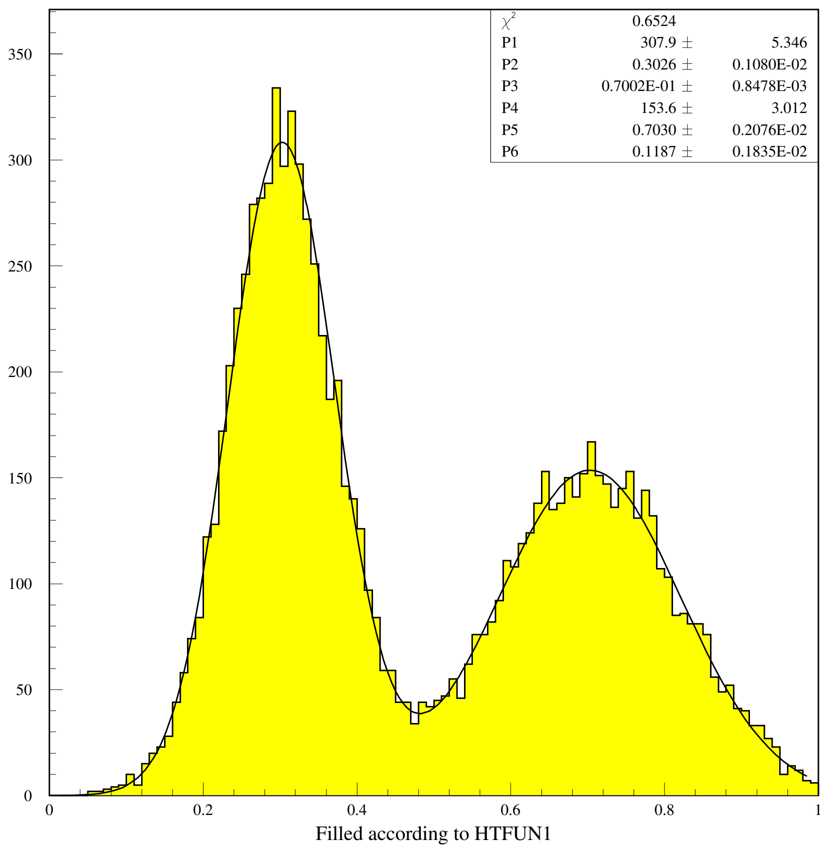


Figure 6.11: Example of a fit using a global double gaussian fit

HESse [maxcalls]

Instructs MINUIT to calculate, by finite differences, the Hessian or error matrix. That is, it calculates the full matrix of second derivatives of the function with respect to the currently variable parameters, and inverts it, printing out the resulting error matrix. The optional argument [**maxcalls**] specifies the (approximate) maximum number of function calls after which the calculation will be stopped.

IMProve [maxcalls]

If a previous minimization has converged, and the current values of the parameters therefore correspond to a local minimum of the function, this command requests a search for additional distinct local minima. The optional argument [**maxcalls**] specifies the (approximate) maximum number of function calls after which the calculation will be stopped.

MIGrad [maxcalls] [tolerance]

Causes minimization of the function by the method of Migrad, the most efficient and complete single method, recommended for general functions (see also MINImize). The minimization produces as a by-product the error matrix of the parameters, which is usually reliable unless warning messages are produced. The optional argument [**maxcalls**] specifies the (approximate) maximum number of function calls after which the calculation will be stopped even if it has not yet converged. The optional argument [**tolerance**] specifies required tolerance on the function value at the minimum. The default tolerance is `0.1`. Minimization will stop when the estimated vertical distance to the minimum (`EDM`) is less than `0.001*[tolerance]*UP` (see `SET ERR`).

MINImize [maxcalls] [tolerance]

Causes minimization of the function by the method of Migrad, as does the MIGrad command, but switches to the SIMplex method if Migrad fails to converge. Arguments are as for MIGrad.

MINOs [maxcalls] [parno] [parno] ...

Causes a Minos error analysis to be performed on the parameters whose numbers [**parno**] are specified. If none are specified, Minos errors are calculated for all variable parameters. Minos errors may be expensive to calculate, but are very reliable since they take account of non-linearities in the problem as well as parameter correlations, and are in general asymmetric. The optional argument [**maxcalls**] specifies the (approximate) maximum number of function calls *per parameter requested,* after which the calculation will be stopped for that parameter.

RELease parno

If **parno** is the number of a previously variable parameter which has been fixed by a command: **FIX parno**, then that parameter will return to variable status. Otherwise a warning message is printed and the command is ignored. Note that this command operates only on parameters which were at one time variable and have been FIXed. It cannot make constant parameters variable; that must be done by redefining the parameter with a PARAMETER command.

REStore [code]

If no [**code**] is specified, this command restores all previously FIXed parameters to variable status. If [**code**]**=1**, then only the last parameter FIXed is restored to variable status.

SCAn [parno] [numpts] [from] [to]

Scans the value of the user function by varying parameter number [**parno**], leaving all other parameters fixed at the current value. If [**parno**] is not specified, all variable parameters are scanned in sequence. The number of points [**numpts**] in the scan is 40 by default, and cannot exceed 100. The range of the scan is by default 2 standard deviations on each side of the current best value, but can be specified as from [**from**] to [**to**]. After each scan, if a new minimum is found, the best parameter values are retained as start values for future scans or minimizations. The curve resulting from each scan is plotted on the output unit in order to show the approximate behavior of the function. This command is not intended for minimization, but is sometimes useful for debugging the user function or finding a reasonable starting point.

SEEk [**maxcalls**] [**devs**]

Causes a Monte Carlo minimization of the function, by choosing random values of the variable parameters, chosen uniformly over a hypercube centered at the current best value. The region size is by default 3 standard deviations on each side, but can be changed by specifying the value of [**devs**].

SET ERRordef up

Sets the value of **up** (default value= 1.), defining parameter errors. MINUIT defines parameter errors as the change in parameter value required to change the function value by **up**. Normally, for chisquared fits **up=1**, and for negative log likelihood, **up=0.5**.

SET LIMits [**parno**] [**lolim**] [**uplim**]

Allows the user to change the limits on one or all parameters. If no arguments are specified, all limits are removed from all parameters. If [**parno**] alone is specified, limits are removed from parameter [**parno**]. If all arguments are specified, then parameter [**parno**] will be bounded between [**lolim**] and [**uplim**]. Limits can be specified in either order, MINUIT will take the smaller as [**lolim**] and the larger as [**uplim**]. However, if [**lolim**] is equal to [**uplim**], an error condition results.

SET PARameter parno value

Sets the value of parameter **parno** to **value**. The parameter in question may be variable, fixed, or constant, but must be defined.

SET PRIntout level

Sets the print level, determining how much output MINUIT will produce. The allowed values and their meanings are displayed after a **SHOw PRInt** command. Possible values for **level** are:

- `-1` No output except from SHOW commands
- `0` Minimum output (no starting values or intermediate results)
- `1` Default value, normal output
- `2` Additional output giving intermediate results.
- `3` Maximum output, showing progress of minimizations.

SET STRategy level

Sets the strategy to be used in calculating first and second derivatives and in certain minimization methods. In general, low values of **level** mean fewer function calls and high values mean more reliable minimization. Currently allowed values are 0, 1 (default), and 2.

SHOw XXXX

All **SET XXXX** commands have a corresponding **SHOw XXXX** command. In addition, the SHOw commands listed starting here have no corresponding SET command for obvious reasons. The full list of SHOw commands is printed in response to the command **HELP SHOw**.

SHOw CORrelations

Calculates and prints the parameter correlations from the error matrix.

SHOw COVariance

Prints the (external) covariance (error) matrix.

SIMplex [**maxcalls**] [**tolerance**]

Performs a function minimization using the simplex method of Nelder and Mead. Minimization terminates either when the function has been called (approximately) [**maxcalls**] times, or when the estimated vertical distance to minimum (`EDM`) is less than [**tolerance**]. The default value of [**tolerance**] is `0.1*UP` (see `SET ERR`).

Chapter 7: Graphics (HIGZ and HPLOT)

7.1 HPLOT, HIGZ and local graphics package

Graphics input/output in PAW is handled by the two packages HPLOT (Histograms PLOTting) and HIGZ (High level Interface to Graphics and Zebra). HIGZ is the basic graphics system of PAW interfacing an basic graphics package while HPLOT, sitting on top of HIGZ, is used for plotting HBOOK objects (Histograms, Ntuples, etc.). The figure below shows the hierarchy between HPLOT, HIGZ and the basic graphics package (X Windows, etc...).

Graphics could be produced in PAW either directly by HIGZ commands or by HPLOT commands. In both cases, all the graphics is under the control of HIGZ. Two distinct modes are available in HIGZ: one is purely graphics (the `G` mode) interfacing the basic graphics package, and the second (the `Z` mode) allows the management of the HIGZ structures (pictures). As an example, the simple PAW command `HISTOGRAM/PLOT` is handled at the different levels as follows:

PAW Level	`HISTOGRAM/PLOT ID`
HPLOT Level	Takes care of `ZONE`, `SET`, `OPTION`, etc.
HIGZ Level	Windows and Viewport, Axis, Boxes, Histogram, Text and Attributes
Basic graphics	Line, Text, Attributes, etc.

7.2 The metafiles

Metafiles are text files used as device independent sources of graphics output for printers of different type. The most widely use metafile in PAW is the PostScript metafile. This type of metafile can be sent directly to a PostScript printer The PostScript metafile type (second parameter of the comman `METAFILE` have the following format:

`-[Format][Nx][Ny][Type]`

Where:

`Format` Is an integer between 0 and 99 which defines the format of the paper. For example if `Format`=3 the paper is in the standard A3 format. `Format`=4 and `Format`=0 are the same and define an A4 page. The A0 format is selected by `Format`=99. The US format Letter is selected by `Format`=100. The US format Legal is selected by `Format`=200. The US format Ledger is selected by `Format`=300.

`Nx, Ny` Specify respectively the number of zones on the x and y axis. `Nx` and `Ny` are integers between 1 and 9.

`Type` Can be equal to:

- `1` Portrait mode with a small margin at the bottom of the page.
- `2` Landscape mode with a small margin at the bottom of the page.
- `4` Portrait mode with a large margin at the bottom of the page.
- `5` Landscape mode with a large margin at the bottom of the page.
  The large margin is useful for some PostScript printers (very often for the colour printers) as they need more space to grip the paper for mechanical reasons.
  Note that some PostScript colour printers can also use the so called "special A4" format permitting the full usage of the A4 area; in this case larger margins are not necessary and `Type`=1 or 2 can be used.
- `3` Encapsulated PostScript. This `Type` permits the generation of files which can be included in other documents, for example in LaTeX files. Note that with this `Type`, `Nx` and `Ny` must always be equal to 1, and `Format` has no meaning. The size of the picture must be specified by the user via the `SIZE` command. Therefore the workstation type for Encapsulated PostScript is -113. 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}
\includegraphics{example.eps}
\caption{Example of Encapsulated PostScript in LaTeX.}
\label{EXAMPLE}
\end{figure}
```

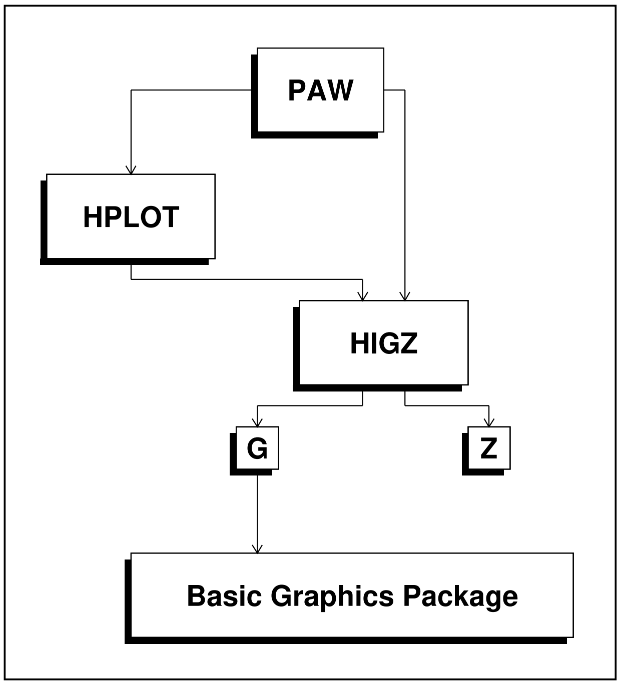


Figure 7.1: HPLOT and HIGZ in PAW

Note that all the figures in this manual are included in this way.

With `Type=1,2,4` and `5` the pictures are centered on the page, and the usable area on paper is proportional to the dimensions of A4 format.

Examples:

`-111` or `-4111` defines an A4 page not divided. `-6322` define an A6 landscape page divided in 3 columns and 2 rows.

1	2	3
4	5	6

The first picture will be drawn in the area 1. The next image will appear in the next area in the order defined above. If a page is filled, a new page is used with the same grid. Note that empty pages are not printed in order to save paper.

Ignoring formats smaller than A12, the total number of possible different PostScript workstation types is: $4 \times 9 \times 9 \times 13 + 1 = 4213$!

The command `GRAPHICS/METAFILE LUN METAFL` is designed to produce metafiles. `LUN` is the logical unit number of an open FORTRAN file and `METAFL` the metafile type. For example, the following four commands will produce a HIGZ/PostScript metafile with the name `"PAW.PS"` containing the graphics representation of histogram number `10`:

```
PAW > FORTRAN/FILE 66 PAW.PS
PAW > GRAPHICS/META 66 -111
PAW > HISTO/PLOT 10
PAW > FORTRAN/CLOSE 66
```

7.3 The HIGZ pictures

The HIGZ pictures have four main goals:

- HIGZ graphics primitives and attributes can be stored in a ZEBRA structure in memory in order to display them later.

- They can be stored on direct access files (in a very compact way), in order to build a picture data base.
- They can be modified with the graphics editor.
- They are structured i.e. they can contains so called “graphics objects” which are used to retrieve objects names and type in the “direct graphics mode” of PAW++.

7.3.1 Pictures in memory

The general command to manage pictures in memory is: PICTURE/IZPICT. This command has two parameters:

PNAME Picture name:

CH	Character string specifying picture name (must begin with a letter)
N	Picture number as displayed by PICT/LIST.
*	All pictures in memory.
' '	A blank indicates the current picture.

CHOPT Option value:

AL	Give a full listing of the pictures in memory.
C	Picture PNAME becomes the current picture.
D	Display the picture PNAME.
F	First picture in memory becomes the current picture.
L	List pictures in memory.
M	Make a new picture in memory with the name PNAME.
N	Next picture in memory becomes the current picture.
P	Print the contents of the picture PNAME.
S	Scratch picture PNAME from memory.

In addition, simpler and more mnemonic commands are available:

```
PAW > PICT/CREATE PNAME           | Create a picture in memory
PAW > PICT/LIST                   | List pictures in memory
 1: PNAME <-- Current Picture
```

The last created picture in memory is called the **current** picture. All graphics primitives (line, text, histogram, etc.) produced by PAW commands will be stored in this picture if it is **active**, i.e. if mode Z is on.

```
PAW > SWITCH Z                     | Switch Z mode on
PAW > PICT/LIST
 1: PNAME <-- Current Picture (Active)
```

Note that the command PICTURE/CREATE will switch automatically Z mode on.

```
PAW > PICT/PLOT PNAME
```

will display picture PNAME. If picture PNAME is not in memory and if the current working directory (as given by CDIR) is a picture file, PAW will try to take this picture from the file before displaying it.

HIGZ pictures can be created automatically by HPLOT via the command:

```
PAW > OPTION ZFL
```

If this command has been typed, each new plot produced by HPLOT will result in a HIGZ picture created in memory. The following example shows how for each HIST/PLOT ID command a new HIGZ picture is created with an automatic naming:

```
PAW > HIST/PLOT 10
PAW > HIST/PLOT 110
PAW > HIST/PLOT 20
PAW > PICT/LIST
 1: PICT1
 2: PICT2
 3: PICT3 <-- Current Picture (Active)
```

A similar command is given by:

```
PAW > OPTION ZFL1
```

which works exactly like `OPTION ZFL` except that only the last created picture is kept in memory. For example, if we had typed `OPTION ZFL1` instead of `OPTION ZFL` in the example above, the result would be:

```
PAW > PICT/LIST
 1: PICT3 <-- Current Picture (Active)
```

The following example is a useful macro showing how to use the HIGZ pictures (via `OPTION ZFL1`) and the metafiles in order to produce a hard copy of the graphics screen:

Macro showing how to convert the current picture in PostScript

```
MACRO POST
FORTRAN/FILE 66 PAW.PS  | Open the FORTRAN file PAW.PS on unit 66
META -66 -111           | PAW.PS is an A4 PostScript file
PICT/PLOT ' '           | Convert the current picture in PostScript
CLOSE 66                | Close PAW.PS
SHELL PRINT PAW.PS      | Send PAW.PS to the local printer
RETURN
```

Typing `EXEC POST`, the current HPLOT picture on the screen will be sent to the printer using the `SHELL` command which issues a system-dependent "`print`" command to the local operating system (e.g. `lp` or `lpr` on Unix).
The command `PICTURE/PRINT` do the same thing:

```
PAW > PICT/PRINT PAW.PS
```

This command transform the current picture into a printable file. The file type is defined according to the extension of the file name i.e.

- **FILE = filename.ps** A PostScript file is generated (-111)
- **FILE = filename.eps** A Encapsulated PostScript file is generated (-113)
- **FILE = filename.tex** A LaTex file is generated (-778)

With this command the metafile type is predefined. It is not possible to change it like in the macro `POST` previously described. If `FILE=HIGZPRINTER` or `FILE=' '` the PostScript file paw.ps (-111) is generated and the operating system command defined by the environment variable HIGZPRINTER is executed. The environment variable HIGZPRINTER could be defined as follow:

```
setenv HIGZPRINTER 'xprint -p513-pub paw.ps'
```

Note that if the environment variable `HIGZPRINTER` is not defined the file `paw.ps` is created but not printed.
Other available commands working on pictures in memory are:

```
PAW > PICT/RENAME PNAME PNAME2
PAW > PICT/COPY PNAME PNAME2
PAW > PICT/DELETE PNAME
```

- `PNAME` can be the complete name, the picture number in memory or `' '`.
- `PNAME2` is the complete picture name.

7.3.2 Pictures on direct access files

HIGZ pictures are stored on direct-access files and hence access times to pictures are fast. Moreover, due to the fact that HIGZ uses high level primitives to describe the picture's structural tree, a storage compaction factor as compared to the equivalent GKS metafiles of between 10 and 100 is routinely obtained.

As HIGZ is interfaced to various basic graphics packages, a picture file can be created on one system (e.g. DECGKS, X11, GL etc.) and transported to another machine to be interpreted with a different graphics package (e.g GKSGRAL, GDDM, DI3000 etc.).

All available commands to handle pictures with ZEBRA files are shown below. Note that in the example the picture names could be "*" (all pictures in memory), " " (current picture) or a number (picture number in memory).

Handling pictures with ZEBRA

```
PAW > * Open an existing picture file PICT.DAT on LUN 4 in Update mode
PAW > PICT/FILE 4 PICT.DAT ! U  | Open the existing file PICT.DAT
PAW > LDIR                      | List the content of the file PICT.DAT

************** Directory ===> //LUN4 <===

                 Created 890512/1110  Modified 890622/1732

===> List of objects
              PICTURE   NAME                          CYCLE
    UNIX                                                 1
    ZEBRA                                                1
    CERN                                                 1
    MARKER                                               1

PAW > IZIN CERN                 | Put picture "CERN" in memory
PAW > PICT/LIST                 | List pictures in memory
 1: CERN
PAW > IZOUT CERN                | Store picture "CERN" in PICT.DAT
PAW > LDIR                      | List the content PICT.DAT

************** Directory ===> //LUN4 <===

                 Created 890512/1110  Modified 890622/1732

===> List of objects
              PICTURE   NAME                          CYCLE
    UNIX                                                 1
    ZEBRA                                                1
    CERN                                                 1
                                                         2
    MARKER                                               1

PAW > PURGE                     | Purge the file PICTURES
PAW > SCRATCH ZEBRA             | Delete the picture ZEBRA from PICT.DAT
PAW > LDIR                      | List the content of PICT.DAT

************** Directory ===> //LUN4 <===

                 Created 890512/1110  Modified 890622/1732

===> List of objects
              PICTURE   NAME                          CYCLE
    UNIX                                                 1
    CERN                                                 2
    MARKER                                               1
```

7.3.3 Automatic storage pictures in memory

After typing the command:

```
PAW > SET AURZ 1
```

the AURZ mode is on and all the subsequent created pictures are stored automatically in the last picture file opened via the command PICTURE/FILE.

Example of the use of pictures in memory

```
PAW > PICT/FILE 4 PICT.DAT ! N    | Open a new picture file PICT.DAT
PAW > HIST/FILE 3 HEXAM.DAT       | Open an existing histogram RZ file
PAW > LDIR                        | List the contain of HEXAM.DAT

 ************** Directory ===> //LUN3 <===

                  Created 880104/1414  Modified 880104/1414

 ===> List of objects
     HBOOK-ID  CYCLE   DATE/TIME   NDATA   OFFSET    REC1    REC2
         10       1   880104/1414     75     725      32
         20       1   880104/1414   1815     800      32      33
         30       1   880104/1414   1066     567      34      35

PAW > OPT ZFL       | Each new plot will result in a HIGZ picture
PAW > SET AURZ 1    | Each new HIGZ picture is stored in PICT.DAT
PAW > HIST/PLOT 0   | All histograms in HEXAM.DAT are plotted
PAW > CDIR //LUN4   | Set the current working directory on PICT.DAT
PAW > LDIR          | List the content of PICT.DAT

 ************** Directory ===> //LUN4 <===

                  Created 890928/1024  Modified 890928/1024

 ===> List of objects
               PICTURE   NAME                         CYCLE
     PICT1                                               1
     PICT2                                               1
     PICT3                                               1
```

Note that if the command PICTURE/FILE is invoked with the option 'A', the AURZ mode is automatically enable.

7.3.4 HIGZ pictures generated in a HPLOT program

HIGZ pictures can be generated in a batch HPLOT program and later visualized in an interactive session with PAW. The HIGZ picture file, like any HBOOK file, can be exchanged between computers using the FTP in binary mode. As the size of the picture data base (see page 101), and hence the associated disk storage requirements, is much smaller than the size of the metafile generated by the basic graphics package, transfer times are drastically reduced. The example below show how to interactively visualize (with PAW) HIGZ pictures produced by HPLOT. In the same way we can visualize and edit pictures generated by any HIGZ based application (GEANT, event scanning programs, etc.)

7.4 Setting attributes

Attributes are parameters like: colour, character font, etc. which could be changed interactively in PAW via the commands PICTURE/IGSET, GRAPHICS/SET and GRAPHICS/OPTION. Each attribute is linked to one or more objects (lines, histogram, etc.). The aim of this section is to give a complete description of the attributes available in PAW and to clarify the differences between IGSET, which changes attributes at the HIGZ level, and SET and OPTION, which act at the HPLOT level.

Store HPLOT pictures with HIGZ

```
      PROGRAM HPICT
*.==========>
*.  HPLOT Program to demonstrate how to store HPLOT
*.  pictures onto direct access HIGZ picture file
*..=========>
      COMMON/PAWC/H(20000)
      DIMENSION SIG(2)
      CHARACTER*20 TITLE
*.-------------------------------------------
*.
      CALL HLIMIT(20000)
* --        Create histograms
      DO 10 ID=1,10
         WRITE(TITLE,1000)ID
 1000    FORMAT('Test number',I3)
         CALL HBOOK1(ID,TITLE,100,-3.,3.,0.)
  10  CONTINUE
* --        Fill histograms
      DO 30 ID=1,10
         DO 20 I=1,1000
            CALL RANNOR(A,B)
            CALL HFILL(ID,A,0.,1.)
  20     CONTINUE
         CALL HFITGA(ID,COEFF,AV,SIGM,CHI2,2,SIG)
  30  CONTINUE
* --       Initialize HPLOT. Set various graphics options.
      CALL HPLINT(0)
      CALL HPLZON(1,2,1,' ')
      CALL HPLOPT('ZFL',1)
      CALL HPLOPT('FIT',1)
      CALL HPLOPT('STAT',1)
      CALL HPLSET('STAT',1.)
      CALL HPLSET('HTYP',244.)
      CALL HPLSET('FWID',5.)
      CALL HPLSET('VFON',-40.)
      CALL HPLSET('TFON',-60.)
      CALL HPLSET('PWID',4.)
      CALL HPLSET('BCOL',1.01)
      CALL HPLSET('CSIZ',0.25)
      CALL HPLSET('CFON',-10.)
*
*   Open a picture file called "hpict.dat".
*   Option 'A' means "Automatic saving of pictures"
*   Option 'N' means "New file"
*   (option 'U' instead of 'N' updates an existing file)
*
      CALL IZOPEN(1,'Pictures','hpict.dat','AN',1024,ISTAT)
*
*   Select HIGZ option to store graphics in ZEBRA memory only
*   No calls to the local graphics package.
*
      CALL IGZSET('Z')
* --      Plot all histograms
      CALL HPLOT(0,' ',' ',0)
      CALL HPLEND
*
      END
```

Using the picture in Paw

```
PAW > PICT/FILE 20 HPICT.DAT
PAW > LDIR
         Directory ===> //LUN20 <===

  Created 891006/1026  Modified 891006/1026

===> List of objects
    PICTURE  NAME                 CYCLE
       PICT1                        1
       PICT2                        1
       PICT3                        1
       PICT4                        1
       PICT5                        1
PAW > META 10 -111
PAW > PICT/PLOT PICT2
PAW > CLOSE 10
PAW > * Print metafile
PAW > *    (see pages 101 and following)
PAW > SHELL print PAW.METAFILE
PAW > EXIT
```

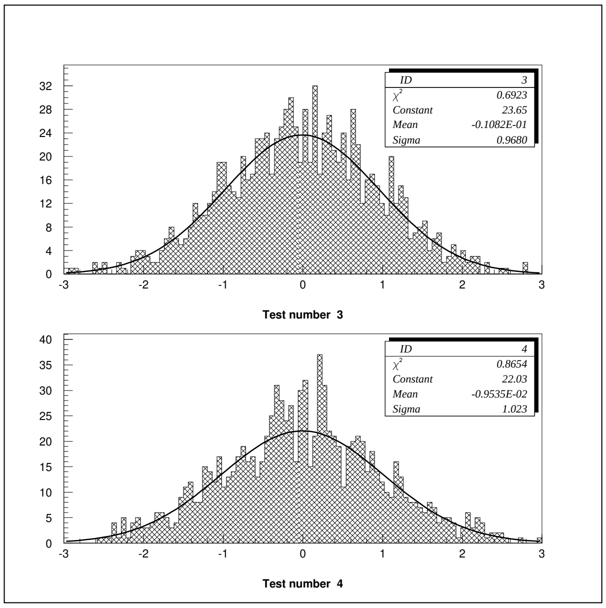


Figure 7.2: Visualising a HIGZ picture produced in a batch HPLOT program

IGSET [CHOPT VAL]

This command is used to set the value of attributes related to primitives and macroprimitives. The first parameter is the mnemonic name of the attribute, the second is the value to be assigned.

`CHOPT` Character variable specifying the name of the attribute to be set. This a character string of 4 characters.

`VAL` Value of the attribute. A value of `0` or no value specified, indicates that the attribute value must be reset to its default value.

Examples of IGSET commands

```
PAW > IGSET MTYP 20        | Change marker type to 20.
                           | This new marker is used by all subsequent
                           | commands using the current marker type.

PAW > IGSET LWID           | Set the line width to its default value.

PAW > IGSET                | Display actual and default values of all HIGZ attributes
PAW > IGSET *              | Set ALL HIGZ attributes to their default values
```

Note that the command `SET` calls `IGSET` if it is called with a `IGSET` option.

OPTION `[ CHOPT ]`

The `OPTION` command has one optional parameter:

`CHOPT` Option name (four characters). Special values are:

- `'*'` Set all HPLOT options to their default values
- `' '` Display actual and default values of all HPLOT options

SET `[ CHOPT VAL ]`

Sets an HPLOT parameter; see table 7.3 and figures 7.3, 7.4, 7.5 and 7.6 for details.

`CHOPT` Character variable of length 4 identifying the parameter to be redefined (must be given in uppercase). Special values are:

- `'*'` All parameters are set to their default values.
- `'SHOW'` A list of all parameters and their values is printed.

`VAR` New value for the parameter specified. Special values are:

- `0.` The corresponding parameters is set to its default value.

Table 7.1: Parameters and default values for `IGSET`

NAME	default	Explanation
'AURZ'	0.	If 1. the last current picture is automatically saved on disk when a new picture is created.
'AWLN'	0.0	Axis wire length. Default is length=0 (no grid)
'BARO'	0.25	Offset of the left edge of the bar with respect to the left margin of the bin for a bar chart (expressed as a fraction of the bin width).
'BARW'	0.50	Width of the bar in a bar chart (expressed as a fraction of the bin width).
'BASL'	0.01	Basic segment length in NDC space (0-1) by (0-1) for dashed lines
'BORD'	0.	Border flag. If = 1., a border is drawn in boxes, pie charts,....
'CHHE'	0.01	CHaracter HEight.
'CSHI'	0.02	Distance between each shifted drawing of a character (in percentage of character height) for characters drawn by TEXT
'FACI'	1.	Fill Area Colour Index.
'FAIS'	0.	Fill Area Interior Style (0.,1.,2.,3.).
'FASI'	1.	Fill Area Style Index.
'LAOF'	0.013	LAbels OFfset.
'LASI'	0.018	LAbels SIze (in World coordinates).
'LTYP'	1.	Line TYPe.
'LWID'	1.00	Line WIDth.
'MSCF'	1.00	Marker SCale Factor.
'MTYP'	1.	Marker TYPe.
'PASS'	1.	Text width (given by number of PASSes) of characters drawn by TEXT. The width is simulated by shifting the “pen” slightly at each pass.
'PICT'	1.	Starting number for automatic pictures naming.
'PLCI'	1.	PolyLine Colour Index.
'PMCI'	1.	PolyMarker Colour Index.
'TANG'	0.00	Text ANGle (for calculating Character up vector).
'TMSI'	0.019	Tick Marks SIze (in world coordinates)
'TXAL'	0.	10*(horizontal alignment)+(vertical alignment).

Table 7.1: Parameters and default values for IGSET (continued)

NAME	default	Explanation
'TXCI'	1.	TeXt Colour Index.
'TXFP'	10.	10*(TeXt Font) + (TeXt Precision).
		(0: hard, 1: string, 2: soft)
'*'		All attributes are set to their default values.
'SHOW'		The current and default values of the parameters controlled by IGSET are displayed.

Table 7.2: Parameters and default values for OPTION

Default	**Alternative**	**Effect**
' '	'A0', 'A1',...	Picture size. Predefined options are: A0, A1, A2, A3, A4, A5, A6
'NOPG'	'*P','**P', '***P'	Suppresses ('NOPG') or adds a 1, 2 or 3 digit page numbers to a plot (Each '*' stands for a digit). The page numbers are incremented automatically
'NEAH'	'EAH'	Plots Errors bars And Histogram, if both are present
'VERT'	'HORI'	Vertical or horizontal orientation of paper
'NAST'	'AST'	Functions are drawn with ('AST ') or without ('NAST') asterisks in each channel.
'NCHA'	'CHA'	Scatter plot are plotted with dots randomised within each bin ('NCHA') or by printing a single character in the middle of the bin ('CHA ')
'SOFT'	'HARD'	Use SOFTware or HARDware characters
'TAB '	'NTAB'	tables (HTABLE) are plotted as tables ('TAB ') or as scatter plots ('NTAB')
'HTIT'	'UTIT'	Option for printing titles. 'HTIT' means use the hbook titles, while 'UTIT' signals the use of user titles
'LINX'	'LOGX'	The scale for the X axis is linear or logarithmic.
'LINY'	'LOGY'	The scale for the Y axis is linear or logarithmic.
		Note that if in hbook the HIDOPT option 'LOGY' or HLOGAR was selected for a particular ID and if neither options 'LINY' nor 'LOGY' are selected then the scale will be logarithmic. If HLOGAR or HIDOPT with option 'LOGY' was called and the option 'LINY' is selected then the scale will be linear
'LINZ'	'LOGZ'	The scale for the Z axis is linear or logarithmic (for lego plots or surfaces).
'BOX '	'NBOX'	By default a rectangular box is drawn around a picture. 'NBOX' suppresses this box
'NTIC'	'TIC'	Cross-wires are drawn ('TIC ') or not drawn ('NTIC') after each plot
'NSTA'	'STA'	Statistics information are printed ('STA ') or not printed ('NSTA') on the picture
'NFIT'	'FIT'	Fit parameters are printed ('FIT ') or not printed ('NFIT') on the picture
'NSQR'	'SQR'	The size of the histogram boxes is set to the largest square (SQR)
'NZFL'	'ZFL'	The picture is stored ('ZFL ') or not stored ('NZFL') in a ZEBRA data base The procedure to create a higz picture is given below.
'NZFL'	'ZFL1'	'ZFL1' has the same effect as 'ZFL ', but only the picture last created is kept in memory.
'NPTO'	'PTO'	"Please Turn Over". With 'PTO ' a carriage return is requested between each new plot.
'NBAR'	'BAR'	1-dimensional histograms are plotted as "Bar charts" ('BAR ') or as contours ('NBAR')
'DVXR'	'DVXI'	Real ('DVXR') or integer ('DVXI') labels are computed for the X axis
'DVYR'	'DVYI'	Real ('DVYR') or integer ('DVYI') labels are computed for the Y axis
'GRID'	'NGRI'	Grid on X and Y axis
'NDAT'	'NDAT'	The date is printed or not on each plot

Table 7.2: Overview of the HPLOPT options (continued)

Default	Alternative	Effect
'NFIL'	'NFIL'	The file name is printed or not on each plot

Table 7.3: Parameters and default values in SET

CHOPT	VAR (default)	Explanation
ASIZ	0.28 cm	axis label size
BARO	0.25	bar offset for "bar charts"
BARW	0.5	bar width for "bar charts"
BCOL	1	zone fill area colour index
BTYP	0	zone fill area style index
BWID	1	box line width
CFON	2	comment font (10*font+precision)
CSHI	0.03	character shift between two pass
CSIZ	0.28 cm	comment size
DASH	0.15	length of basic dashed segment for dashed lines
DATE	2	date position
DMOD	1	line style for histogram contour (see HPLOT)
ERRX	0.50	error on X (% of bin width)
FCOL	1	function fill area COLor
FILE	1	file name position
FIT	101	fit values to be plotted
FPGN	1	first PaGe Number
FTYP	0	function fill area TYPe
FWID	1	function line width
GFON	2	global title font (10*font+precision)
GRID	3	grid line type
GSIZ	0.28 cm	global title size
HCOL	1	histogram fill area colour index
HMAX	0.90	histogram maximum for scale (in percent)
HTYP	0	histogram fill area style index
HWID	1	histogram line width
KSIZ	0.28 cm	Hershey character size (cf. KEY)
LFON	2	axis labels font (10*font+precision)
NDVX	10510.00	number of divisions for X axis
NDVY	10510.00	number of divisions for Y axis
NDVZ	10510.00	number of divisions for Z axis
PASS	1.	number of pass for software characters
PCOL	1	picture fill area colour index
PSIZ	0.28 cm	page number size
PTYP	0	picture fill area style index
PWID	1	picture line width
SMGR	0.	stat margin right (in percent)
SMGU	0.	stat margin up (in percent)
SSIZ	0.28 cm	asterisk size (for functions)
STAT	1111	stat values to be plotted

Table 7.3: Parameters and default values in SET (continued)

CHOPT	VAR (default)	Explanation
TFON	2	general comments font (10*font+precision)
TSIZ	0.28 cm	histogram title size
VFON	2	axis values font (10*font+precision)
VSIZ	0.28 cm	axis values size
XCOL	1	X axis COLor
XLAB	1.40 cm	distance Y axis to labels
XMGL	2.00 cm	X margin left
XMGR	2.00 cm	X margin right
XSIZ	20.0 cm	length of picture along X
XTIC	0.30 cm	X axis tick mark length
XVAL	0.40 cm	distance between the Y axis and the axis values
XWID	1	X ticks width
XWIN	2.00 cm	X space between zones
YCOL	1	Y axis COLor
YGTI	1.50 cm	Y position of global title
YHTI	1.20 cm	Y position of histogram title
YLAB	0.80 cm	distance X axis to labels
YMGL	2.00 cm	Y margin low
YMGU	2.00 cm	Y margin up
YNPG	0.60 cm	Y position for the page number
YSIZ	20.0 cm	length of picture along Y
YTIC	0.30 cm	Y axis tick mark length
YVAL	0.20 cm	distance between the X axis and the axis values
YWID	1	Y ticks width
YWIN	2.00 cm	Y space between zones
2SIZ	0.28 cm	scatter plot and table character. size

7.5 More on labels

By default, labels used by AXIS and PIE are numeric labels. The command GRAPHICS/PRIMITIVES/LABELS (or LABELS for short), allows the user to define up to nine alphanumeric set of labels (numbered from 1 to 9). These labels can then be used in subsequent commands using PIE or AXIS primitives of HIGZ.

The LABELS command has three parameters:

LABNUM An integer between 1 and 9. It identifies the labels set.

NLABS The number of items to be placed on the labels (up to 50).

CHLABS NLABS character strings specifying the label items.

The label sets thus defined can be used for axes on all plots produced by PAW (HPLOT histograms, graphs, vectors drawing, etc.) via the SET NDVX (NDVY) command. These commands have the following structure:

Example of NXDV specification

```
    SET NDVX i            e.g. SET NDVX 512
or
    SET NDVX i.jk         e.g. SET NDVX 10.25
```

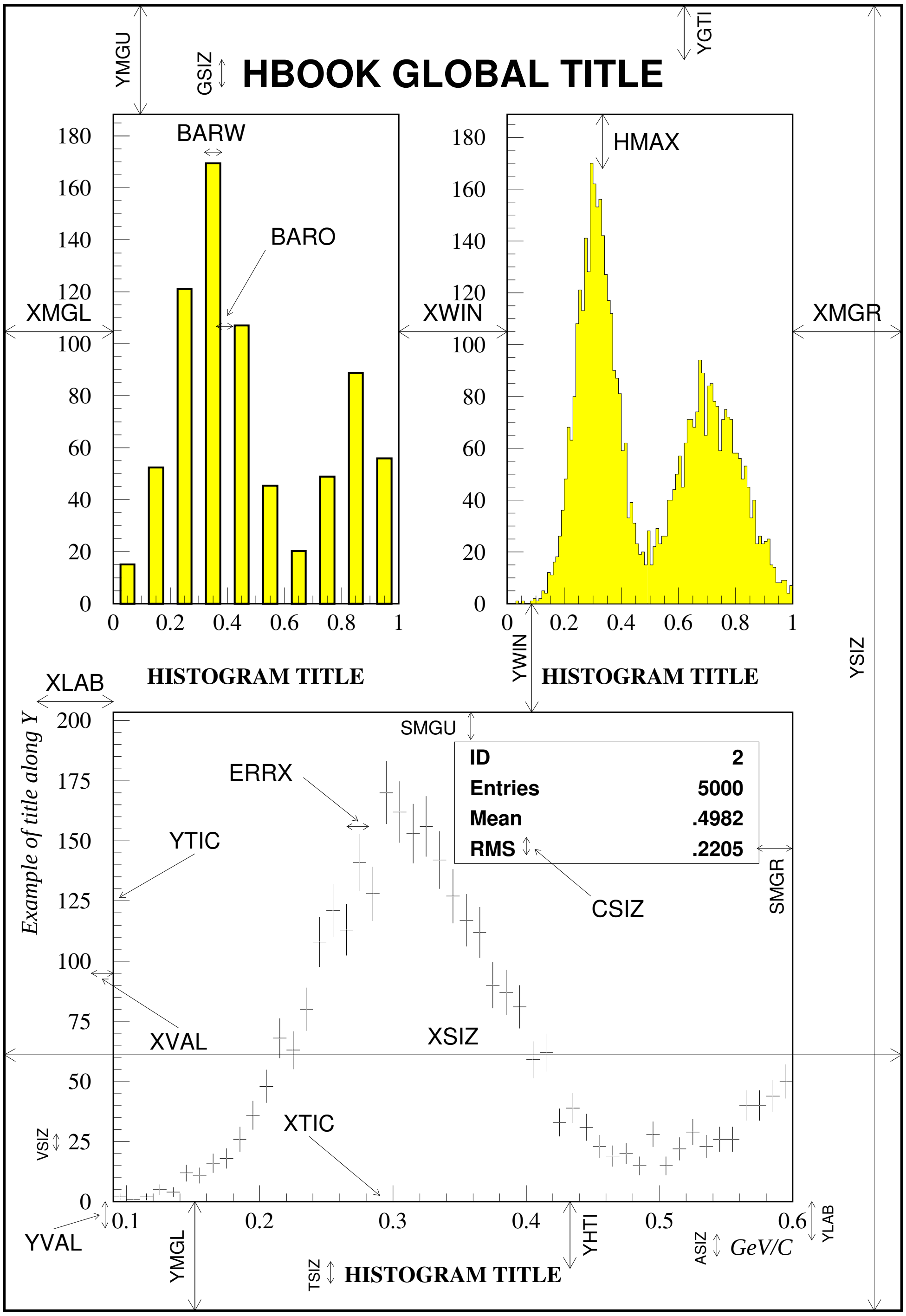


Figure 7.3: A graphical view of the SET parameters

In the first case the number i contains 100 times the number of secondary divisions plus the number of primary divisions. (e.g. 512 means 12 primary and 5 secondary division. By adding 10000 times N3 to i a third level of divisions is available.

In the second case the number in front of the dot (i) indicates the total number of divisions, the first digit following the dot (j) the label identifier (LABNUM) (if this number is equal to 0 numeric labels are drawn). The second digit after the (k) dot indicates the position where the labels have to be drawn (i.e. the *text justification* parameter, in this case 5, indicating horizontally written text centered on the interval). Study figures 7.4 and 7.5 for details. These two figures show that the labels can be centered on the tick marks (1 to 4) or on the divisions (5 to 8). If the labels are centered on the tick marks, note that the number of items in the command LABELS must be equal to the number of tick marks (which is equal to the number of divisions **plus one**), otherwise the last alphanumeric label on the axis will be undefined.

By default, the number of primary divisions given by SET NDVX n, SET NDVY n or SET NDVZ n is optimized to have a reasonable labelling. The number of primary divisions is also optimized according the number of zones (command ZONE) i.e : along the X direction the number of primary divisions is divided by the_number_of_X _zones along the Y direction the number of primary divisions in divided by (the_number_of_Y_zones)/2.

If the number of divisions has to be exactly equal to the number given by SET NDVX n, SET NDVY n or SET NDVZ n, a negative value must be used i.e.:

Forcing an exact number of divisions

```
    SET NDVX -i            e.g. SET NDVX -512
or
    SET NDVX -i.jk         e.g. SET NDVX -10.25
```

For example to label each subsequent X-axis with the names of the months of the year centered in the middle of each bin one can use:

Example of alphanumeric labels on an axis

```
PAW > LABEL 1 12 JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC
PAW > SET NDVX -12.15
```

7.6 Colour, line width, and fill area in HPLOT

The aspect of HPLOT pictures can be modified via the xWID, xTYP and xCOL attributes, where x can be H, B, P, or F, defined as follows:

- B zone Box
- F Function
- H Histogram
- P Page

The values given to the parameters PTYP, BTYP, HTYP, and FTYP are the HIGZ fill area interior styles. Interior style provided by the basic graphics package (i.e. GKS) can be used (cf the corresponding documentation) but in order to have the same result on all devices, numbers greater than 100 (HIGZ styles: 7.7) should be used. Figure 7.6 shows how to use the xTYP parameter.

The parameters PCOL, BCOL, HCOL and FCOL are equivalent to PTYP, BTYP, HTYP, and FTYP respectively, but instead of changing the hatch style, they change the colour of the same areas. It is possible to specify both the border and the inside color for the Histogram, Box Page, and Function (HCOL, BCOL, PCOL, FCOL).

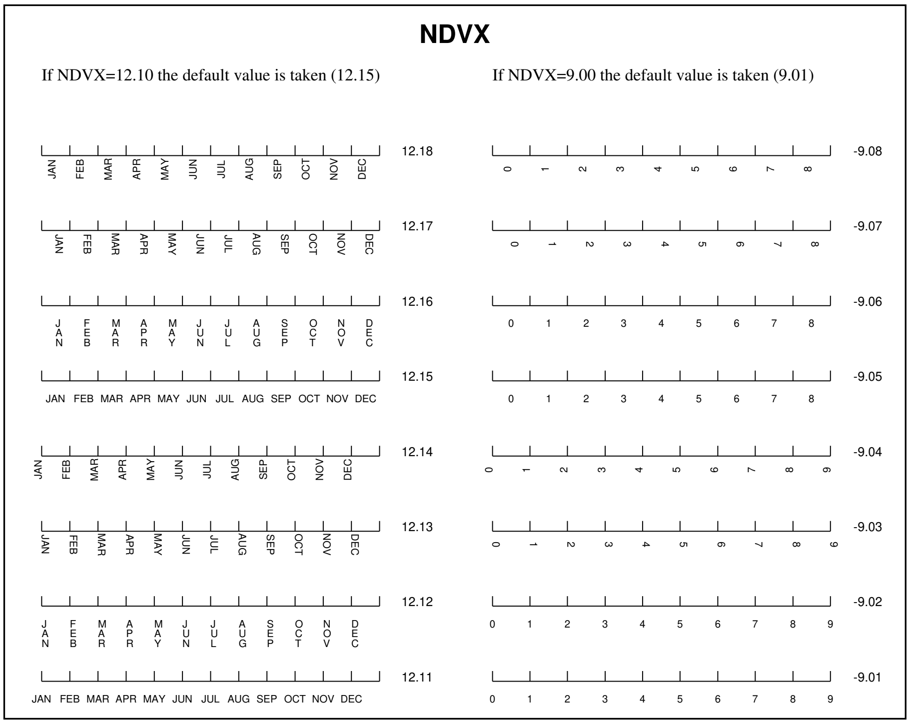


Figure 7.4: Example of labelling for horizontal axes

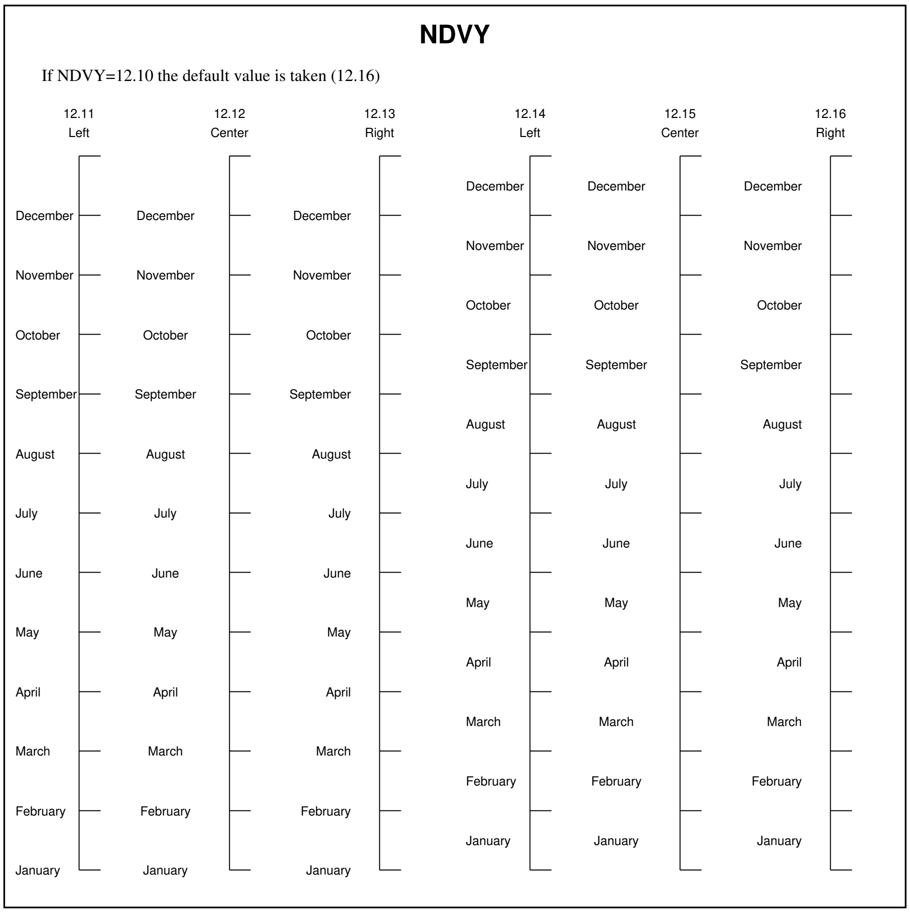


Figure 7.5: Example of labelling for vertical axes

Example of HCOL specification

```
Ex:
            +---- 1 The Histogram is filled
            |     0 Only the border is drawn
            |+--- Border color (here 2) if the histogram is filled
            ||++- Inside color (here 3) if the histogram is filled
            ||||  Border color if the histogram is not filled
            ||||
            VVVV
 SET  HCOL  1203
```

The same mechanism is also available for FCOL, BCOL and PCOL.

If PCOL, BCOL, HCOL or FCOL are between 1 and 99, then only the contour of the corresponding area is changed. If they are between 1001 and 1099, then the surface is filled with the colour determined by the corresponding fill area colour index (1 to 99). If they are between 1199 and 1999, then the surface is filled with the colour determined by the corresponding fill area colour index (1 to 99) and the border is drawn with the corresponding line color index (1 to 9).

If one of the *COL is greater than 1000 the corresponding value of the Fill Area Interior Style (for HTYP, BTYP, PTYP or FTYP) is automatically set to 1 (solid).

In addition, BCOL has two digits after the dot. The first one specifies the colour of the zone box shadowing and the second the colour of the statistic box shadowing.

7.7 Information about histograms

Four options are available to plot additional informations on HPLOT pictures: DATE, FILE, STAT and FIT.

```
PAW > OPTION DATE        | Plot date and hour on current HPLOT picture
PAW > OPTION FILE        | Plot file name of current histogram
PAW > OPTION STAT        | Plot statistics of current histogram
PAW > OPTION FIT         | Plot Fit parameters of current histogram
```

For each of these OPTION commands a corresponding SET parameter is available:

```
PAW > SET DATE i    | Default is 2
PAW > SET FILE i    | Default is 1
```

where i defines the position of the date or file name:

i = 1 : Top left corner of page/current histogram.

i = 2 : Top right corner

i = 3 : Bottom left corner

i = 4 : Bottom right corner

For example the command:

```
PAW > SET DATE 3
```

sets the position of the date to the bottom left corner of the HPLOT pictures.

```
PAW > SET STAT i    | Default is 1111
```

where i corresponds to binary status bits AOURMEI as follows:

A=1 Draw the contents of all channels

O=1 Draw number of overflows

U=1 Draw number of underflows

`R=1` Draw R.M.S.

`M=1` Draw mean value

`E=1` Draw number of entries

`I=1` Draw histogram identifier

For example the command:

```
PAW > SET STAT 10
```

sets the statistics informations to be only the number of entries.

```
PAW > SET FIT i      | Default is 101
```

where `i` corresponds to binary status bits `CEP` as follows:

`C=1` Draw χ^2

`E=1` Draw errors

`P=1` Draw fit parameters

For example to draw only the result of the χ^2 fit one would use:

```
PAW > SET FIT 100
```

For all these `OPTION`s, the **character size** is specified with the command `SET CSIZ` and the character font used with `SET CFON`.

Fill area style, marker and line type

The Fill Area Interior Style, The Fill Area Style Index, the Marker TYPe and the Line TYPe are set respectively using the `IGSET` parameters `FAIS`, `FASI`, `MTYP` and `LTYPE`.

Example

```
PAW > IGSET FAIS 3       | Fill area are hatched
PAW > IGSET FASI 244     |   with the style index
PAW > IGSET MTYP 25      | Marker type is an empty square
PAW > IGSET LTYP 15      | Line type is dotted
```

HIGZ provides some portable fill area styles index coded using three digits `ijk` as follows:

`i:` Distance between each hatch in mm

`j:` Angle between 90 and 180 degrees

`k:` Angle between 0 and 90 degrees

These numbers are coded according to table 7.4 and examples are shown in figure 7.7.

Example

```
PAW > IGSET FAIS 3       | Fill area interior style is hatched
PAW > IGSET FASI 190     | Hatch type is 190
```

These commands will yield hatching with two sets of lines at 90° and 0° spaced 1 mm apart.

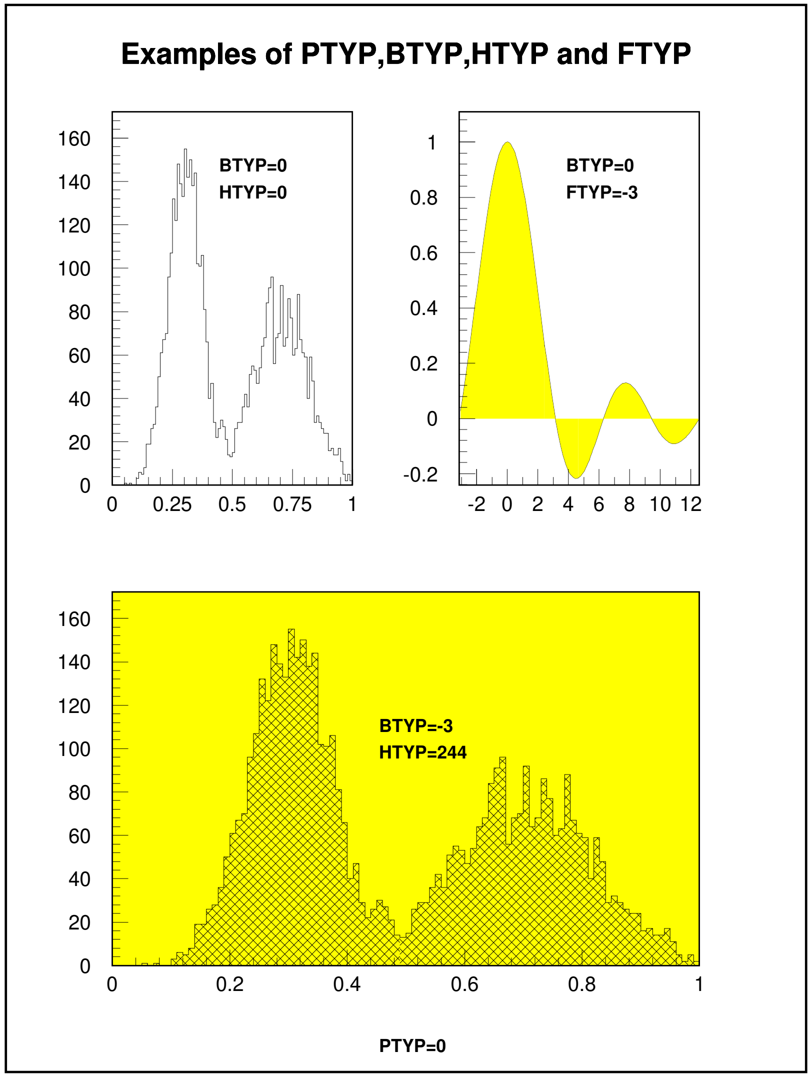


Figure 7.6: Usage of fill area types in HPLOT

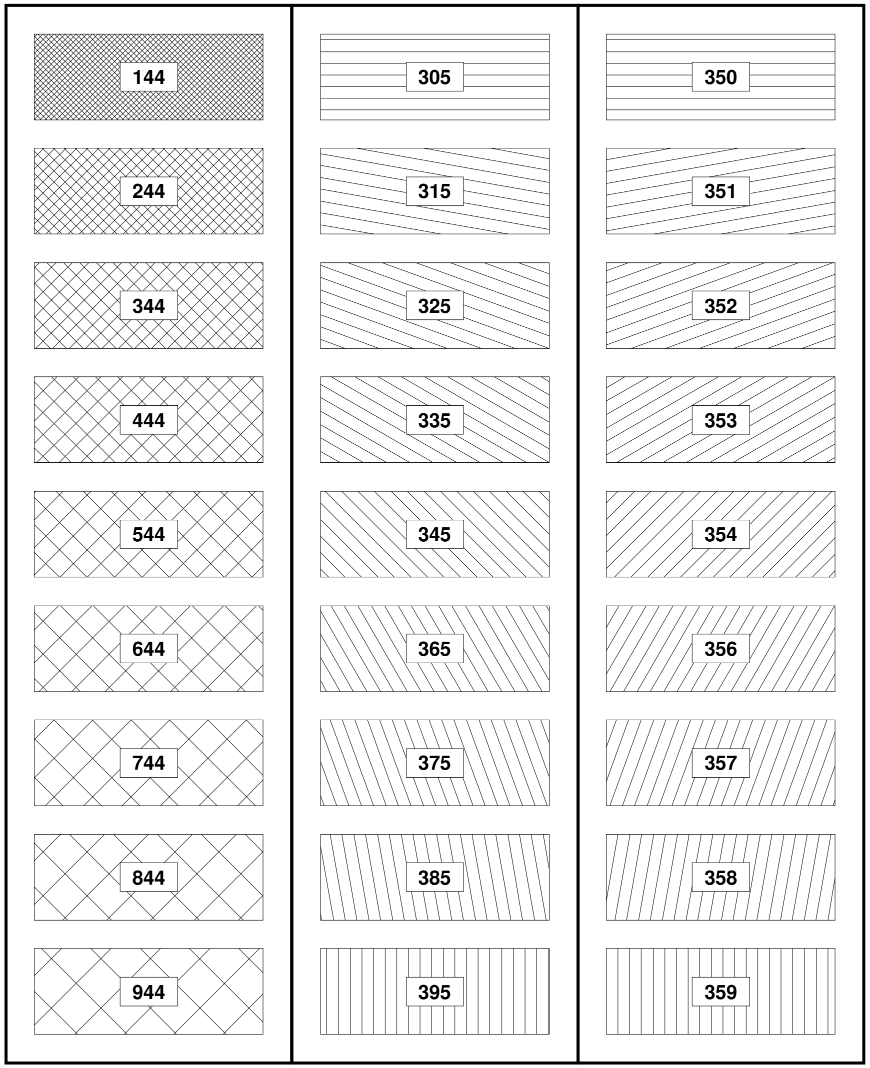


Figure 7.7: HIGZ portable hatch styles

Marker Type	Marker
31	✳
30	☆
29	★
28	✚
27	◇
26	△
25	□
24	○
23	▼
22	▲
21	■
20	●

Figure 7.8: HIGZ portable marker types

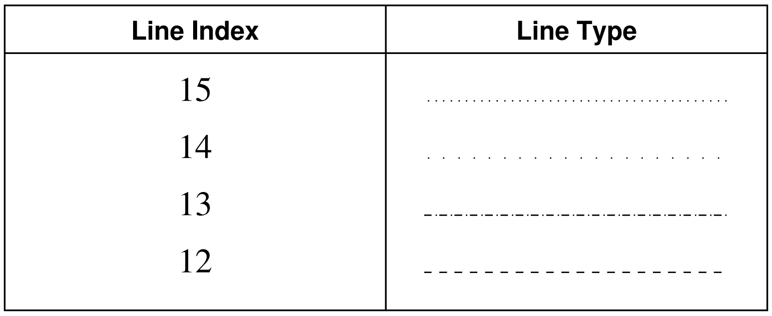


Figure 7.9: HIGZ portable line types

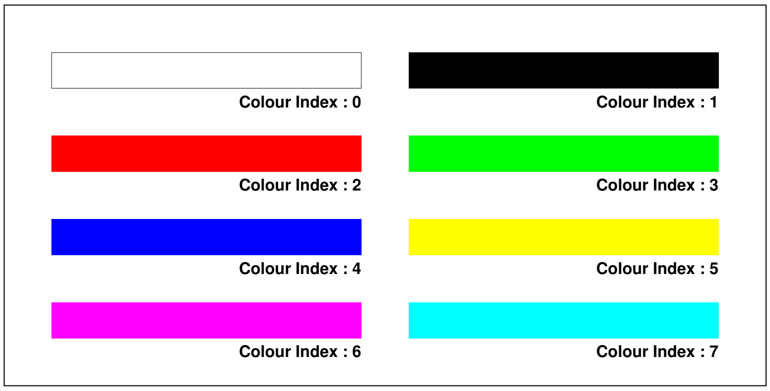


Figure 7.10: PostScript grey level simulation of the basic colours

i	Distance	j	Angle	k	Angle
		0	180°	0	0°
1	0.75mm	1	170°	1	10°
2	1.50mm	2	160°	2	20°
3	2.25mm	3	150°	3	30°
4	3.00mm	4	135°	4	45°
5	3.75mm	5	not drawn	5	not drawn
6	4.50mm	6	120°	6	60°
7	5.25mm	7	110°	7	70°
8	6.00mm	8	100°	8	80°
9	6.75mm	9	90°	9	90°

Table 7.4: Codification for the HIGZ portable fill area interior styles

7.8 Text drawing

In PAW, text output can be produced in two ways:

1. *Automaticaly* with commands like `GRAPH` or `HISTO/PLOT` in which a lot of text is drawn: the axis labels, the histogram title, the global title, the statistics etc. . The attributes (font, colour or size) and the placement of these texts are controled with the command `SET`. In the rest of the chapter, the text produce *automaticaly* will be called *HPLOT text*

2. *Directly* with the commands `ITX` and `TEXT`. The attributes of `ITX` are controlled with the command `IGSET` whereas the attributes of `TEXT` are given with the command parameters.

Text placement

The text placement specify where the text must be drawn. For the *HPLOT text*, the text position is always in centimeters whereas for `ITX` or `TEXT` the current coordinate system is used.

HPLOT text

The possible text placements for *HPLOT text* are described in the following example:

```
PAW > SET XVAL 0.40 | distance between the Y axis and the axis values
PAW > SET YVAL 0.20 | distance between the X axis and the axis values
PAW > SET YLAB 0.80 | distance X axis to labels
PAW > SET XLAB 1.40 | distance Y axis to labels
PAW > SET YGTI 1.50 | Y position of global title
PAW > SET YHTI 1.20 | Y position  of histogram title
PAW > SET YNPG 0.60 | Y position for the page number
PAW > HISTO/PLOT 10 | the histogram 10 is drawn with previous settings
```

See figure 7.3 for more details.

ITX

In the command `ITX` the text position is defined with two mandatory parameters (`X` and `Y`):

```
PAW > SELNT 1         | cm coordinates
PAW > ITX 5 5 'Hello' | 'Hello' is drawn at the position (5,5)
```

TEXT

In the command TEXT the text position is defined with two mandatory parameters (X and Y):

```
PAW > SELNT 1            | cm coordinates
PAW > TEXT 5 5 'Hello' 1 | 'Hello' is drawn at the position (5,5)
```

Text size

For all the texts drawn with PAW commands, the text size is always specified in centimeters.

HPLOT text

The possible text sizes for *HPLOT text* are described in the following example:

```
PAW > SET ASIZ 0.28 | axis label size
PAW > SET CSIZ 0.28 | comment size
PAW > SET GSIZ 0.28 | global title size
PAW > SET KSIZ 0.28 | Hershey character size
PAW > SET 2SIZ 0.28 | scatter plot and table character. size
PAW > SET TSIZ 0.28 | histogram title size
PAW > SET VSIZ 0.28 | axis values size
PAW > HISTO/PLOT 10 | the histogram 10 is drawn with previous settings
```

See figure 7.3 for more details.

ITX

The text character heigh attribute for use by future invocations of ITX is set using the CHHE parameter as follows:

```
PAW > IGSET CHHE 1    | set the character heigh to 1 cm.
PAW > ITX 5 5 'Hello' | the size of 'Hello' is 1 cm.
```

TEXT

In the command TEXT the text size is a mandatory parameter (SIZE).

```
PAW > TEXT 5 5 'Hello' 1 | the size of 'Hello' is 1 cm.
```

Text orientation

The text orientation is an angle (in degrees) between the X axis and the text axis. By default this angle is equal to 0.

HPLOT text

Text orientation cannot be changed with some SET parameters for the *HPLOT text*. It is always automaticaly computed. For example in the command ATITLE, which draws the axis titles, the title on the Y axis is automaticaly drawn with an angle of 90 degrees.

ITX

The text orientation attribute for use by future invocations of ITX is set using the TANG parameter as follows:

```
PAW > IGSET TANG 90   | set the text angle to 90 degrees.
PAW > ITX 5 5 'Hello' | 'Hello' is drawn with an angle of 90 degrees.
```

TEXT

In the command TEXT the text orientation is an optional parameter (ANGLE).

```
PAW > TEXT 5 5 'Hello' ! 90 | 'Hello' is drawn with an angle of 90 degrees
```

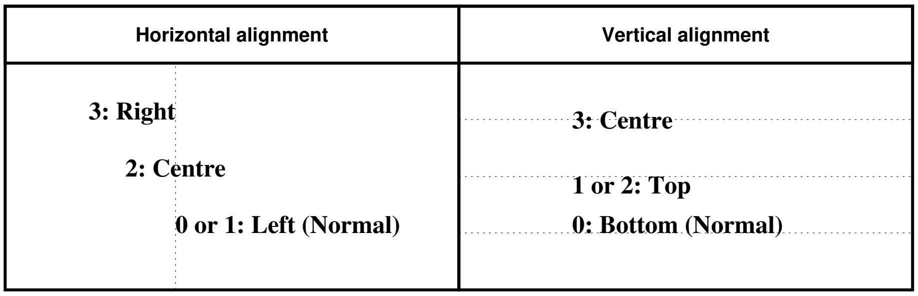


Figure 7.11: Text alignment

Text alignment

The text alignment controls the placement of the character string with respect to the specified text position.

HPLOT text

Text alignment cannot be changed for the *HPLOT text*. It is automaticaly computed.

ITX

The text alignment attributes for use by future invocations of `ITX` are set using the `TXAL` parameter as follows:

```
PAW > IGSET TXAL (10*(horizontal alignment) + (vertical alignment))
```

The horizontal and vertical alignments parameters must be in the range `0-3`. The horizontal alignment specifies which end of the string (or its geometric center) is aligned with the specified point given in `ITX`. The vertical alignment controls whether the top of tall characters (or the bottom of capital letters) line up with the specified point (see figure 7.11).

ITXALH horizontal alignment

- 0 normal (usually same as 1)
- 1 left end of string at specified point
- 2 center of string at specified point
- 3 right end of string at specified point

ITXALH vertical alignment

- 0 normal
- 1 top of tallest chars plus any built in spacing
- 2 top of tallest chars
- 3 halfway between 2 and 4

```
PAW > IGSET TXAL 23   | The horizontal and vertical alignments are centered
PAW > ITX 5 5 'Hello' | 'Hello' is drawn center adjusted
```

TEXT

In the command `TEXT` the text alignment is an optional parameter (`CHOPT`). Only the horizontal alignment can be changed among three possible values: Left, Center or Right.

```
PAW > TEXT 5 5 'Hello' 1 ! L | 'Hello' is drawn left adjusted (default)
PAW > TEXT 5 5 'Hello' 1 ! C | 'Hello' is drawn center adjusted
PAW > TEXT 5 5 'Hello' 1 ! R | 'Hello' is drawn right adjusted
```

Text colour

The text colour is define via a colour index in the colour table.

HPLOT text

```
PAW > SET XCOL 2    | X axis color
PAW > SET YCOL 3    | Y axis color
PAW > HISTO/PLOT 10 | the histogram 10 is drawn with previous settings
```

ITX

The text colour attribute for use by future invocations of `ITX` is set using the `TXCI` parameter as follows:

```
PAW > IGSET TXCI 3    | set the text colour to green.
PAW > ITX 5 5 'Hello' | 'Hello' is drawn in green.
```

TEXT

The text colour attribute for use by future invocations of `TEXT` is set using the `TXCI` parameter as follows:

```
PAW > IGSET TXCI 2      | set the text colour to red.
PAW > TEXT 5 5 'Hello' ! | 'Hello' is drawn in red.
```

Text font and precision

Text font selects the desired character font e.g. a roman font, a sans-serif font, etc. Text precision specifies how closely the graphics package implementation must follow the current size and orientation attributes. String (0) precision is most liberal (hardware), stroke (2) precision is most strict. Character precision is in the middle (1). The value of text font is dependent upon the basic graphics package used. However, font number 0, with precision 2 is always available, independently from the basic graphics package used.Hardware characters are available with all the basic graphics packages. With X11, a large variety of font is available. They are the same as the PostScript fonts (see figure 7.15).

HPLOT text

```
PAW > SET CFON -60  | comment font is Helvetica Bold
PAW > SET GFON -20  | global title font is Times Bold
PAW > SET LFON -60  | axis labels font is Helvetica Bold
PAW > SET TFON -20  | general comments is Times Bold
PAW > SET VFON -60  | axis values font is Helvetica Bold
PAW > HISTO/PLOT 10 | the histogram 10 is drawn with previous settings
```

Note that `SET *FON ffp` set all the *HPLOT text* font to the same value `ffp`.

ITX

Text font and precision attributes for use by later invocations of `ITX` are set with `TXFP` as follows:

```
PAW > IGSET TXFP (10*(Text font) + (text precision))
```

TEXT

This command draws a software character text, independently from the basic graphics package used by HIGZ. It can produce over 300 different graphic signs. The way in which software characters are defined is via a string of valid characters, intermixed by other characters, acting as “escape” characters (e.g. a change of alphabet, upper or lower case). The string is interpreted by `TEXT` and the resulting characters are defined according to the figure 7.12, which shows the list of available software characters. This command allows the user to mix different types of characters (roman, greek, special, upper and lower case, sub and superscript). There are a total of 10 control characters.

List of escape characters and their meaning			
$<$	go to lower case	$>$	go to upper case (default)
[	go to greek (Roman = default)	]	end of greek
”	go to special symbols	#	end of special symbols
$\uparrow$	go to superscript	?	go to subscript
!	go to normal level of script	&	backspace one character
$	termination character (optional)		

Note that characters can be also entered directly in lower case or upper case instead of using the control characters `<` and `>`.

The boldface characters may be simulated by setting the attributes 'PASS' and 'CSHI' with IGSET. The meaning of these attributes is the following: Every stroke used to display the character is repeated PASS times, at a distance (in percentage of the character height) given by CSHI.

PostScript text fonts

PostScript files the text can be generated with PostScript fonts. The figure 7.15 shows all the PostScript fonts available on most PostScript printers. Note that the fonts -15 to -24 are the same than -1 to -14, but they are drawn in hollow mode.

The correspondence between ASCII and ZapfDingbats font is given on figures 7.16 and 7.17. TEXT control characters are taken into account. In addition the character $\sim$ switches to the ZapfDingbats character set.

List of escape characters and their meaning			
$<$	go to lower case (optional)	$>$	go to upper case (optional)
[	go to greek (Roman = default)	]	end of greek
”	go to special symbols	#	end of special symbols
$\sim$	go to ZapfDingbats	#	end of ZapfDingbats
$\uparrow$	go to superscript	?	go to subscript
!	go to normal level of script	&	backspace one character
$	termination character (optional)		

The PostScript fonts can be used with precision 0 or precision 1. On the screen, a PostScript font used with precision 1 appears like the TEXT characters, with precision 0 its appears as hardware character (X11 fonts). In both cases the PostScript file is the same.

Note that characters can also be entered directly in lower or upper case instead of using the escape characters `<` and `>`. Examples of PostScript text and math are shown in Figures 7.13 and 7.14.

Upper Roman	Lower Roman	Upper Greek	Lower Greek	Upper Special	Lower Special
A	a	A	α	±	±
B	b	B	β	|	|
C	c	H	η	✚	✡
D	d	Δ	δ	$	$
E	e	E	ε	!	!
F	f	Φ	φ	#	#
G	g	Γ	γ	>	>
H	h	X	χ	?	?
I	i	I	ι	∫	∫
J	j	I	ι	:	:
K	k	K	κ	;	;
L	l	Λ	λ	<	<
M	m	M	μ	[	[
N	n	N	ν	]	]
O	o	O	ο	≥	≤
P	p	Π	π	⌇	⌇
Q	q	Θ	ϑ	⌇	⌇
R	r	P	ρ	√	√
S	s	Σ	σ	♤	🔔
T	t	T	τ	♡	🌴
U	u	ϒ	υ	◇	🌲
V	v	X	χ	♧	♧
W	w	Ω	ω	&	&
X	x	Ξ	ξ	×	×
Y	y	Ψ	ψ	%	∝
Z	z	Z	ζ	∞	∞
0	0	0	0	◍	○
1	1	1	1	▥	□
2	2	2	2	△	△
3	3	3	3	▽	◊
4	4	4	4	☆	☆
5	5	5	5	→	→
6	6	6	6	↑	↑
7	7	7	7	←	←
8	8	8	8	↓	↓
9	9	9	9	,	,
.	.	.	.	◇	□
,	,	,	,	,	,
+	+	+	+	+	+
−	-	−	-	_	_
*	*	*	*	✳	✳
/	/	/	/	/	/
=	=	=	=	=	=
(	(	(	(	(	(
)	)	)	)	)	)

Figure 7.12: Characters available in `IGTEXT`

```
PAW > IGSET LWID 6
PAW > BOX 0 16 0 5
PAW > IGSET CHHE 0.5
PAW > IGSET TXAL 3
PAW > IGSET TXFP -130
PAW > ITX 3 4 'K\355nstler in den gr\345\373ten st\311dten
PAW > ITX 3 3 '\253\265 l''\372uvre on conna\333t l''artisan\273
PAW > ITX 3 2 '\(proverbe fran\321ais\
PAW > ITX 3 1 '\252\241Ma\337ana\41 \322ag&\306!das&\313!\272, dit l''\3231\325ve.
```

Künstler in den größten Städten
«À l’œuvre on connaît l’artisan»
(proverbe français).
“¡Mañana! Çağdaş”, dit l’élève.

Figure 7.13: Example of PostScript text (result of input above)

```
PAW > IGSET LWID 6
PAW > BOX 0 16 0 5
PAW > IGSET CHHE 0.5
PAW > IGSET TXAL 23
PAW > IGSET TXFP -130
PAW > ITX 8 4 'e^+!e^-! "5# Z^o! "5# ll&^-!, qq&^\261!'
PAW > ITX 8 3 '| a&^[\256]! \267 b&^[\256]! | = [\345] a^i?&jk!+b^kj?&i'
PAW > ITX 8 2 'i ("d#?[m!y]&^\261![g^m]! + m [y]&^\261! ) = 0" r# (~r# + m^2!) [y] = 0'
PAW > ITX 8 1 'L?em! = e J^[m]?&em! A?[m]! , J^[m]?&em!=l&^\261![ g?m]!l , M^j?&i! = [\345&?a]! A?[a! t^a]j?&i! '
```

$$e^+e^- \rightarrow Z^o \rightarrow l\bar{l},\ q\bar{q}$$
$$|\ \vec{a} \bullet \vec{b}\ | = \sum a^i_{jk} + b^{kj}_i$$
$$i\ (\partial_\mu \bar{\psi} \gamma^\mu + m\ \bar{\psi}\ ) = 0 \Leftrightarrow (\square + m^2)\ \psi = 0$$
$$L_{em} = e\ J^\mu_{em}\ A_\mu\ ,\ J^\mu_{em} = \bar{l}\ \gamma_\mu l\ ,\ M^j_i = \sum_\alpha A_\alpha\ \tau^{\alpha j}_i$$

Figure 7.14: Example of PostScript text and maths (result of input above)

Font/Prec	PostScript Font Style	
-1/0	*ABCDEFghijlk0123456789*	Times-Italic
-2/0	**ABCDEFghijlk0123456789**	Times-Bold
-3/0	***ABCDEFghijlk0123456789***	Times-BoldItalic
-4/0	ABCDEFghijlk0123456789	Helvetica
-5/0	*ABCDEFghijlk0123456789*	Helvetica-Oblique
-6/0	**ABCDEFghijlk0123456789**	Helvetica-Bold
-7/0	***ABCDEFghijlk0123456789***	Helvetica-BoldOblique
-8/0	ABCDEFghijlk0123456789	Courier
-9/0	*ABCDEFghijlk0123456789*	Courier-Oblique
-10/0	**ABCDEFghijlk0123456789**	Courier-Bold
-11/0	***ABCDEFghijlk0123456789***	Courier-BoldOblique
-12/0	ΑΒΗΔΕΦγχιλκ0123456789	Symbol
-13/0	ABCDEFghijlk0123456789	Times-Roman
-14/0	✡✢✣✤✥✦❇❈❉❊●❋✐✑✒✓✔✕✖✗✘✙	ZapfDingbats
-15/0	*ABCDEFghijlk0123456789*	Times-Italic
-16/0	**ABCDEFghijlk0123456789**	Times-Bold
-17/0	***ABCDEFghijlk0123456789***	Times-BoldItalic
-18/0	ABCDEFghijlk0123456789	Helvetica
-19/0	*ABCDEFghijlk0123456789*	Helvetica-Oblique
-20/0	**ABCDEFghijlk0123456789**	Helvetica-Bold
-21/0	***ABCDEFghijlk0123456789***	Helvetica-BoldOblique
-22/0	ΑΒΧΔΕΦγηιφλκ0123456789	Symbol
-23/0	ABCDEFghijlk0123456789	Times-Roman
-24/0	✡✢✣✤✥✦❇❈❉❊○❋✐✑✒✓✔✕✖✗✘✙	ZapfDingbats

Figure 7.15: PostScript text fonts.

Input	*Upper Roman*	*Upper Greek*	*Upper Special*	*Upper Zapf*
A	A	Α	±	✡
B	B	Β	|	✢
C	C	Η	∃	✣
D	D	Δ	∀	✤
E	E	Ε	!	✥
F	F	Φ	#	✦
G	G	Γ	>	✧
H	H	Χ	?	★
I	I	Ι	∫	✩
J	J	Ι	:	✪
K	K	Κ	;	✫
L	L	Λ	<	✬
M	M	Μ	[	✭
N	N	Ν	]	✮
O	O	Ο	≥	✯
P	P	Π	{	✰
Q	Q	Θ	}	✱
R	R	Ρ	√	✲
S	S	Σ	♠	✳
T	T	Τ	♥	✴
U	U	Υ	♦	✵
V	V	Χ	♣	✶
W	W	Ω	≤	✷
X	X	Ξ	×	✸
Y	Y	Ψ	%	✹
Z	Z	Ζ	∞	✺
0	0	0	⊗	✐
1	1	1	⊕	✑
2	2	2	∅	✒
3	3	3	◊	✓
4	4	4	•	✔
5	5	5	→	✕
6	6	6	↑	✖
7	7	7	←	✗
8	8	8	↓	✘
9	9	9	↔	✙
.	.	.	.	✎
,	,	,	,	✌
+	+	+	+	☞
-	-	−	∠	✍
***	*	*	〈	☛
/	/	/	÷	✏
=	=	=	≠	✝
(	(	(	≡	✈
)	)	)	″	✉
{	{	{	{	❛
}	}	}	}	❝

Input	*Lower Roman*	*Lower Greek*	*Lower Special*	*Lower Zapf*
a	a	α	≈	❁
b	b	β	≅	❂
c	c	η	⊥	❃
d	d	δ	∂	❄
e	e	ε	ƒ	❅
f	f	φ	∩	❆
g	g	γ	∪	❇
h	h	χ	⊃	❈
i	i	ι	⊇	❉
j	j	ι	⊄	❊
k	k	κ	⊂	❋
l	l	λ	⊆	●
m	m	μ	∈	❍
n	n	ν	∉	■
o	o	ο	∇	❏
p	p	π	∧	❐
q	q	θ	∨	❑
r	r	ρ	⇔	❒
s	s	σ	⇐	▲
t	t	τ	⇑	▼
u	u	υ	⇒	◆
v	v	χ	⇓	❖
w	w	ω	&	◗
x	x	ξ	ϖ	❘
y	y	ψ	~	❙
z	z	ζ	ℵ	❚
:	:	:	:	✚
;	;	;	;	✛
**	\\	∴	∴	✼
_	‾	‾	‾	✿
|	|	|	|	❜
%	%	%	%	☎
\47	’	∋	∋	✇
\74	<	<	<	✜
\76	>	>	>	✞
\133	[	[	[	✻
\135	]	]	]	✽
\42	"	∀	∀	✂
\43	#	#	#	✃
\136	^	⊥	⊥	✾
\77	?	?	?	✟
\41	!	!	!	✁
\46	&	&	&	✆
\44	$	∃	∃	✄
\176	~	~	~	❞

Figure 7.16: PostScript characters (1).

Input	Upper Roman	Upper Greek	Upper Special	Upper Zapf
\241	¡	ϒ	ϒ	❡
\242	¢	′	′	❢
\243	£	≤	≤	❣
\244	⁄	⁄	⁄	❤
\245	¥	∞	∞	❥
\246	ƒ	ƒ	ƒ	❦
\247	§	♣	♣	❧
\250	¤	♦	♦	♣
\251	'	♥	♥	♦
\252	“	♠	♠	♥
\253	«	↔	↔	♠
\254	‹	←	←	①
\255	›	↑	↑	②
\256	ﬁ	→	→	③
\257	ﬂ	↓	↓	④
\260	à	°	°	⑤
\261	–	±	±	⑥
\262	†	″	″	⑦
\263	‡	≥	≥	⑧
\264	·	×	×	⑨
\265	À	∝	∝	⑩
\266	¶	∂	∂	❶
\267	•	•	•	❷
\270	‚	÷	÷	❸
\271	„	≠	≠	❹
\272	”	≡	≡	❺
\273	»	≈	≈	❻
\274	…	…	…	❼
\275	‰	⏐	⏐	❽
\276	â	⎯	⎯	❾
\277	¿	↵	↵	❿
\300	Â	ℵ	ℵ	➀
\301	`	ℑ	ℑ	➁
\302	´	ℜ	ℜ	➂
\303	ˆ	℘	℘	➃
\304	˜	⊗	⊗	➄
\305	¯	⊕	⊕	➅
\306	˘	∅	∅	➆
\307	˙	∩	∩	➇
\310	¨	∪	∪	➈
\311	ä	⊃	⊃	➉
\312	˚	⊇	⊇	➊
\313	¸	⊄	⊄	➋
\314	Ä	⊂	⊂	➌
\315	˝	⊆	⊆	➍
\316	˛	∈	∈	➎
\317	ˇ	∉	∉	➏

Input	Lower Roman	Lower Greek	Lower Special	Lower Zapf
\321	ç	∇	∇	➑
\322	Ç	®	®	➒
\323	é	©	©	➓
\324	É	™	™	➔
\325	è	∏	∏	→
\326	È	√	√	↔
\327	ê	⋅	⋅	↕
\330	Ê	¬	¬	➘
\331	ë	∧	∧	➙
\332	Ë	∨	∨	➚
\333	î	⇔	⇔	➛
\334	Î	⇐	⇐	➜
\335	ï	⇑	⇑	➝
\336	Ï	⇒	⇒	➞
\337	ñ	⇓	⇓	➟
\340	Ñ	◊	◊	➠
\341	Æ	〈	〈	➡
\342	ô	®	®	➢
\343	ª	©	©	➣
\344	Ô	™	™	➤
\345	ö	∑	∑	➥
\346	Ö	⎛	⎛	➦
\347	û	⎜	⎜	➧
\350	Ł	⎝	⎝	➨
\351	Ø	⎡	⎡	➩
\352	Œ	⎢	⎢	➪
\353	º	⎣	⎣	➫
\354	Û	⎧	⎧	➬
\355	ü	⎨	⎨	➭
\356	Ü	⎩	⎩	➮
\357	å	⎪	⎪	➯
\360				
\361	æ	〉	〉	➱
\362	Ä	∫	∫	➲
\363	ÿ	⌠	⌠	➳
\364	Ÿ	⎮	⎮	➴
\365	ı	⌡	⌡	➵
\366	á	⎞	⎞	➶
\367	Á	⎟	⎟	➷
\370	ł	⎠	⎠	➸
\371	ø	⎤	⎤	➹
\372	œ	⎥	⎥	➺
\373	ß	⎦	⎦	➻
\374	ù	⎫	⎫	➼
\375	Ù	⎬	⎬	➽
\376		⎭	⎭	➾
\377				

Figure 7.17: PostScript characters (2).

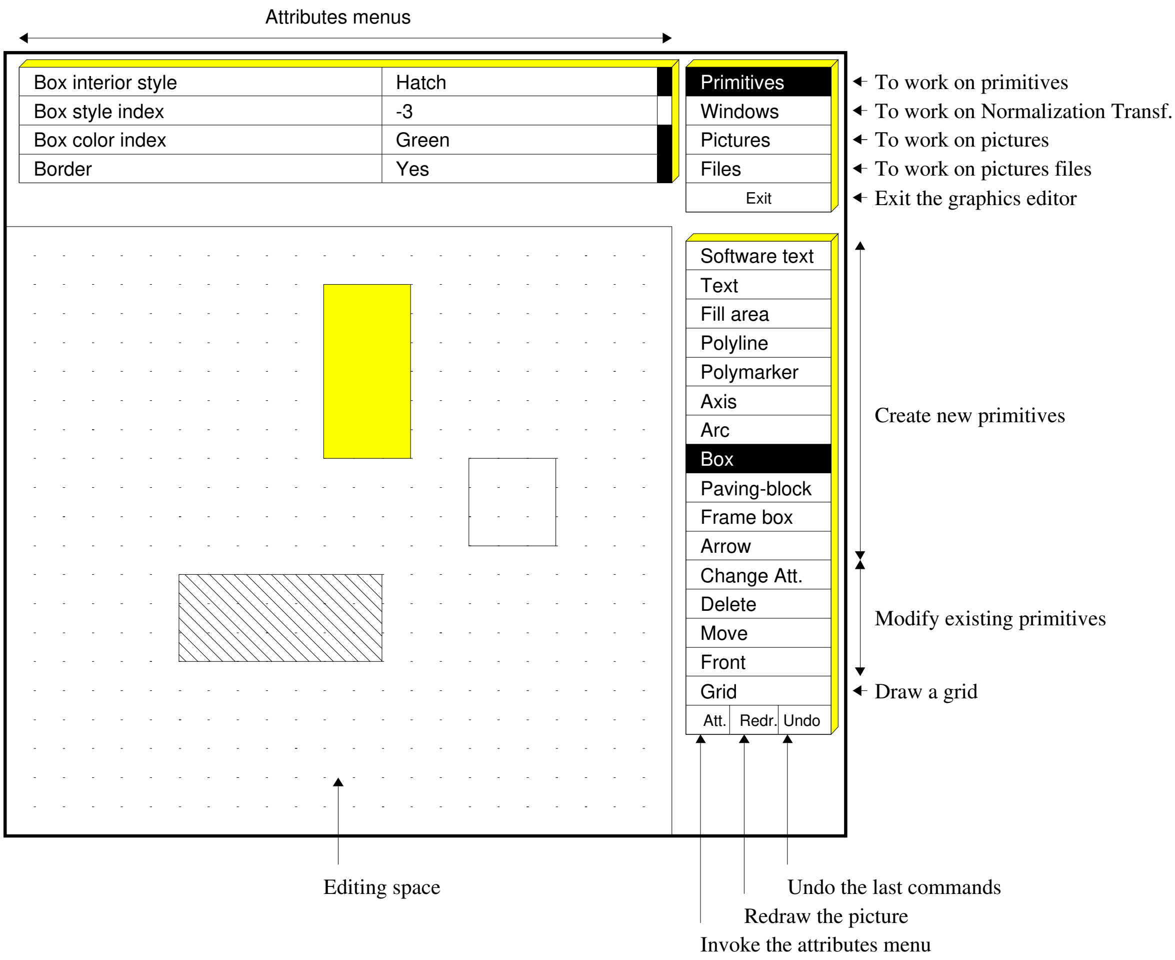


Figure 7.18: The HIGZ graphics editor

7.9 The HIGZ graphics editor

The HIGZ pictures in memory can be modified interactively with the HIGZ graphics editor. The command `PICT/MODIFY` invokes the HIGZ editor (see figure 7.18 for more details):

```
PAW > PICT/MODIFY PNAME
```

`PNAME` can be the complete name, the picture number in memory or ' '.

Chapter 8: Distributed PAW

8.1 Access to remote files from a PAW session

When running PAW, it is often necessary to access files (e.g. HBOOK files) which reside on a different computer. The ZFTP program described above can be used if a very frequent access to the file is required. A more convenient mechanism is the possibility to access the files directly. On many systems, one may now use `NFS` [12] for this purpose. Under some circumstances, for example if the HBOOK file is not in exchange mode and it is to be accessed from a computer running a different operating system, an alternate approach is required. To fill this gap the PAW server is provided. This works using a conventional Client/Server model. The client (PAW) typically runs on a workstation. When the PAW command RLOGIN is invoked, a PAW server is automatically started on the remote machine, normally a mainframe or data server.

Once the `RLOGIN REMOTE` command has been executed, the PAW Current Directory is set to `//REMOTE`. The PAW client can now instruct the PAW server to attach a file using the `RSHELL` command (e.g. `rshell file pawtest.dat`). If an histogram with HBOOK ID=10 is on the remote file, than the PAW command `Histo/Plot 10` will plot this histogram on the local workstation. The histogram resides on `//PAWC` like other histograms coming from local files.

The `RSHELL` command may be used to communicate with the PAW server. The expression typed following `RSHELL` is passed to the server. The current implementation of the PAW server recognizes the commands:

`rshell file filename`	Server connects filename
`rshell cdir //lun11`	Server changes current directory
`rshell ld`	Server lists current directory
`rshell ld //`	Server lists all connected files
`rshell message`	Server pass message to operating system

Access to remote files from a workstation

```
PAW > rlogin CERNVM                       | connect to CERNVM
PAW > rshell file HRZTEST.DAT             | PAW server connects HRZTEST DAT A to //LUN11
PAW > histo/plot 10                       | plot histogram 10 from CERNVM
PAW > histo/fit 20 G                      | fit histo 20 with a gaussian and plot it
PAW > rlogin VXCRNA                       | connect to VXCRNA
PAW > rshell file DISK$DL:[PAW]HEXAM.DAT;3 | PAW server on VXCRNA connects file to //LUN11
PAW > histo/plot 110                      | plot histogram 110 from VXCRNA
PAW > rshell file HRZTEST.DAT             | PAW server on VXCRNA connects file to //LUN12
PAW > histo/plot 110 s                    | plot histogram 110 from HRZTEST.DAT
                                          | on VXCRNA on the existing picture
PAW > rshell ld //                        | list all files connected on VXCRNA
PAW > cdir //CERNVM                       | Change current PAW directory to CERNVM
PAW > histo/plot 110                      | plot histogram 110 from CERNVM
PAW > histo/plot //VXCRNA/110             | plot histogram 110 from VXCRNA
PAW > cdir //PAWC                         | current directory to local memory
PAW > histo/list                          | list all histograms in //PAWC
PAW > Histo/delete 0                      | delete all histograms in memory
PAW > hrin //VXCRNA/0                     | read all histograms from VXCRNA
                                          | file HRZTEST.DAT to //PAWC
PAW > cdir //CERNVM                       | change directory to CERNVM
PAW > rshell file NEW.DAT.D 1024 N        | creates a new file on the D disk
PAW > hrout 0                             | write all histograms from //PAWC
                                          | to CERNVM file NEW DAT D
```

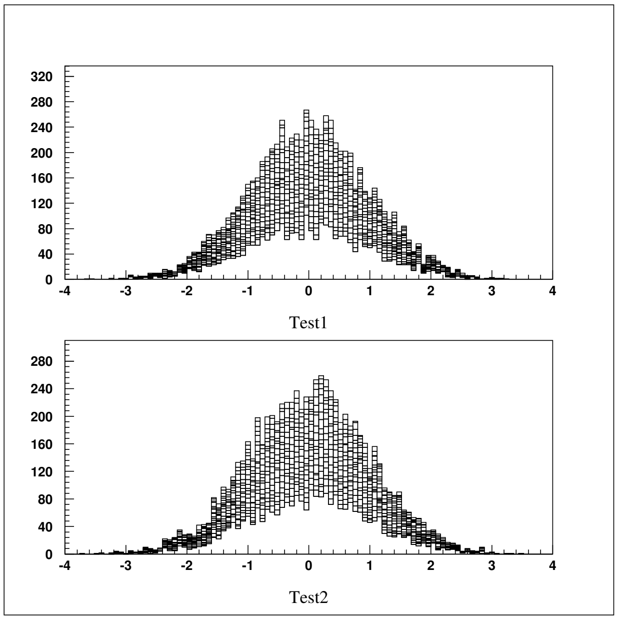
8.2 Using PAW as a presenter on VMS systems (global section)

In addition to the facilities described in the previous section, the standard version of PAW may be used as an online presenter on VMS systems using the mechanism of global sections. It is possible for two processes to reference the same histograms using **global sections**. For example, the first process may be a **histogram producer** (e.g. a monitoring task) and the second process **PAW**. As the histograms are being gradually filled by the first task, PAW can view them, and even reset them. To use the global sections, it is also necessary to "page align" the common which is in the global section. This is achieved in the "link step" when making the process (see example). The relevant statements are `SYS$INPUT/OPTIONS` to tell the linker that some options follow the link statement, and `PSECT=PAWC,PAGE` which is the option to page align the `/PAWC/` common.

```
      PROGRAM PRODUCE
      PARAMETER MAXPAGES=100
      COMMON/PAWC/IPAWC(128*MAXPAGES)
      CHARACTER*8 GNAME
      INTEGER*4 HCREATEG
*
      GNAME='GTEST'
      WAIT_TIME=1.
      NUMEVT=1000
*...............           Create Global section
      NPAGES=HCREATEG(GNAME,IPAWC,128*MAXPAGES)
      IF(NPAGES.GT.0) THEN
         PRINT 1000,GNAME
 1000    FORMAT(' Global Section: ',A,' created')
      ELSE
         IERROR=-NPAGES
         PRINT 2000,IERROR
 2000    FORMAT(' Global Section Error', I6)
         GO TO 99
      ENDIF
      CALL HLIMIT(128*NPAGES)
*...............           Book histos.
      CALL HBOOK1(10,'Test1$',50,-4.,4.,0.)
      CALL HBOOK1(20,'Test2$',50,-4.,4.,0.)
*...............           Fill histos.
      DO 20 I=1,NUMEVT
         DO 10 J=1,100
            CALL RANNOR(A,B)
            CALL HFILL(10,A,0.,1.)
            CALL HFILL(20,B,0.,1.)
 10      CONTINUE
        CALL LIB$WAIT(WAIT_TIME)
 20   CONTINUE
*
 99   STOP
      END

$ fort produce
$ link produce,SYS$INPUT/OPTIONS,-
cern$library:packlib/lib,kernlib/lib
PSECT=PAWC,PAGE
```

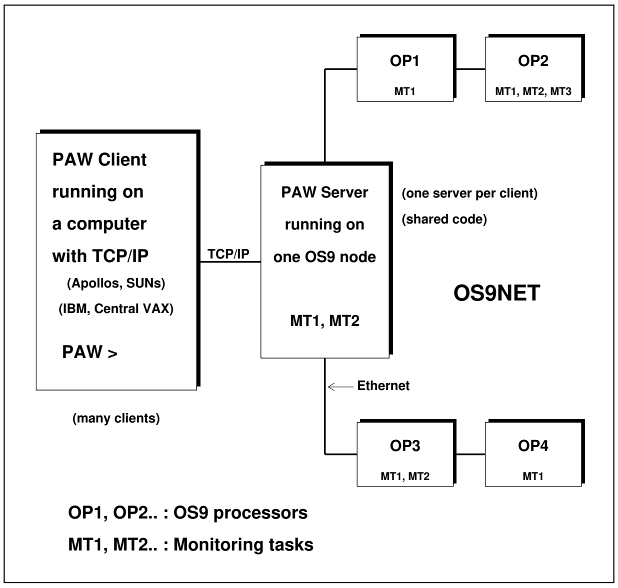
```
PAW > edit produce
   macro produce ntimes=100
     nt=[ntimes]
     zone 1 2
     histo/plot 10 K
     histo/plot 20 K
   loop:
       histo/plot 10 U
       histo/plot 20 U
       wait ' ' 1
       nt=[nt] -1
       if nt>0 goto loop
   return
PAW > global GTEST
PAW > exec produce ntimes=20
```



8.3 Using PAW as a presenter on OS9 systems

The technique described in previous sections may also be used to access HBOOK histograms being filled by a monitoring task on OS9 systems from a standard PAW session running on a machine with the TCP/IP software.

```
      INDIRECT PAWC
      PROGRAM PRODUCE
*
*        Monitoring task MT1 in processor OP2.
*
      PARAMETER NWPAW=10000
      COMMON/PAWC/IPAWC(NWPAW)
*
      CALL HLIMIT(NWPAW)
*
*       Book histos.
*
      CALL HBOOK1(10,'TEST1$',50,-3.,3.,0.)
      CALL HBOOK1(20,'TEST2$',50,-3.,3.,0.)
*
*       Fill histos.
*
      NUMEVT=10000
      DO 20 I=1,NUMEVT
         DO 10 J=1,100
            CALL RANNOR(A,B)
            CALL HFILL(10,A,0.,1.)
            CALL HFILL(20,B,0.,1.)
 10      CONTINUE
 20   CONTINUE
*
 99   STOP
      END
```



Example of how to access OS9 modules from PAW

```
PAW > rlogin O-OPAL01                    | connect to an OS9 machine
PAW > rshell module OP2/MT1              | PAW server connects to OP2/MT1
                                         | (Processor OP2, Monitoring Task MT1)
PAW > histo/plot 10                      | plot histogram 10
PAW > hrin 0                             | read all histograms into //PAWC
PAW > Histo/File 1 local.dat 1024 N      | create a new file local.dat
                                         | on the client machine
PAW > hrout 0                            | save all histograms from //PAWC
                                         | to the local file
PAW > rshell module OP3/MT2              | PAW server connects to another
                                         | OS9 monitoring task
PAW > Output 56 os9.listing              | Change output file on client
PAW > rshell ldir                        | list all histograms in MT2
                                         | on file os9.listing
PAW > Output -56                         | Change output file to default (unit 6)
                                         | file os9.listing is closed
```

Chapter 9: PAW++: A guided tour

PAW++ is a powerful OSF/Motif based Graphical User Interface to the popular Physics Analysis Workstation PAW. The graphical user interface makes the full and rich command set of PAW available to even the naive user. Simple point and click operations are enough to execute commands that were previously accessable only to expert users. Figure 9.1 compares the functionalities of basic PAW with PAW++.

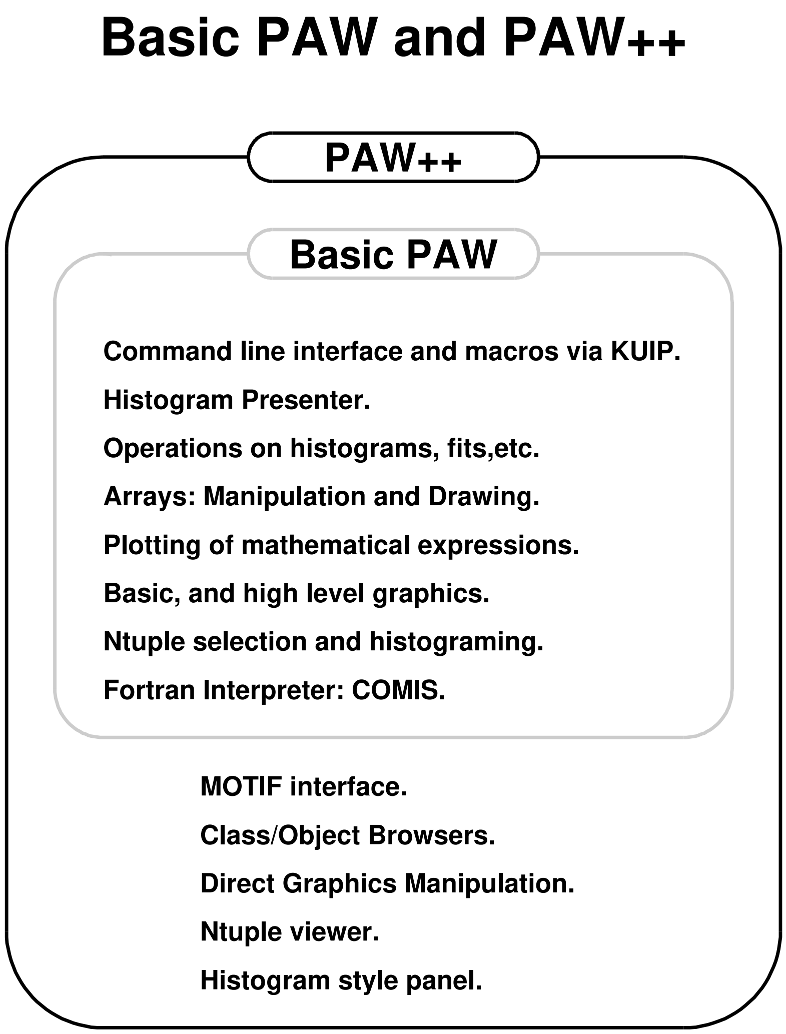


Figure 9.1: PAW and PAW++ compared

At present PAW++ is available on Unix workstations and VAX/VMS.

PAW++ has, in addition to the conventional command line and macro types of interface, the following dialogue modes:

Pull Down menus They are useful to understand the command structure of the PAW system.

Command panels They give a “panel representation” of the commands.

Object Browser This is in many ways similar to the well-known browsers in the PC/MAC utilities or the visual tools on some workstations.

Direct graphics One can click in the graphics area and identify automatically which object has been selected. A pop-up menu appears with a list of possible actions on this object. For example, by clicking with the right mouse button on a histogram, one can make directly a gaussian fit, a smoothing etc. Pop-up menus are available by clicking on the **Graphics Window** to automatically produce PostScript, Encapsulated PostScript, LaTeX files or print the picture on your local printer.

Histogram Style Panel Buttons are available to change histogram attributes, colours, line styles, fonts, and axes representation. 2-D histograms can be rotated interactively. Zooming and rebinning can be performed interactively in real time.

Ntuple Viewer Just click on the Ntuple column name to histogram the column.

The new system is largely self-explanatory. Only a subset of PAW has been converted to this new user interface, but work is currently in progress to offer many new facilities in future releases.

On all system on which CERNLIB is installed, it is enough to type `paw++` to enter the system.

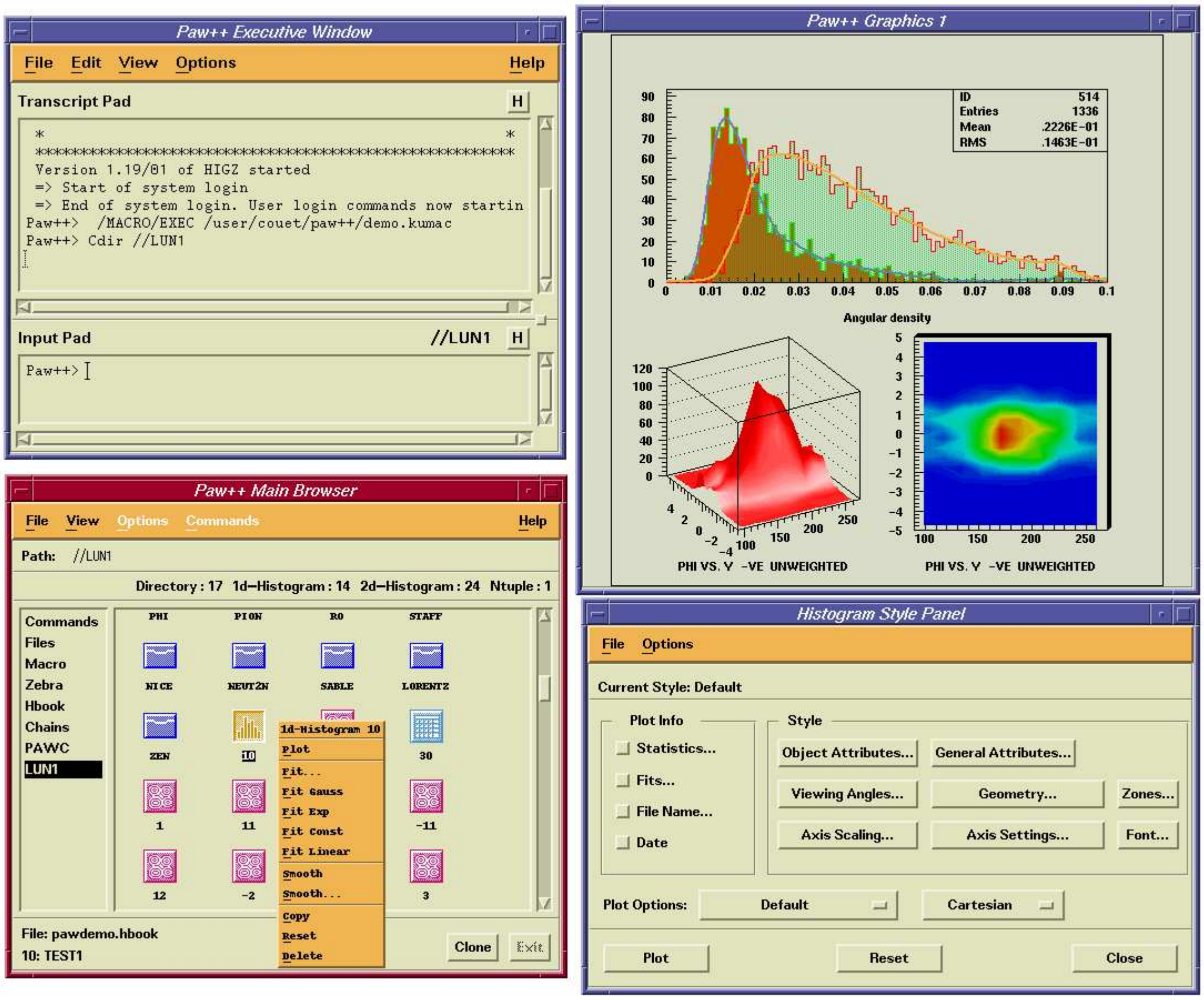
PAW++ starts up with three windows on the screen:

The "PAW++ Executive Window" includes a menu bar, a **Transcript Pad**, a current working directory indicator and an **Input Pad**.

The "PAW++ Graphics 1" window displays the graphics output from HIGZ/X11. Objects, e.g. histograms, displayed in the **Graphics Window** can be manipulated by pointing at them, pressing the right mouse button and selecting an operation from the popup menu. Pointing at the edge of the **Graphics Window** (between displayed object and window border) brings up a general popup menu. Up to 4 additional **Graphics Window** can be opened by selecting "Open New Window" from this menu.

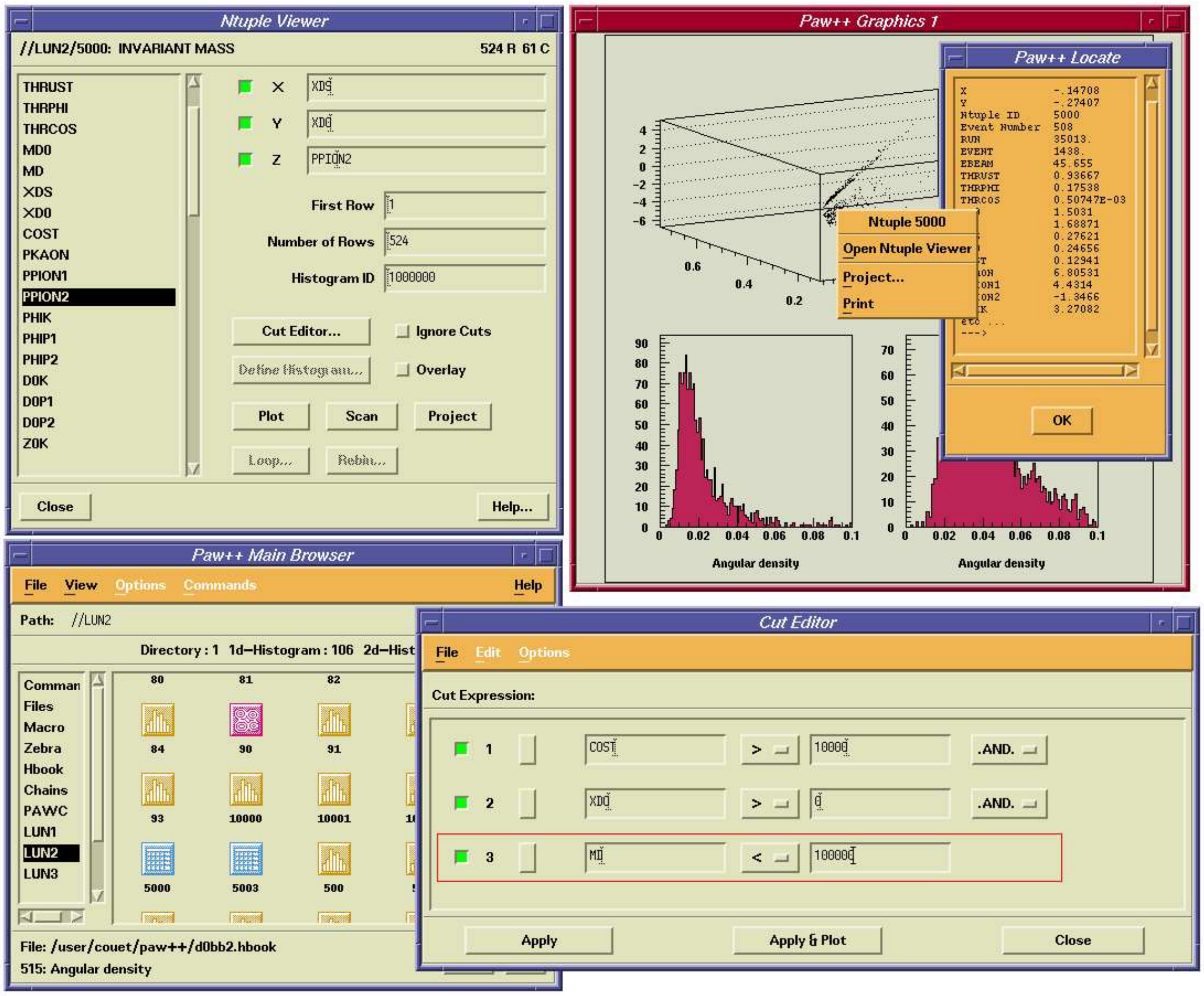
The "PAW++ Main Browser" displays all browsable classes and connected hbook files. Up to 4 additional browsers can be opened via the "View" menu of the "PAW++ **Executive Window**" or via the "Clone" button on the browsers. For more information on the browsers see the "Help" menus.

Figures 9.2 on page 134 and 9.3 on page 135 give a detailed overview of the various windows of PAW++.



- The upper left corner is the PAW++ **Executive Window**, with its **Input Pad** at the bottom and the **Transcript Pad** at the top.
- The PAW++ Browser, where the various entities (pictures, 1-D and 2-D histograms and Ntuples) are all defined with their own symbol, is shown bottom left. A "pop-up" menu has been activated for the chosen 1-D histogram. Several actions like `Plot`, `Smooth`, `Fit` etc... can be performed via this menu.
- The **Graphics Window** is seen top right. A 1-D view of the data points and two 2-D views (a Surface-plot and a colored contour plot) are shown. On the 1-D view, two 1-D histograms are superimposed. The results of a "smoothing" type of fit to the data points is also drawn. Information about the data and the fit can be found in the inserted window.
- The **Histogram Style Panel** at the lower right allows graphics attributes of the histogram to be controlled.

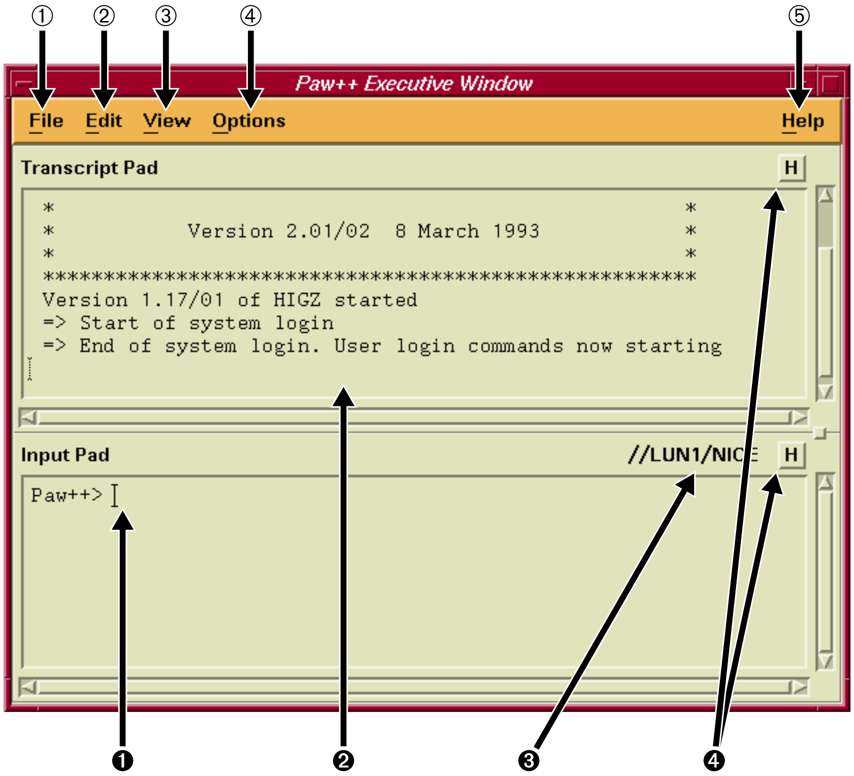
Figure 9.2: PAW++ windows explained (I)



– The upper left corner shows the **Ntuple Viewer**. The left window shows the name of the various variables, characterizing the selected Ntuple. Other windows and press-buttons specify which combinations of the various variables and which events have to be treated (plotted, scanned, . . .).
– The lower left contains the PAW++ Browser, with this time an Ntuple selected. A double on a Ntuple icon open automatically the **Ntuple Viewer** on the active Ntuple.
– The **Graphics Window** is seen top right and shows a 3-D view of the combination of three variables, whose cuts are specified with the **Cut Editor** (see below).
– Direct graphics interactions with Ntuple data are possible. Just by clicking on a point in the **Graphics Window**, the event description is displayed in the **PAW++ Locate** window.
– The **Cut Editor** panel, shown at the lower right, allows various combinations of cuts to be specified and applied.

Figure 9.3: PAW++ windows explained (II)

9.1 The Executive Window



This window allows to type commands on the keyboard like in the normal PAW system. In fact this window is the `kxterm` program provide with the KUIP package.

This terminal emulator combines the best features from the (now defunct) Apollo DM pads (like: **Input Pad** and **Transcript Pad**, automatic file backup of **Transcript Pad**, string search in pads, etc.) and the Korn shell emacs-style command line editing and command line recall mechanism.

Commands are typed in the **Input Pad** ❶ behind the application prompt. Via the toggle buttons H ❹ the **Input Pad** and/or **Transcript Pad** can be placed in hold mode. In hold mode one can paste or type a number of commands into the **Input Pad** and edit them without sending the commands to the application. Releasing the hold button will causes `kxterm` to submit all lines, upto the line containing the cursor, to the application. To submit the lines below the cursor, just move the cursor down. In this way one can still edit the lines just before they are being submitted to the application.

❶ In the **Input Pad** one can type, retrieve and edit command line with the help of a Korn shell emacs-style command line editing mode. See in appendix the complete list of the editing keys.
❷ The **Transcript Pad** ❷ shows the executed commands and command output. When in hold mode ❹ the transcript pad does not scroll to make the new text visible. Mouse operations like "Copy Paste" are allowed in the transcript pad. It is also possible to search a character string (see the menu bar description).
❸ Every time the current directory is changed, the **Current working directory indicator** is updated. The current working directory can be changed by clicking on a item in the **PATH window** of the **Main Browser** or by clicking on a icon directory in the **Main Browser** itself.
❹ Hold buttons.

① Allows manipulation of the **Transcript Pad**.
② Allows character string seach, copy/paste in the **Transcript Pad**.
③ Allows to invoke other panel.
④ Some general settings are available in this menu.
⑤ Online help.

9.1.1 The Executive Window menu bar

In this section, is describe the full functionality of the pull down menu available in the Menu Bar of the **Executive Window**.

File

About Kxterm... Displays version information about Kxterm.

About <Application>... Displays version information about the application Kxterm is servicing.

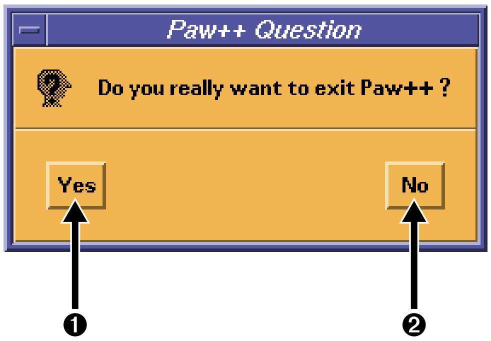
Save Transcript Write the contents of the transcript pad to the current file. If there is no current file a file selection box will appear.

Save Transcript As... Write the contents of the transcript pad to a user-specified file.

Print... Print the contents of the transcript pad (not yet implemented).

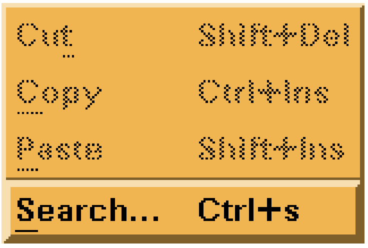
Kill Send a SIGINT signal to the application to cause it to core dump. This is useful when the application is hanging or blocked. Use only in emergency situations.

Exit Exit Kxterm and the application. When this option is selected or when `EXIT` is typed in the **Input Pad**, the following panel is displayed:



❶ The exit is performed.
❷ The exit procedure is canceled.

Edit



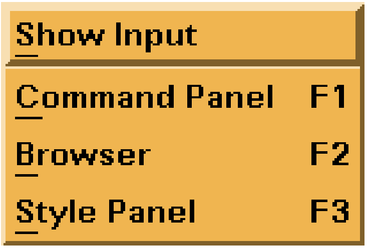
Cut Remove the selected text. The selected text is written to the Cut and Paste buffer. Using the "Paste" function, it can be written to any X11program. In the transcript pad "Cut" defaults to the "Copy" function.

Copy Copy the selected text. The selected text is written to the Cut and Paste buffer. Using the "Paste" function, it can be written to any X11program.

Paste Insert text from the Cut and Paste buffer at the cursor location into the **Input Pad**.

Search... Search for a text string in the transcript pad.

View



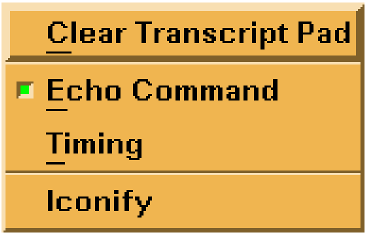
Show Input Show in a window all commands entered via the **Input Pad**.

Command Panel

Browser

Style Panel

Options



Clear Transcript Pad Clear all text off of the top of the transcript pad.

Echo Command Echo executed commands in transcript pad.

Timing Report command execution time (real and CPU time).

Iconify Iconify Kxterm and all windows of the application.

Help

On Kxterm The help you are currently reading.

On Edit Keys Help on the emacs-style edit key sequences.

9.2 The Main Browser

The KUIP/Motif Browser interface is a general tool to display and manipulate a tree structure of objects which are defined either by KUIP itself (commands, files, macros, etc.) or by the application.

The “Clone” button at the bottom creates a new independent browser window. The “Exit” button destroys the browser window. The **Main Browser** cannot be destroyed (only iconized).

The middle part of the browser is divided into two windows:

1. The left hand “class window” shows the list of all currently connected classes of objects. Some classes, e.g. the command tree and the file system, are predefined. Other classes allow to attach new files using the commands in the “File” menu. Clicking with the left mouse button on one of the items in the class window displays its content in the other window. Pressing the right mouse button inside the class window shows a popup menu of possible operations, e.g. creating a new object in the current directory.

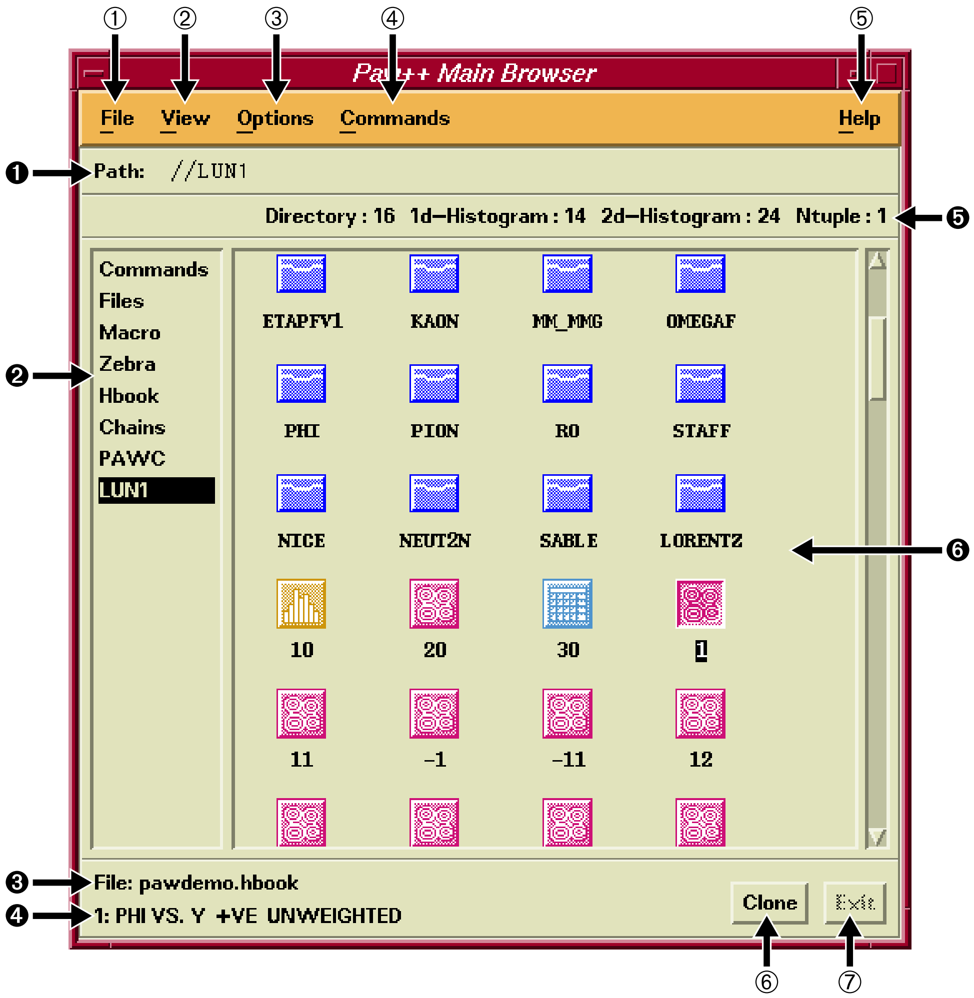
2. The right hand “object window” shows the content of the currently selected class directory. The “View” menu allows the change the way objects are displayed, i.e. to choose the icon size and the amount of information shown for each object. Objects are selected by clicking on them with the left mouse button. Pressing the right mouse button pops up a menu of possible operations depending on the object type.

An item in a popup menu is selected by pointing at the corresponding line and releasing the right mouse button. Double clicking with the left mouse button is equivalent to selecting the first menu item.

Each menu item executes a command sequence where the name of the selected object is filled into the appropriate place. By default the command is executed immediately whenever possible. The commands executed can be seen by selecting “Echo Commands” in the “Options” menu of the **Executive Window**. In case some mandatory parameters are missing a panel is displayed where the remaining arguments have to be filled in. The command is executed then by pressing the “OK” or “Execute” button in that panel. (If it is not the last one in the sequence of commands bound to the menu item the application is blocked until the “OK” or “Cancel” button is pressed.)

The immediate command execution can be inhibited by holding down the CTRL-key BEFORE pressing the right mouse button. Some popup menus also contain different menu item for immediate and delayed execution, e.g. “Execute” and “Execute...” for class “Commands”

The path of the currently selected directory is always displayed below the menu bar. The directory can be changed by pointing at the tail of the wanted subpath and clicking the left mouse button. Clicking a second time on the same path segment performs the directory change and updates the object window. To go downwards in the directory hierarchy double click on the subdirectory displayed in the object window.



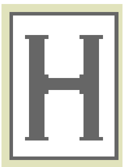
❶ Current PATH (“PATH window”).
❷ Class window.
❸ Name of file currently selected in the class window.
❹ Name of the object currently selected in the object window.
❺ Number and type of object currenlty in the the object window.
❻ Object window.

① File menu.
② View menu.
③ Options menu.
④ Commands menu.
⑤ Help menu.
⑥ Clone button.
⑦ Exit button.

9.2.1 The objects in the “object window”

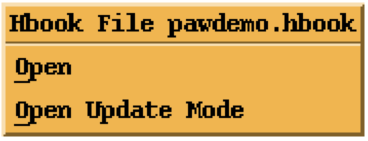
This section describes all the PAW++ object available in the **Main Browser**.

HBOOK files



Double click with the left mouse button on this icon, open the corresponfing HBOOK file with the command `HISTOGRAM/FILE`.

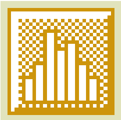
Select a **HBOOK files** icon with the left mouse button and press the right mouse button to obtain the following menu:



Open	Open the highlighted HBOOK file in read-only mode.
Open Update Mode	Open the highlighted HBOOK file in update mode.

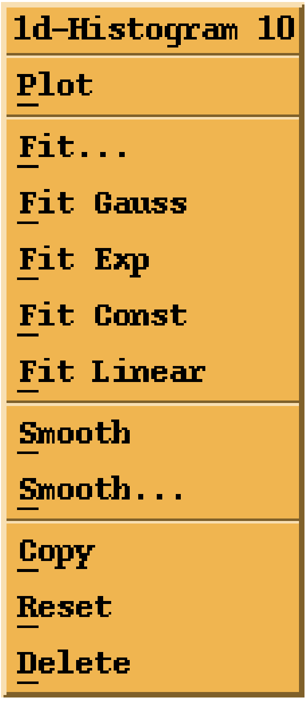
Note that the HBOOK file name is displayed in the menu title.

1D histograms



Double click with the left mouse button on this icon, produce the plot of the corresponding histogram with the command `HISTOGRAM/PLOT`. The histogram becomes the current histogram for the **Histogram Style Panel**.

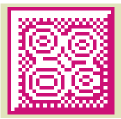
Select a **1D histograms** icon with the left mouse button and press the right mouse button to obtain the following menu:



Plot	Plot the corresponding histogram (default action). The histogram becomes the current histogram for the **Histogram Style Panel**.
Fit...	Perform the command `Histo/Fit` on the corresponding histogram. The command panel is automatically displayed
Fit Gauss	Perfom a gaussian fit on the corresponding histogram.
Fit Exp	Perform an exponential fit on the corresponding histogram.
Fit Const	Perform a `P0` fit on the corresponding histogram.
Fit Linear	Perform a `P1` fit on the corresponding histogram.
Smooth	Smooth the corresponding histogram.
Smooth...	Perform the command `Smooth` on the corresponding histogram. The command panel is automatically invoked.
Copy	Copy corresponding histogram onto an other histogram. The command panel is automatically invoked.
Reset	Reset the corresponding histogram.
Delete	Delete the corresponding histogram.

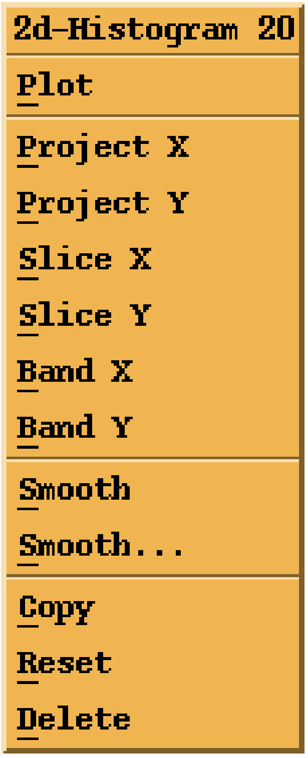
Note that the histogram identifier is displayed in the menu title.

2D histograms



Double click with the left mouse button on this icon, produce the plot of the corresponding histogram with the command `HISTOGRAM/PLOT`. The histogram becomes the current histogram for the **Histogram Style Panel**.

Select a **2D histograms** icon with the left mouse button and press the right mouse button to obtain the following menu:



Plot Plot the corresponding histogram (default action). The histogram becomes the current histogram for the **Histogram Style Panel**.

Project X Generate the `X` projection, perform the projection and plot the result (commands `ProX`, `Hi/Proj`, and `Hi/Plot`).

Project Y Generate the `Y` projection, perform the projection and plot the result (commands `ProY`, `Hi/Proj`, and `Hi/Plot`).

Slice X Generate the `X` slices, perform the projection and plot the first slice (commands `SliX`, `Hi/Proj`, and `Hi/Plot`).

Slice Y Generate the `Y` slices, perform the projection and plot the first slice (commands `SliY`, `Hi/Proj`, and `Hi/Plot`).

Band X Generate the `X` bands, perform the projection and plot the first band (commands `BanX`, `Hi/Proj`, and `Hi/Plot`).

Band Y Generate the `Y` bands, perform the projection and plot the first band (commands `BanY`, `Hi/Proj`, and `Hi/Plot`).

Smooth Smooth the corresponding histogram.

Smooth... Perform the command `Smooth` on the corresponding histogram. The command panel is automatically invoked.

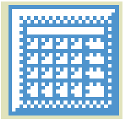
Copy Copy corresponding histogram onto an other histogram. The command panel is automatically invoked.

Reset Reset the corresponding histogram.

Delete Delete the corresponding histogram.

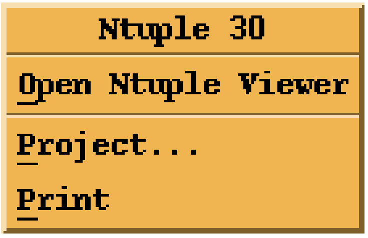
Note that the histogram identifier is displayed in the menu title.

Ntuples



Double click with the left mouse button on this icon, open the **Ntuple Viewer** on the corresponding Ntuple.

Select a **Ntuples** icon with the left mouse button and press the right mouse button to obtain the following menu:



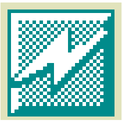
Open Ntuple Viewer Open Ntuple Viewer on the highlighted Ntuple.

Project... Project the highlighted Ntuple. The Command panel `Ntuple/Proj` is automatically invoked.

Print Print the highlighted Ntuple (Command `Ntuple/Print`).

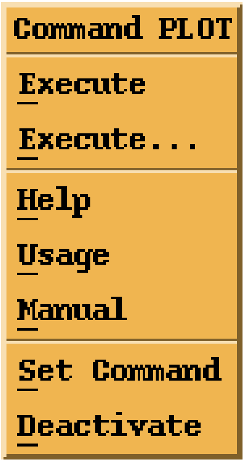
Note that the Ntuple identifier is displayed in the menu title.

PAW commands



Double click with the left mouse button on this icon, execute the corresponding PAW command.

Select a PAW **commands** icon with the left mouse button and press the right mouse button to obtain the following menu:



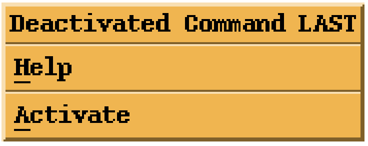
Execute	Execute the command with the default parameters. If a mandatory parameter is missing, the command panel is automatically invoked.
Execute...	Display the command panel.
Help	Display the help on the command.
Usage	Display the command usge in the **Transcript Pad** of the **Executive Window**.
Manual	Equivalent to `HELP`.
Set Command	This command becomes the one executed when a directive typed on the keyboard is not an existing PAW command.
Deactivate	The command is deactivated.

Note that the command name is displayed in the menu title.

Deactivated PAW commands

Double click with the left mouse button on this icon, execute the help on corresponding PAW command.

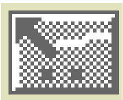
Select a **Deactivated** PAW **commands** icon with the left mouse button and press the right mouse button to obtain the following menu:



Help	Display the help on the command.
Activate	The command is activated.

Note that the deactivated command name is displayed in the menu title.

Up

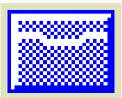


Double click with the left mouse button on this icon, allow to go one level up in the directory tree. This icon is alway the first one of the **content window**.

Select a **Up** icon with the left mouse button and press the right mouse button to obtain the following menu:

List	Allow to go one level up in the directory tree.

Directory

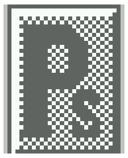


Double click with the left mouse button on this icon, change the current working directory.

Select a **Directory** icon with the left mouse button and press the right mouse button to obtain the following menu:

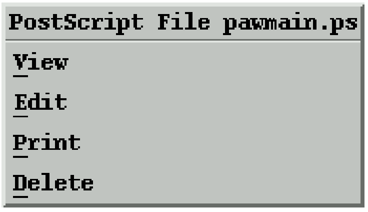
List	Change the current working directory.

PostScript files



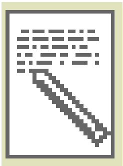
Double click with the left mouse button on this icon, invoke the ghostview on the corresponding file.

Select a **PostScript files** icon with the left mouse button and press the right mouse button to obtain the following menu:



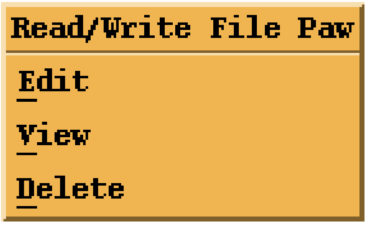
View	Invoke GhostView on the file.
Edit	Edit the file.
Print	Print the file.
Delete	Delete the file.

Read-Write files



Double click with the left mouse button on this icon, invoke the editor on the corresponding file.

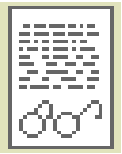
Select a **Read-Write files** icon with the left mouse button and press the right mouse button to obtain the following menu:



Edit	Edit the file.
View	Read the file.
Delete	Delete the file.

Note that the file name is displayed in the menu title.

Read-only files



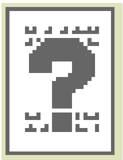
Double click with the left mouse button on this icon, invoke the editor in view mode on the corresponding file.

Select a **Read-only files** icon with the left mouse button and press the right mouse button to obtain the following menu:

View	Read the file.
Delete	Delete the file.

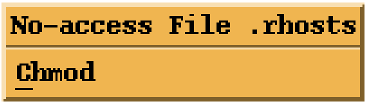
Note that the file name is displayed in the menu title.

No-access files



Double click with the left mouse button on this icon, invoke the shell command `chmod` on the corresponding file.

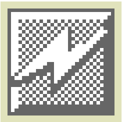
Select a **No-access files** icon with the left mouse button and press the right mouse button to obtain the following menu:



Chmod Try to change the permissions of the file.

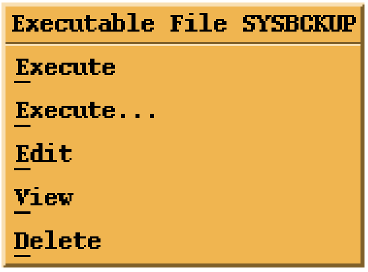
Note that the file name is displayed in the menu title.

Executable files



Double click with the left mouse button on this icon, invoke the command SHELL on the corresponding file.

Select a **Executable files** icon with the left mouse button and press the right mouse button to obtain the following menu:



Execute Invoke the command SHELL on the file.

Execute... Open the command panel SHELL with the file name.

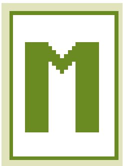
Edit Edit the file.

View Read the file.

Delete Delete the file.

Note that the file name is displayed in the menu title.

PAW **Macros**



Double click with the left mouse button on this icon, execute the corresponding macro.

Select a PAW **Macros** icon with the left mouse button and press the right mouse button to obtain the following menu:

Exec Execute the macro.

Exec... Open the command panel EXEC with the macro name. It is useful to give parameters to the macro.

Edit Edit the macro.

View Read the macro.

Delete Delete the macro.

Note that the macro name is displayed in the menu title.

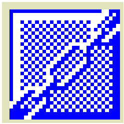
Pictures

Double click with the left mouse button on this icon, plot the corresponding picture.

Select a **Pictures** icon with the left mouse button and press the right mouse button to obtain the following menu:

Plot	Plot the highlighted picture.
Do PostScript	Produce the PostScript file `PNAME.ps`, where `PNAME` is the name of the highlighted picture.
Create	Create a new picture. The command panel `Picture/Create` is automatically invoked.
Rename	Rename the highlighted picture. The command panel `Picture/Rename` is automatically invoked.
Delete	Rename the highlighted picture.

Chains

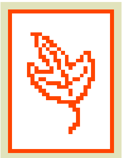


Double click with the left mouse button on this icon, allow to go one level deeper in the chain tree.

Select a **Chains** icon with the left mouse button and press the right mouse button to obtain the following menu:

List	List the available chains.
Show Tree	Show the tree from the highlighted chain.
Delete Chain	Delete the highlighted chain.

Last chain level

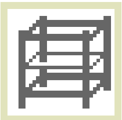


Last chain element.

Select a **Last chain level** icon with the left mouse button and press the right mouse button to obtain the following menu:

List	List the available chains.
Delete Chain Entry	Delete the highlighted chain element.

ZEBRA Stores

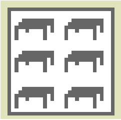


Double click with the left mouse button on this icon, allow to go inside the corresponding ZEBRA store.

Select a **ZEBRA Stores** icon with the left mouse button and press the right mouse button to obtain the following menu:

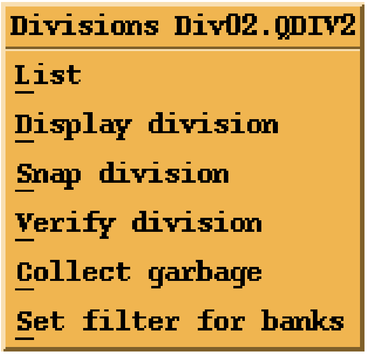
List	Display divisions of the store
Show store DZSTOR	Show parameters of the store (`CALL DZSTOR`)

ZEBRA Divisions



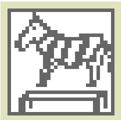
Double click with the left mouse button on this icon, allow to go inside the corresponding ZEBRA division.

Select a **ZEBRA Divisions** icon with the left mouse button and press the right mouse button to obtain the following menu:



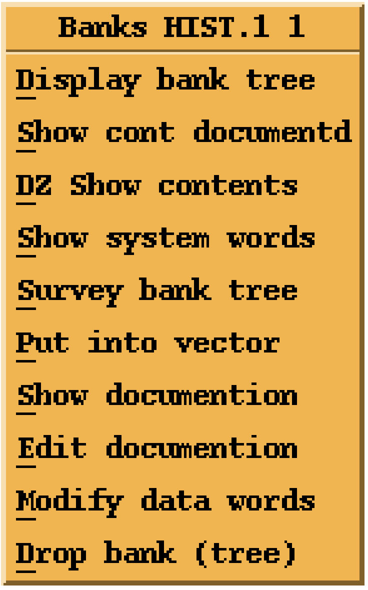
List	Display banks of the division as icons.
Display division	Show layout of banks in divisions graphically.
Snap division	Show a snapshot of division parameters. (`CALL DZSNAP`).
Verify division	Verify division (`CALL DZVERI`).
Collect garbage	`CALL MZGARB` in selected division.
Set filter for banks	Allow to display only banks whose hollerith. identifiers match a wild card selection.

ZEBRA Banks



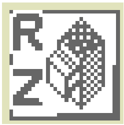
Double click with the left mouse button on this icon, draw the bank tree from the corresponding ZEBRA bank.

Select a **ZEBRA Banks** icon with the left mouse button and press the right mouse button to obtain the following menu:



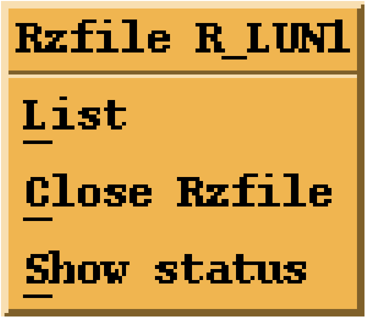
Display bank tree	Display graphically the structure below the selected bank (see picture banktree.eps).
Show cont documented	Display the data of the bank with their description if a documentation data base is provided (see CERN Q101).
DZ Show contents	`CALL DZSHOW` fore selected bank.
Show system words	List contents of the links and system words.
Survey bank tree	`CALL DZSURV` for selected bank.
Put into vector	Put data contents of the bank into a KUIP vector.
Show documentation	Display the documentation for the bank (if provided).
Edit documentation	Edit a bank descriptor, if no available yet provide a template.
Modify data words	Self explaining.
Drop bank (tree)	Self explaining.

RZ Files



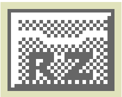
Double click with the left mouse button on this icon, allow to go inside the corresponding ZEBRA/RZ file.

Select a RZ **Files** icon with the left mouse button and press the right mouse button to obtain the following menu:



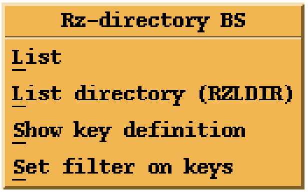
Close RZfile	Self explaining.
List	Display keys.
List directory	`CALL RZLDIR`.
Show key definition	self explaining.
Set filter on keys	Allow to display only entries whose key words match a wild card selection.
Show status	`CALL RZSTAT`.

RZ **Directories**



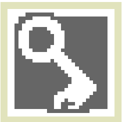
Double click with the left mouse button on this icon, allow to go inside the corresponding ZEBRA/RZ directory.

Select a RZ **Directories** icon with the left mouse button and press the right mouse button to obtain the following menu:



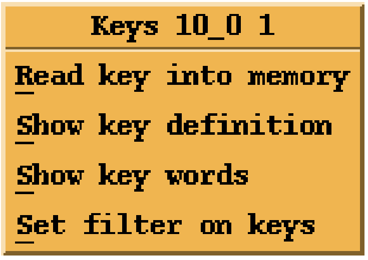
List	List the highlighted RZ directory.
List directory (RZLDIR)	Perform `RZLDIR` on the highlighted RZ directory.
Show key definition	Display the key definition.
Set filter on keys	Defines a filter on the keys.

RZ **Keys**



Double click with the left mouse button on this icon, allow to read into memory the corresponding ZEBRA/RZ key.

Select a RZ **Keys** icon with the left mouse button and press the right mouse button to obtain the following menu:

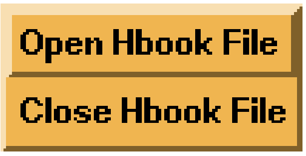


Read key into memory	Allow to inspect the data of a key.
Show key definition	Self explaining.
Show key words	Self explaining.
Set filter on keys	See above.

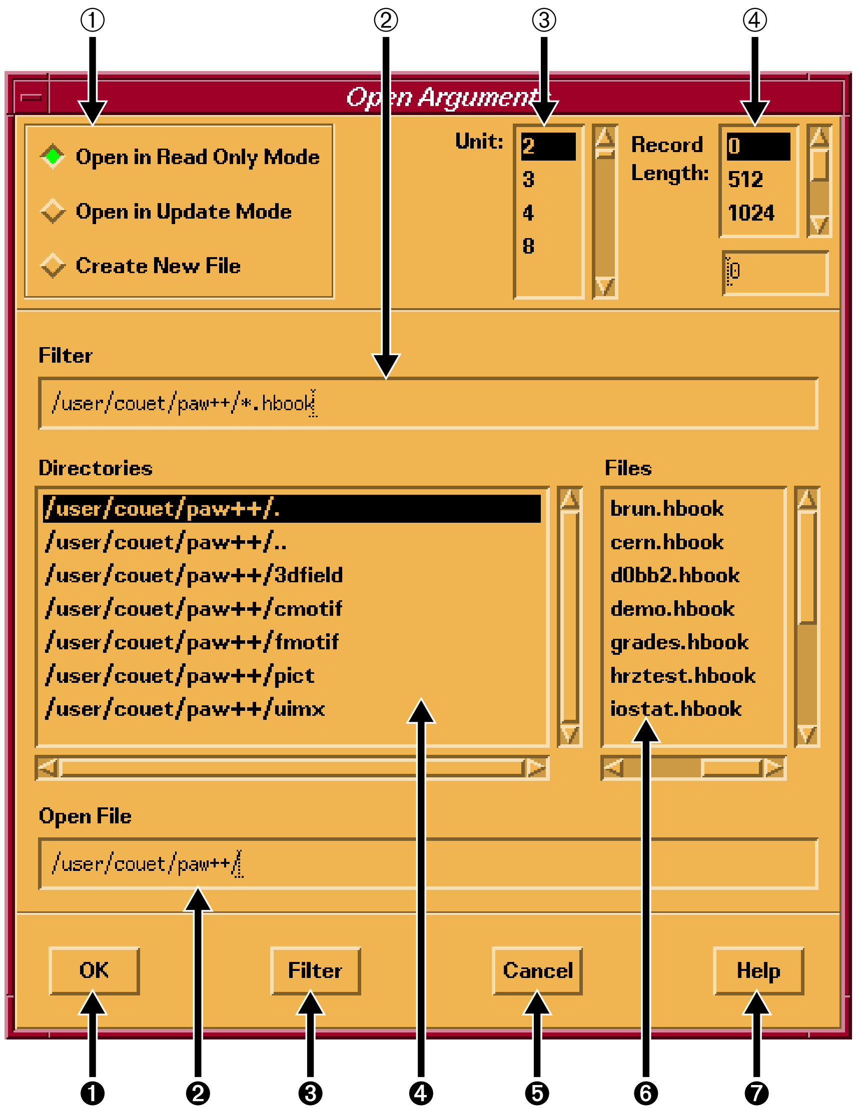
9.2.2 The Main Browser Menu Bar

In this section, is describe the full functionality of the pull down menu available in the Menu Bar of the **Main Browser**.

File



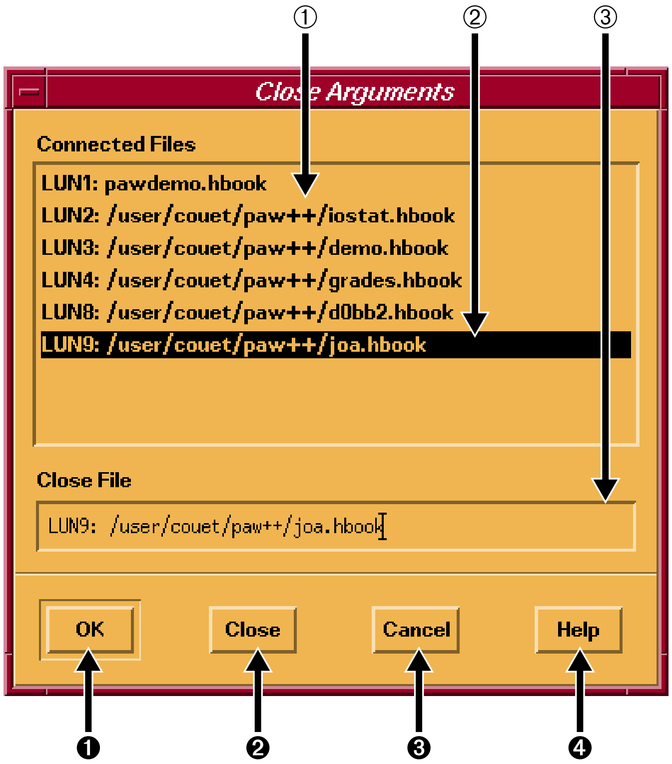
Open Hbook file	Display the **Open Arguments** panel (see after).
Close Hbook file	Display the **Close Arguments** panel (see after).



① Toggle buttons to choose the openning mode.
② Filter apply on the file list ❻.
③ Possible logical units. Only the free units are displayed. The next free unit is highlighted. Any other unit is invalid.
④ Possible record length. A record length of `0` means that the system will compute the correct one automatically.

❶ The file is open and this panel is closed.
❷ File name of the opened file.
❸ Apply the filter defined in ②.
❹ List of the subdirectories available. Double click on a directory name change the current directory.
❺ Cancel the current opened panel and clode it.
❻ List of the file in the current directory matching the filter.
❼ Help

Note that a double click with the left mouse button on a HBOOK file icon in the object window of the **Main Browser** open also the HBOOK file. This panel is usefull to specify a filter different form the default filter `*.hbook` used in the object window.



① List of the currently connected hbook files.

② A simple click with the left mouse button a file name in the connected files list, highlight the filename and put it in the **Close file** field ③.

③ Name of the file to be closed. This field can be filled directly by tipyng on the keyboard, or by a simple click with the left mouse button in the **Connected Files** list ①.

❶ When a file is selected, clicking on this button or typing `<CR>` allows to perform the action (close the file) and close the panel.

❷ Close the selcted file and leave the panel opened.

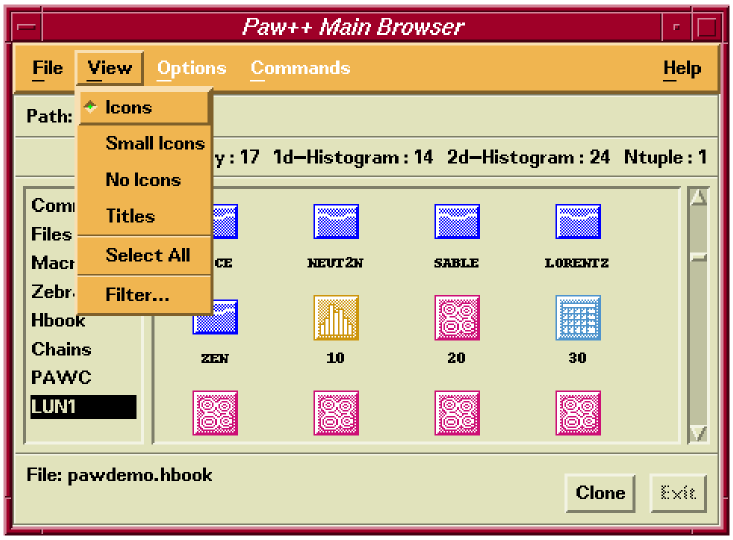
❸ Cancel the current operation and close the panel.

❹ Give some help.

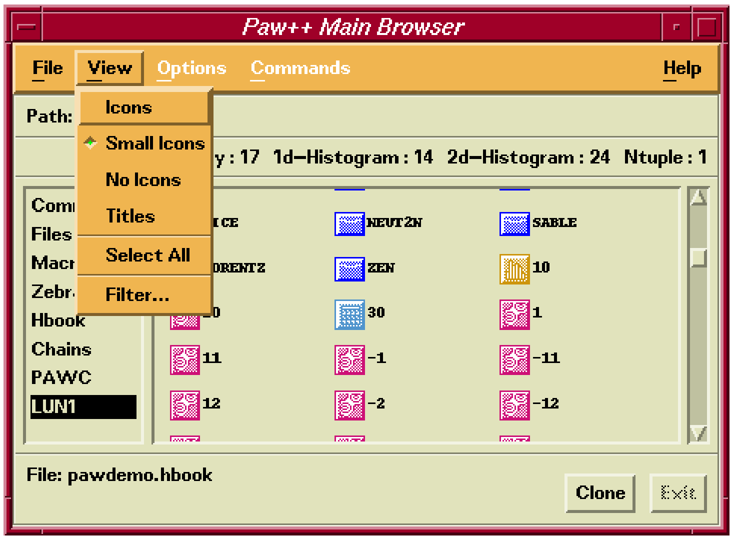
View

This pull down menu allows to define the "viewing" for the objects in the "object window" of the **Main Browser**.

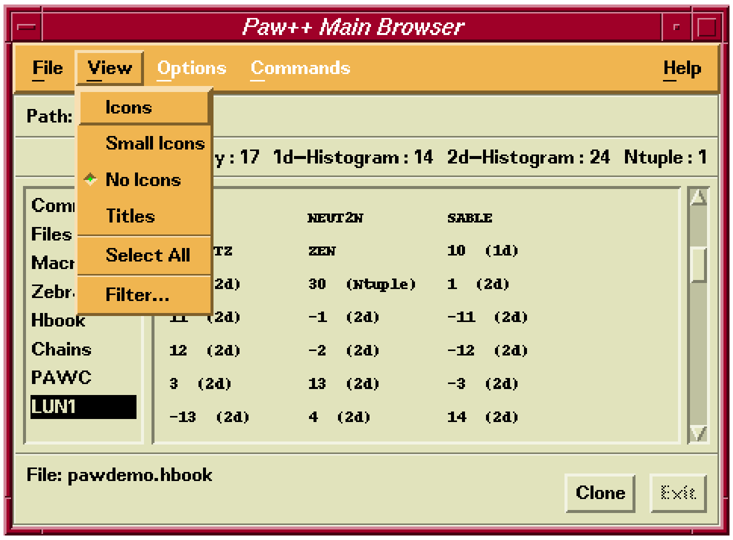
Icons	The objects are represented with big icons (default).
Small Icons	The objects are represented with small icons.
No Icons	Only the object identifier and type are displayed.
Titles	Small icons, objects identifiers and titles are displayed.
Select All	All the objects are selected.
Filter...	Apply a filter on object names.



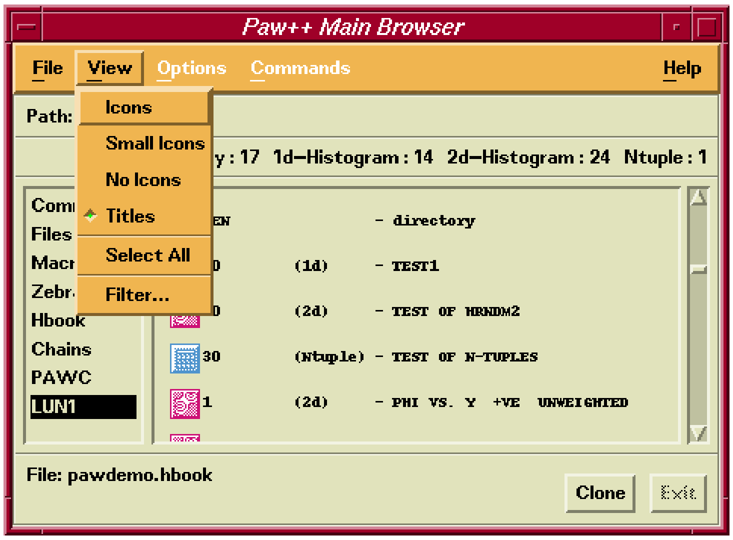
Icons: icons and the object identifiers are displayed.



Small Icons: small icons and the object identifiers are displayed.



No Icons : object identifiers and titles are displayed.



Titles : small icons and the object identifiers and titles are displayed.

Options

Raise Window	Raise a given window.
Command Argument Panel...	Get help on a given command.

Commands

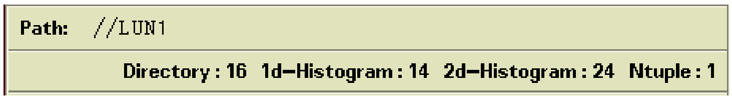
This menu allows to access the tree of the PAW commands. Only the top levels are describe in this section. Note the tree of the PAW commands can also be accessed via the item “Commands” in the “PATH Window” of the **Main Browser**.

Kuip	Command Processor commands.
Macro	Macro Processor commands.
Vector	Vector Processor commands.
Histogram	Manipulation of histograms, Ntuples.
Function	Operations with Functions. Creation and plotting.
Ntuple	Ntuple creation and related operations.
Graphics	Interface to the graphics packages HPLOT and HIGZ.
Picture	Creation and manipulation of HIGZ pictures.
Fortran	Interface to MINUIT, COMIS, SIGMA and FORTRAN Input/Output.
Network	To access files on remote computers.
Dzdoc	Access Dzdoc

Help

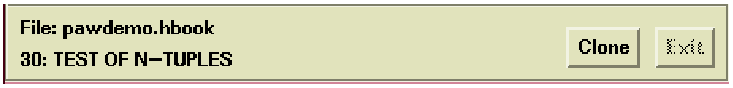
9.2.3 Information Windows

Top



On the top of the **Main Browser** is displayed the current directory PATH and the content of the current directory i.e. the number of objects of each type.

Bottom

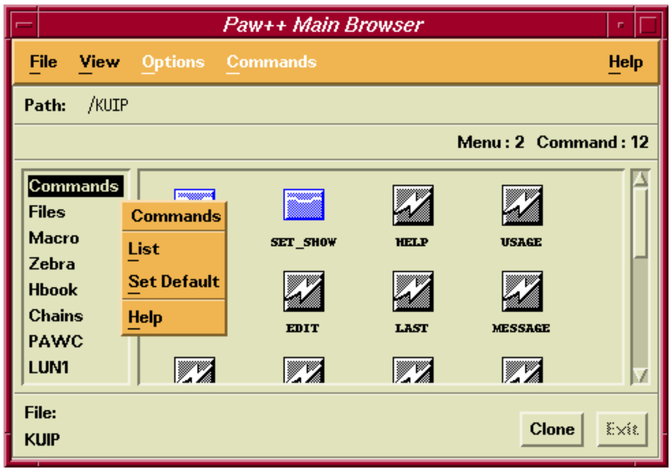


On the bottom of the **Main Browser** is displayed the name of the current file (HBOOK files for example) in which the objects are stored. If the objects are not stored in a file (like the commands), the file name is just blank. Below the file name, the full name of the currently selected object is displayed.

9.2.4 Content Window

In this section are describe the different menu available in the “Content Window”.

Commands

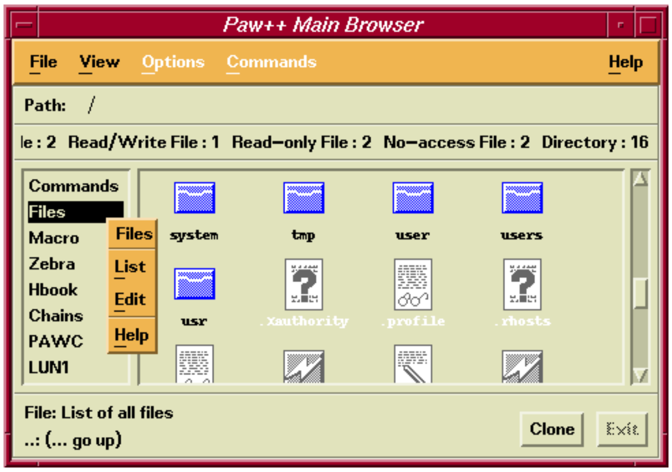


List List the content of the current menu.

Set Default Set the root for searching commands to /.

Help Display some help.

Files



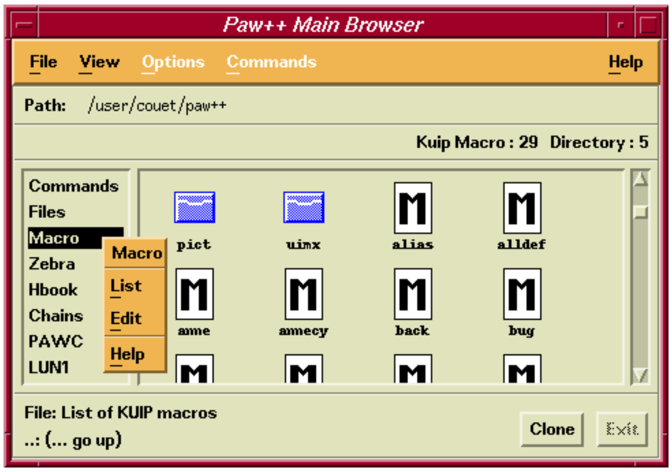
List List the content of the current working directory (OS).

Chdir to ... Change directory.

Edit Edit a file.

Help Display some help.

Macro

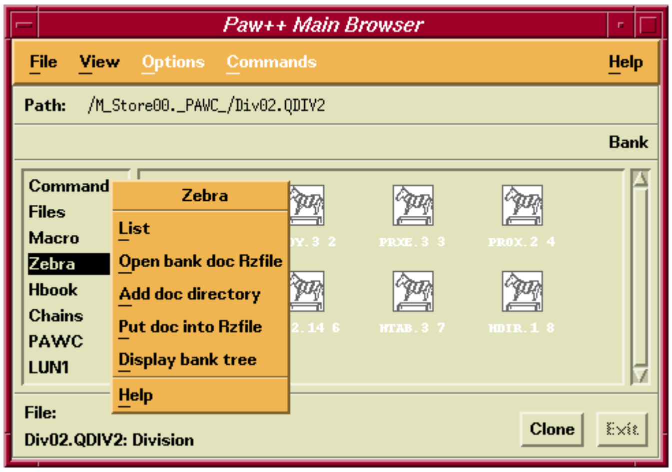


List List all the macros in the current working directory.

Edit Edit a macro.

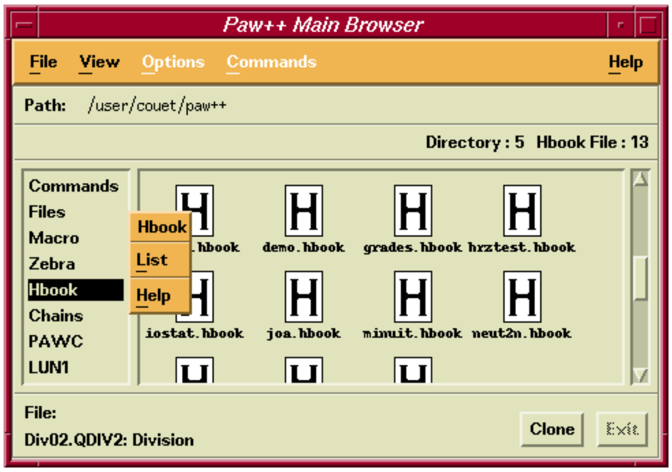
Help Display some help.

Zebra



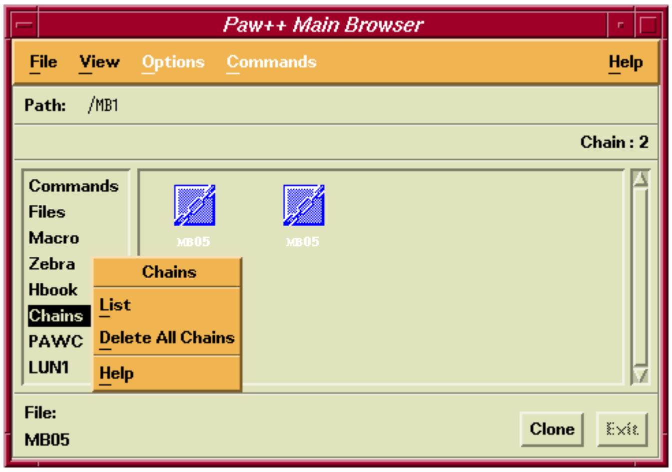
List	List the ZEBRA file connected.
Open bank doc Rzfile	Open bank doc Rzfile.
Add doc directory	Add doc directory.
Put doc into Rzfile	Put doc into Rzfile.
Display bank tree	Display bank tree.
Help	Display some help.

Hbook

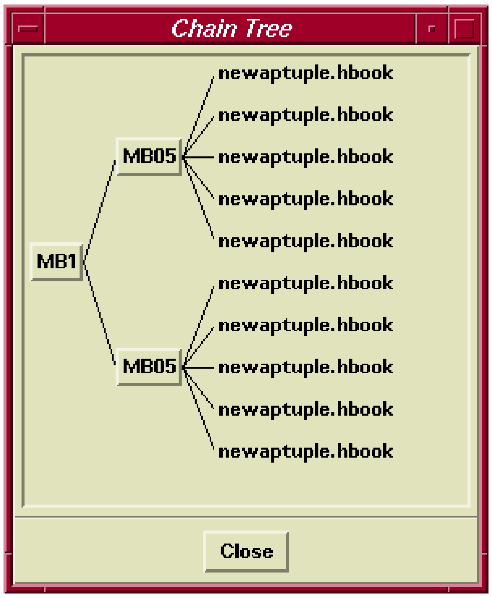


List	List all the HBOOK files in the current working directory.
Help	Display some help.

Chains

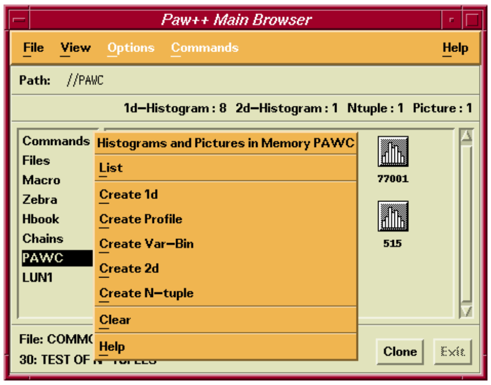


List	List the chains currently in memory.
Delete All Chains	Delete all the chains from memory.
Help	Display some help.



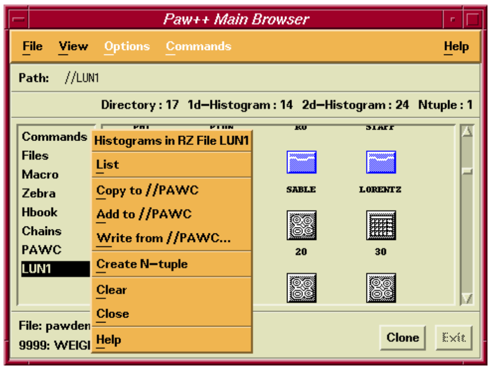
This panel allows to navigate in the chain tree. Just clicking on a chain name change the level from which the chain will be traversed.

PAWC



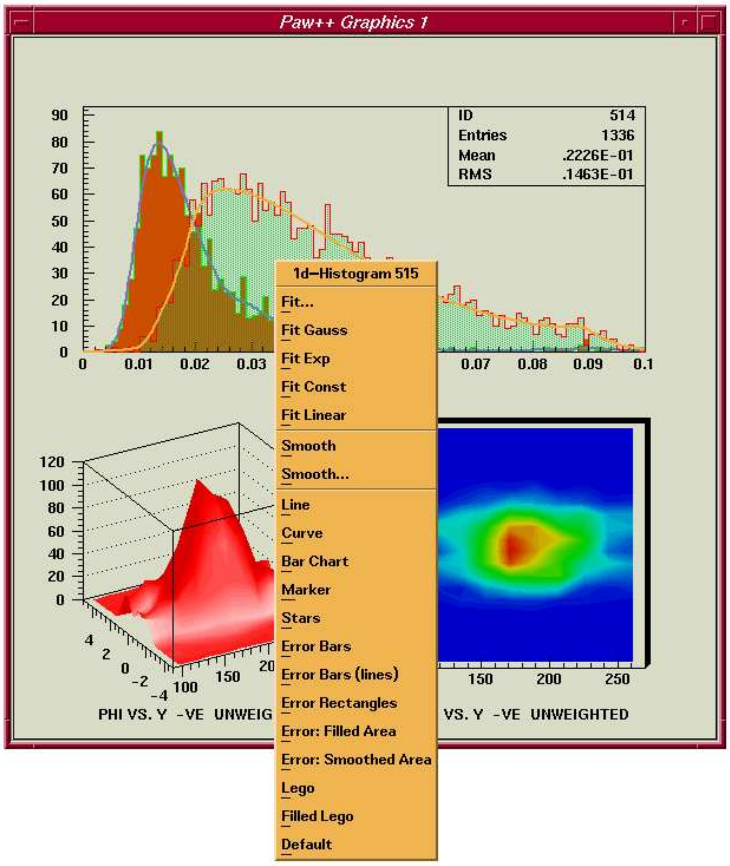
List	List all the HBOOK objects in memory.
Create 1d	Create a 1d histogram.
Create Profile	Create a Profile histogram.
Create Var-Bin	Create a variable bin size histogram.
Create 2d	Create a 2d histogram.
Create N-tuple	Create a row wise Ntuple histogram.
Clear	Delete histograms from memory.
Help	Provide some help.

Hbook Files (//LUNn)



List	List all the HBOOK objects in this file.
Copy to //PAWC	Copy the highlighted HBOOK object in memory.
Add to //PAWC	Add the highlighted HBOOK object in memory.
Write from //PAWC...	Save the highlighted HBOOK object on disk.
Create N-tuple	Create a row wise Ntuple histogram.
Clear	Delete histograms from disk.
Close	Close the selected hbook file
Help	Provide some help.

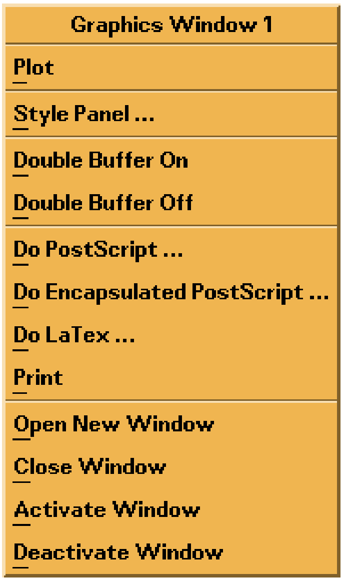
9.3 Graphics



PAW++ allows direct graphics manipulation of the objects like Histograms or Ntuples. To perform actions on object from the **Graphics Window**, it is enough to move the mouse cursor on the **Graphics Window** and to click with the right mouse button on the object. A pull down menu will be displayed according to the object picked. In this section are described the different menus available in the **Graphics Window**.

9.3.1 The Graphics Window

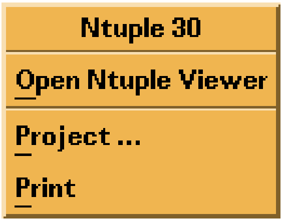
When no object is picked in the **Graphics Window** for instance when the background of the window is picked the following menu is displayed.



Plot	PLot the current picture.
Style Panel...	Invoke the **Histogram Style Panel**.
Double Buffer On	Set the double buffer on.
Double Buffer Off	Set the double buffer off.
Do PostScript...	Generate the Postscipt file `paw.ps`.
Do Encapsulated PostScript...	Generate the Encapsulated Postscipt file `paw.eps`.
Do LaTex...	Generate the LaTex file `paw.tex`.
Print	Print the current picture.
Open New Window	Open a new window.
Close Window	Close the current window.
Activate Window	Activate the current window.
Deactivate Window	Deactivate the current window.

9.3.2 Ntuple

An Ntuple picked in **Graphics Window** with the right mouse button displays the following menu:



Open Ntuple Viewer	Open the Ntuple browser.
Project...	Project the picked ntuple.
Print	Print the picked ntuple

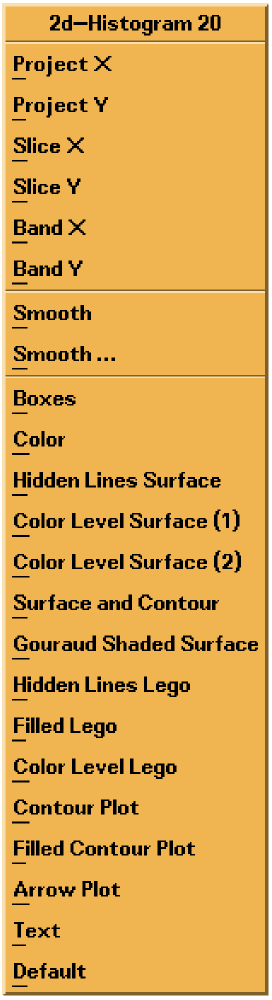
9.3.3 1D-Histogram

When a 1D-Histogram is picked in **Graphics Window** with the right mouse button, the following menu is displayed:

Fit Command...	Invoke the fit command.
Fitting panel...	Invoke the fit panel.
Fit Gauss	Perform a gaussian fit.
Fit Exp	Perform a exponential fit.
Fit Const	Fit with a constant.
Fit Linear	Perform a linear fit.
Smooth	Smooth.
Smooth...	Invoke the smooth command.
Line	Draw the histogram with a line.
Curve	Draw the histogram with a curve.
Bar Chart	Draw the histogram as a bar chart.
Marker	Draw the histogram with markers.
Stars	Draw the histogram with stars.
Error Bars	Draw the histogram with error bars.
Error Bars (lines)	Draw the histogram with error bars ended with tick marks.
Error Rectangles	Draw the histogram with error rectangles.
Error: Filled Area	Draw the histogram as a filled area.
Error: Smoothed Area	Draw the histogram a a smoothed and filled area.
Lego	Draw the histogram as a lego plot.
Filled Lego	Draw the histogram as a filled lego plot.
Default	Default histogram drawing.

9.3.4 2D-Histogram

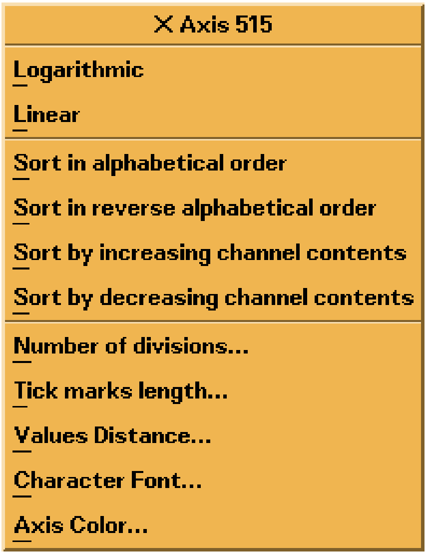
When a 2D-Histogram is picked in **Graphics Window** with the right mouse button, the following menu is displayed:



Project X	Fill the X projection and display it.
Project Y	Fill the Y projection and display it.
Slice X	Define slices on X and display slice 1.
Slice Y	Define slices on Y and display slice 1.
Band X	Define bands on X ans display band 1.
Band Y	Define bands on Y and display band 1.
Smooth	Smooth the picked histogram.
Smooth...	Display the smooth panel on the picked histogram.
Boxes	Boxes plot.
Color	Color plot
Hidden Lines Surface	Hidden lines surface plot.
Color Level Surface (1)	Color level surface plot (1).
Color Level Surface (2)	Color level surface plot (2).
Surface and Contour	Surface and contour plot.
Gouraud Shaded Surface	Gouraud shaded surface plot.
Hidden Lines Lego	Hidden lines lego plot.
Filled Lego	Filled lego plot.
Color Level Lego	Color level lego plot.
Contour Plot	Contour plot (line).
Filled Contour Plot	Filled contour plot.
Arrow Plot	Arrow plot.
Text	Text plot.
Default	Default (scatter plot or text plot).

9.3.5 X Axis

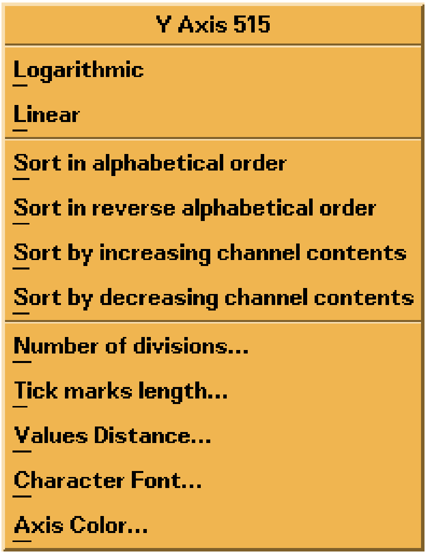
When a X-Axis is picked in **Graphics Window** with the right mouse button, the following menu is displayed:



Logarithmic	Log scale on.
Linear	Linear scale on.
Sort in alphabetical order	Reorder the bins.
Sort in reverse alphabetical order	Reorder the bins.
Sort by increasing channel contents	Reorder the bins.
Sort by decreasing channel contents	Reorder the bins.
Number of divisions...	Define number of X divisions.
Tick marks length...	Tick marks size.
Values Distance...	Labels distance.
Character Font...	Labels font.
Axis Color...	Axis color.

9.3.6 Y Axis

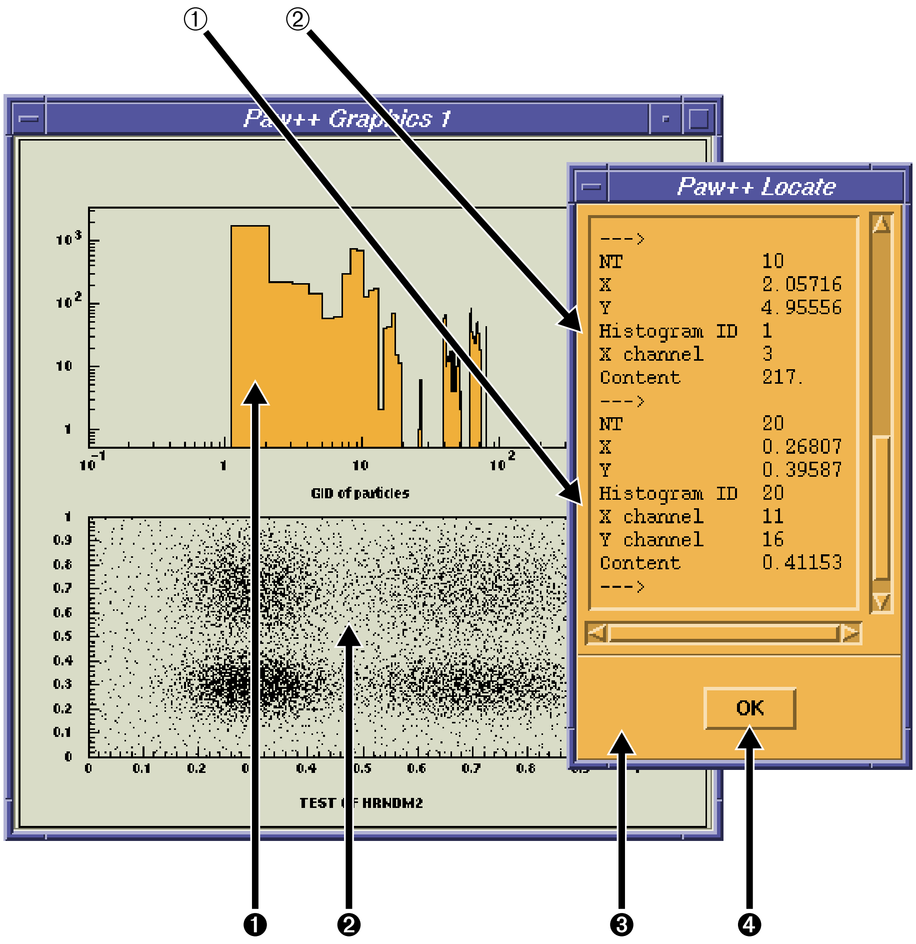
When a Y-Axis is picked in **Graphics Window** with the right mouse button, the following menu is displayed:



Logarithmic	Log scale on.
Linear	Linear scale on.
Sort in alphabetical order	Reorder the bins.
Sort in reverse alphabetical order	Reorder the bins.
Sort by increasing channel contents	Reorder the bins.
Sort by decreasing channel contents	Reorder the bins.
Number of divisions...	Define number of Y divisions.
Tick marks length...	Tick marks size.
Values Distance...	Labels distance.
Character Font...	Labels font.
Axis Color...	Axis color.

9.3.7 Locate on Histograms

To retrieve interactively on the **Graphics Window** an histogram identifier a bin number, a `(X,Y)` position etc... , place the mouse cursor on the graphics area and click with the left mouse button on the interesting region. The information about the picked histogram will appear in the window called **PAW++ Locate**.

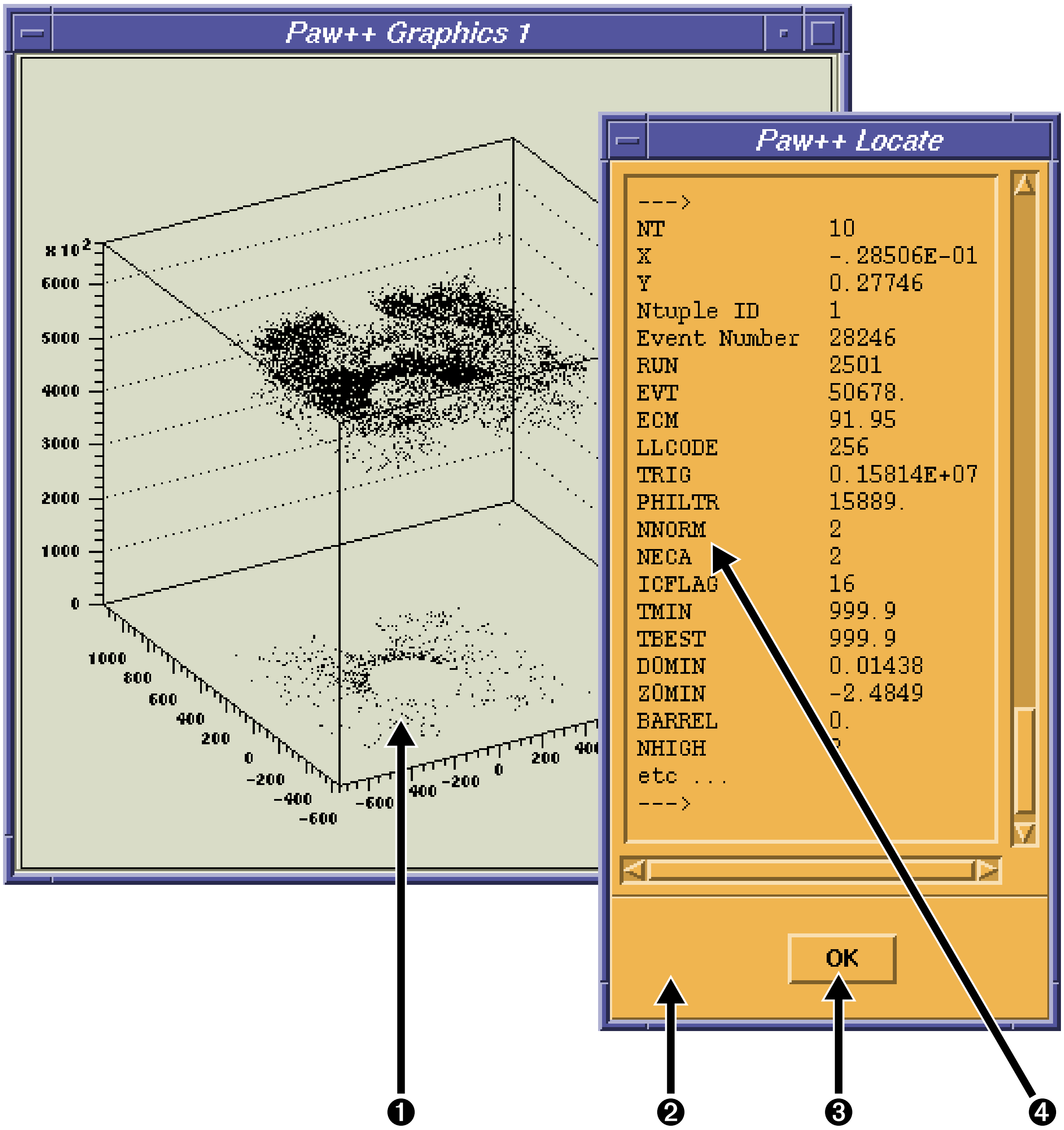


❶ 1D Histogram (with LOG scale).
❷ 2D Histogram.
❸ **PAW++ Locate** window.
❹ To release the Output window.

① Info the the 1D Histogram.
② Info the the 2D Histogram.

9.3.8 Locate on Ntuples

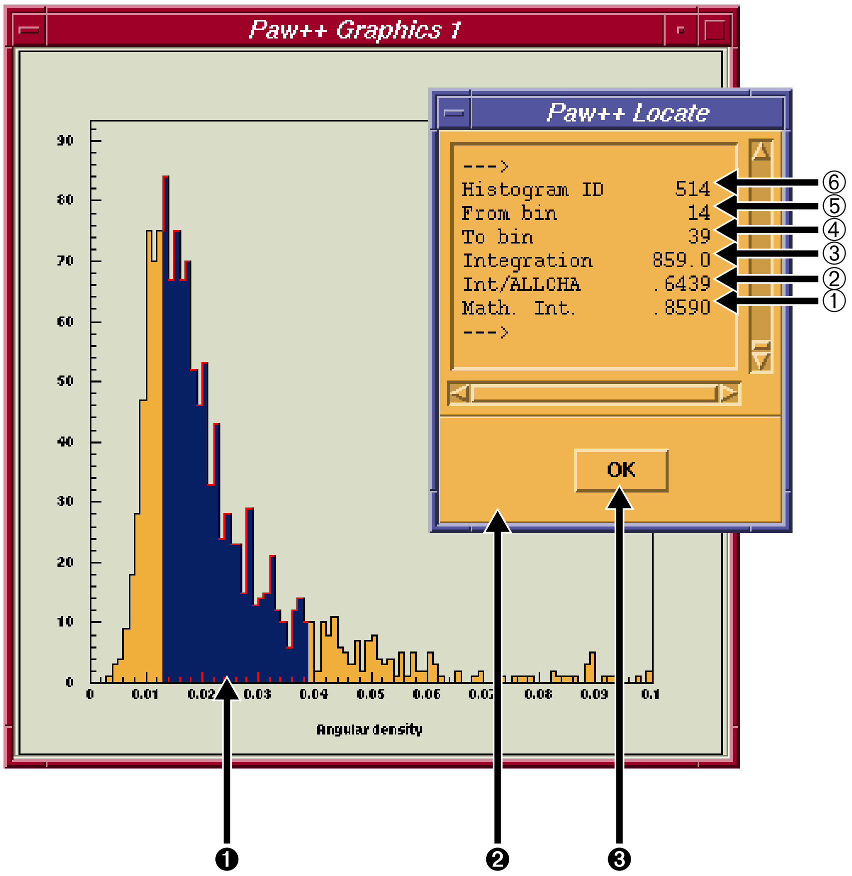
Just by clicking with the left mouse button on a Ntuple drawing, one can get the event description in the **PAW++ Locate** window. If the mouse cursor is moved on the Ntuple drawing with the left mouse button pressed, the event description will change in real time in **PAW++ Locate**.



❶ Ntuple drawing.
❷ **PAW++ Locate** window.
❸ To release the Output window.
❹ event description.

9.3.9 Integrate Histograms

To integrate interactively an histogram, place the mouse cursor on the bin from which the integration will start, and drag the cursor with the left mouse button pressed to the last bin. The result will appears in real time in a separated window called **PAW++ Locate** ❷.

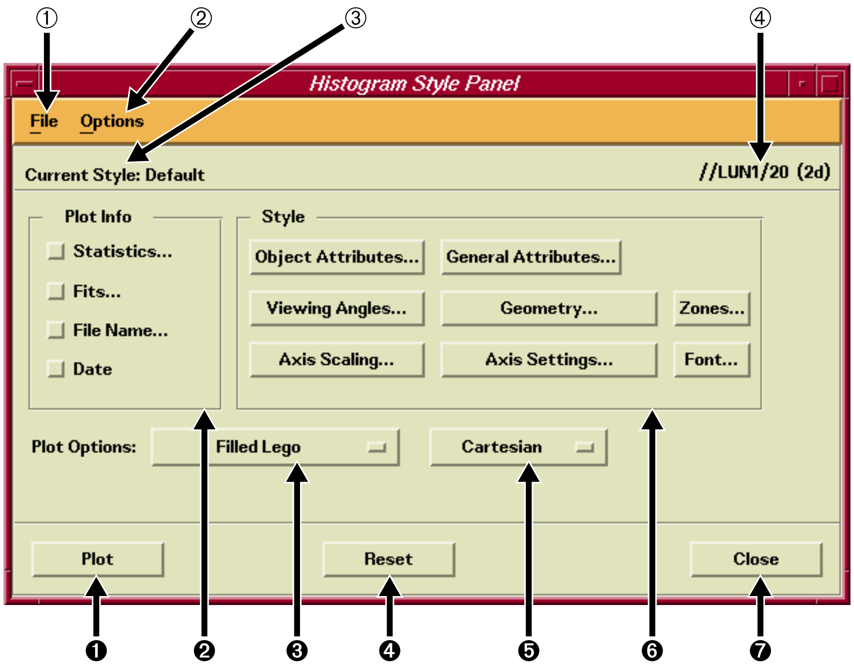


❶ Integrated area.
❷ Output window. It is possible to copy (via the mouse) the text inside this window.
❸ To release the Output window.

① Histogram identifier.
② First bin for the integration.
③ Last bin for the integration.
④ Value of the integral.
⑤ Normalized integral.
⑥ "Mathematical" integral. Each bin contribution is multiply by the bin witdh.

9.4 The Histogram Style Panel

The **Histogram Style Panel** allows to manipulate and present histograms. It works on one histogram only: the "Current histogram". To set the current histogram it is enough to plot it for the **Main Browser**, via a double click on the icon.



❶ Plot the current histogram.
❷ Add informations on the plots.
❸ Define the graphical option used to plot the current histogram.
❹ Reset the default attributes.
❺ Define the coordinate system used to draw lego and surface plots.
❻ Define attibutes used to draw the current histogram.
❼ Close the **Histogram Style Panel**.

① File menu.
② Options menu.
③ Current style name.
④ Current histogram name and type.

9.4.1 The Histogram Style Panel Menu Bar

In this section, is describe the full functionality of the pull down menu available in the Menu Bar of the **Histogram Style Panel**.

File

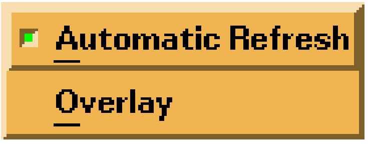
Open Style Allows to choose and execute a "Style Macro". This "Style Macro" becomes the "current style". This field ③ in the **Histogram Style Panel** is updated with the "current style" name. The "Style Macro" have by default the extension `.sty`.

Save Style Save the "current style". When a style is saved, all the current attribute values are saved in the "Style Macro".

Save Style As... Save the "current style" with a new name.

Close

Options



Automatic Refresh By default the "Automatic Refresh" is on: each time the "current picture" is changed, the graphics window is updated. When this mode is off, the user has to click on one of the Apply button available.

Overlay Each time a new histogram, vector, or ntuple drawing is produced, a clear window is performed. To superimpose all the drawing on the same image, it is enough to put this option on. This option is the equivalent of the option `S` in the command `HISTO/PLOT`.

9.4.2 Plot Info

This set of toggle buttons allow to add some usefull information on the curren plot. If the Automatic refresh mode is on, the plot is automatically refresh.

Statistics... Allow to draw (or not) the statistics on the plot (PAW command `OPTION STAT`). When the toggle button is set on, a panel is displayed in order to specify with parameters will be visible.

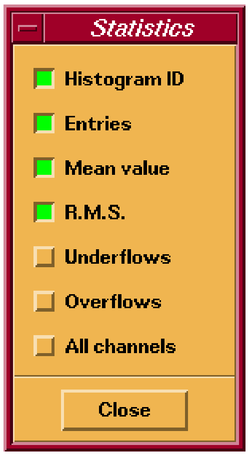
Fits... Allow to draw (or not) the fit parameters on the plot (PAW command `OPTION FIT`). When the toggle button is set on, a panel is displayed in order to specify with parameters will be visible.

File Name... Allow to draw (or not) the file name on the plot (PAW command `OPTION FILE`).When the toggle button is set on, a panel is displayed in order to specify the file name position.

Date... Allow to draw (or not) the date on the plot (PAW command `OPTION DATE`).When the toggle button is set on, a panel is displayed in order to specify the date position

Statistics ...

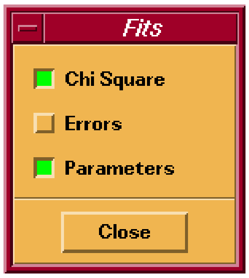
This panel in the equivalent of the PAW command `SET STAT`. It allows to specify which statistics informations are displayed on the plot.



Histogram ID	The histogram identifier is displayed.
Entries	The number of entries is displayed.
Mean value	The mean value is displayed.
R.M.S.	The R.M.S. is displayed.
Underflows	The underflows are displayed.
Overflows	The overflows are displayed.
All channels	The content of the total number of channel is displayed.

Fits ...

This panel in the equivalent of the PAW command `SET FIT`. It allows to specify which fit parameters are displayed on the plot.



Chi Square	The chi square is displayed.
Errors	The errors are displayed.
Parameters	The fit parameters are displayed.

File Name ...

This panel in the equivalent of the PAW command `SET FILE`. It allows to specify the file name position on the plot.

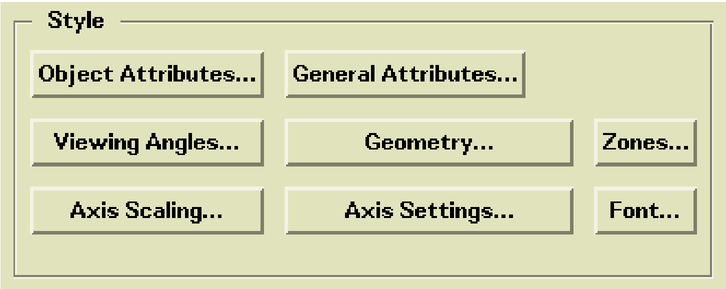
Top Left	The file name is drawn on the top left of the plot (default).
Top Right	The file name is drawn on the top right of the plot
Bottom Left	The file name is drawn on the bottom left of the plot
Bottom Right	The file name is drawn on the bottom left of the plot

Date ...

This panel in the equivalent of the PAW command SET DATE. It allows to specify the date position on the plot.

Top Left	The date is drawn on the top left of the plot
Top Right	The name is drawn on the top right of the plot (default).
Bottom Left	The date is drawn on the bottom left of the plot
Bottom Right	The date is drawn on the bottom left of the plot

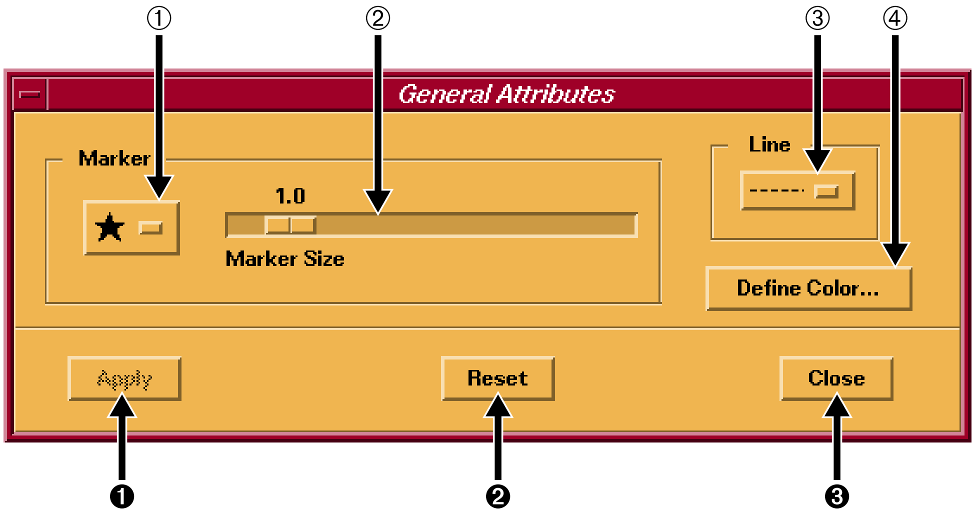
9.4.3 Style



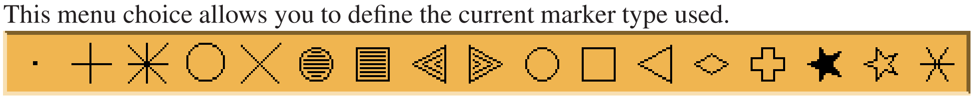
The various buttons invoke the corresponding panel, e.g., Object Attributes... invokes the "Object Attributes" panel.

9.4.4 General Attributes

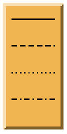
The "General Attributes" panel allow to define attributes like marker type, marker size, line type or color definition for the low level graphics primitives like the lines, the markers the boxes etc...



① This menu choice allows you to define the current marker type used.



② This scale allows you to change the marker scale factor.
③ This menu choice allows you to define the current line style used.

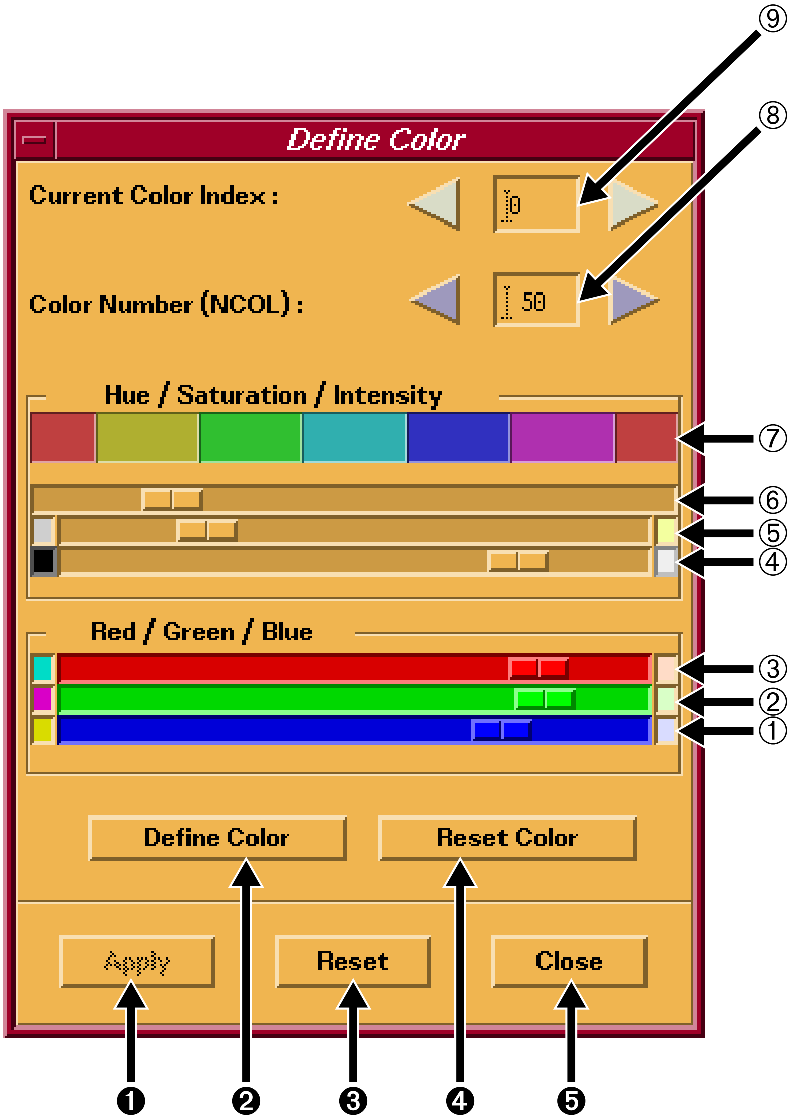


④ This push button opens the “Define Color” panel (see below).

❶ By default the “automatic refresh” is on and as soon as an attribute is changed, the current picture is updated with the new attribute value. But when the “automatic refresh” is off, this button becomes active a should pressed in order to update the current picture with the new attribute value.
❷ This push button allow to reset the default value of all the attributes manageable in this panel.
❸ Close this panel.

Define Color

This panel is invoked when the button number ④ is pressed in the “General Attributes” panel. This panel allows to define a color in `RGB` or `HLS` modes.

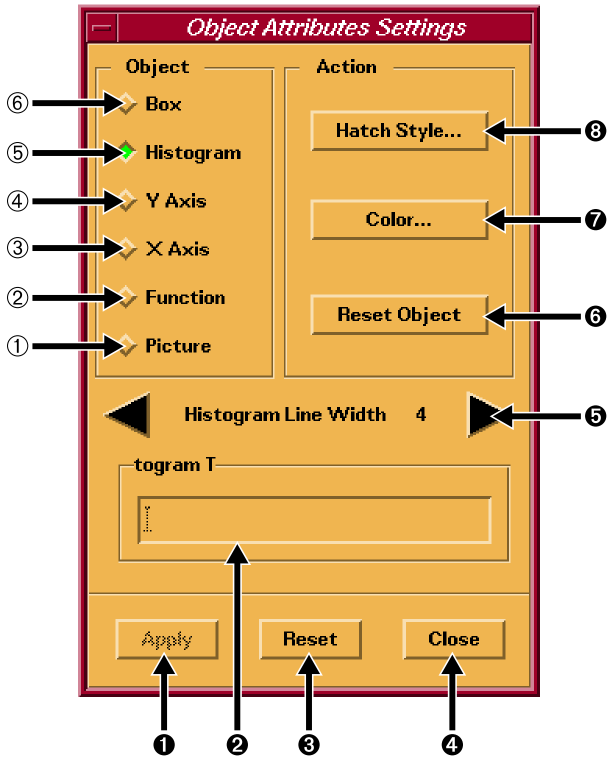


① Percentage of Blue in the color define by the **Current Color index** ⑨.
② Percentage of Blue in the color define by the **Current Color index** ⑨.
③ Percentage of Blue in the color define by the **Current Color index** ⑨.
④ Ligth.
⑤ Saturation
⑥ Hue.
⑦ Hue scale.
⑧ Maximum number of colors.
⑨ Colors index to be changed.

❶ Apply the changes.
❷ Define the color.
❸ Reset the color.
❹ Reset.
❺ Close the panel

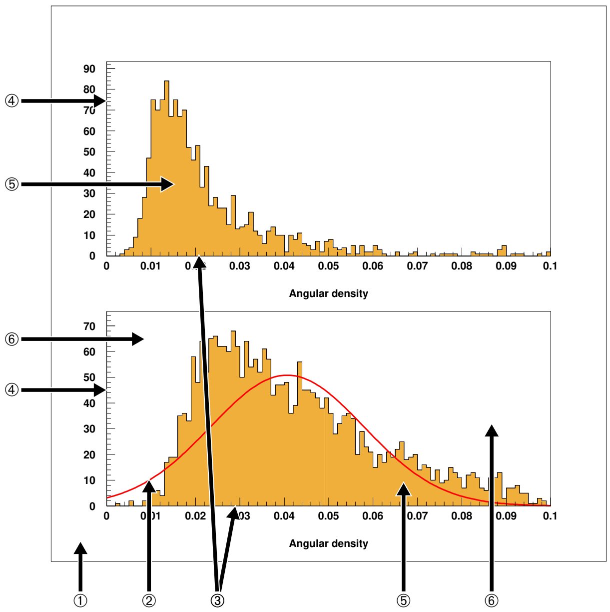
9.4.5 Object Attributes

The “Object Attributes” panel allows to define the graphics attributes of the HPLOT objects managed by PAWsuch as: Histograms, Axis etc.. . On the left part of this panel the type of object can be define via a list of toggle buttons. For example here “Histogram” is selected: all the attributes definable in the panel will be apply on the histograms (histogram color, histogram line width etc...).

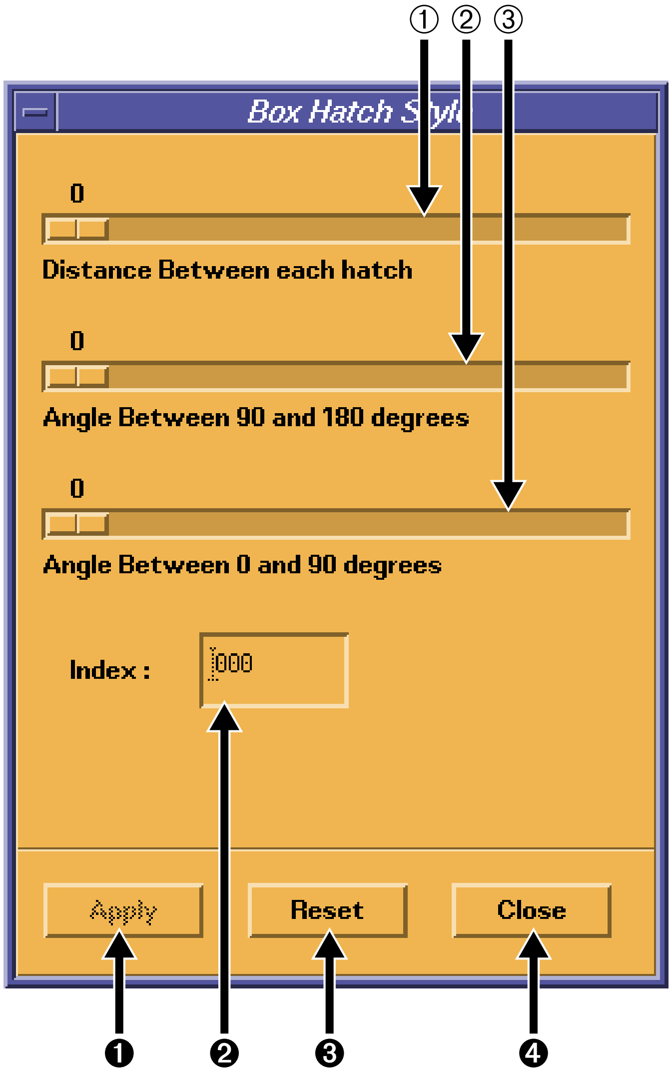


❶ Apply the changes if the “automatic refresh” is not on.
❷ Change the title of the selected object.
❸ Reset all the attributes.
❹ Close this panel
❺ Change the line width of the selected object.
❻ Reset the attributes of the selected object.
❼ Invoke the “Object Colors” panel.
❽ Invoke the “Object Hatch Style” panel.

The zones affected by the buttons ① to ⑥, are shown in the figure below.



Object Hatch Style

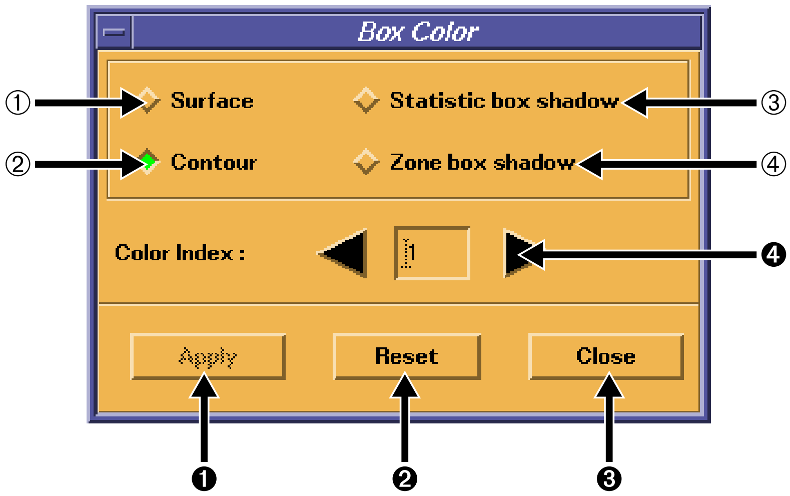


① Define the distance between tow hatches.
② Define the angle of the first set of hatches.
③ Define the angle of the second set of hatches.

❶ Apply
❷ Define the hatches type by number.
❸ Reset the default.
❹ Close this panel.

Object Colors

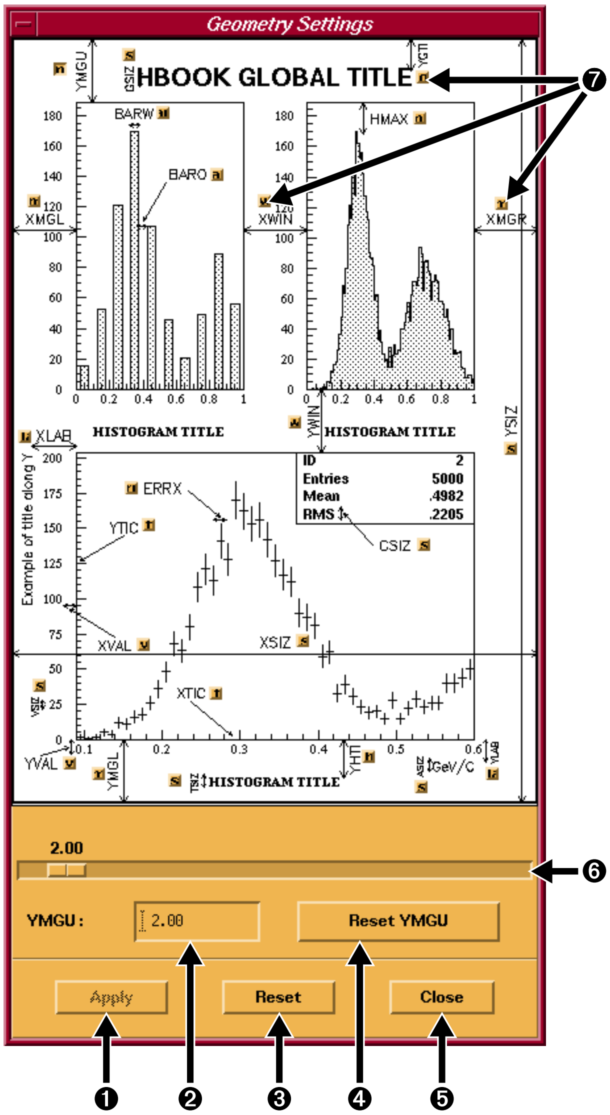
.



① Surface color.
② Contour color.
③ Statistic box shadow color.
④ Zone box shadow color.

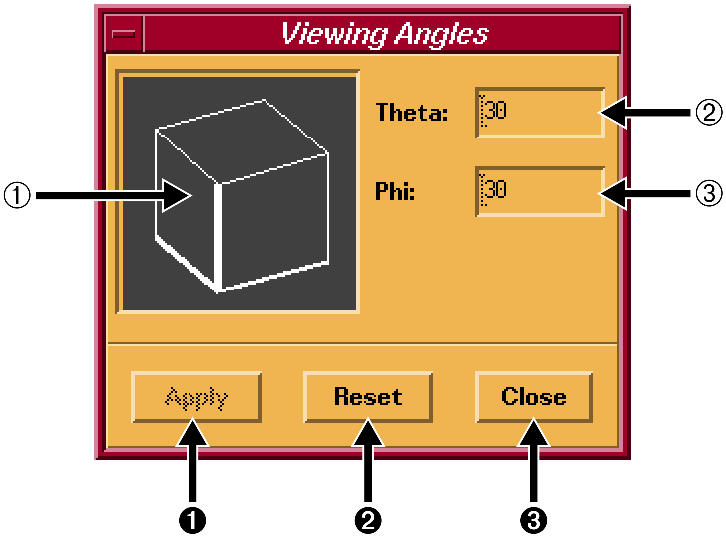
❶ Apply
❷ Reset the color index.
❸ Close the panel.
❹ Define the color index.

9.4.6 Geometry



❶ Apply.
❷ Define the attribute value by number.
❸ Reset the default value.
❹ Name of the current attribute changed.
❺ Close the panel.
❻ Vary continuously the attribute selected.
❼ Select the attribute to be modified.

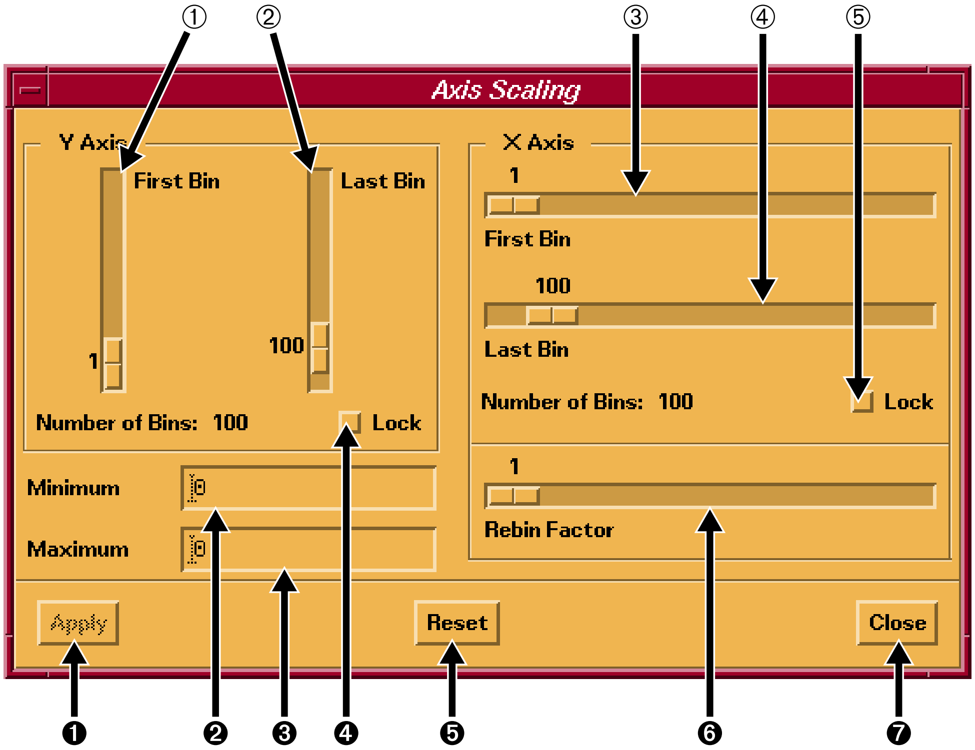
9.4.7 Viewing Angles



① Apply.
② Reset the both angles to 30 degrees.
③ Close the panel.

❶ Rotating cube use to define the angles.
❷ Allow to specify the theta value.
❸ Allow to specify the phi value.

9.4.8 Axis Scaling

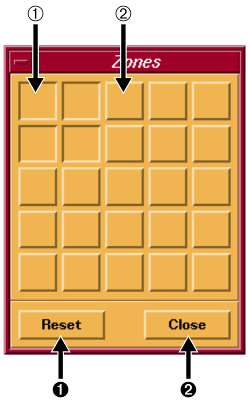


- ① Change the Y first bin value.
- ② Change the Y second bin value.
- ③ Change the X first bin value.
- ④ Change the X first bin value.
- ⑤ Lock the range between the first ans the last X bins.

- ❶ Apply.
- ❷ Set the minimum Z value.
- ❸ Set the maximum Z value.
- ❹ Lock the range between the first ans the last Y bins.
- ❺ Reset the default values.
- ❻ Rebin the 1D histograms.
- ❼ Close the panel.

9.4.9 Zones

This panel is a direct interface to the Zone command.

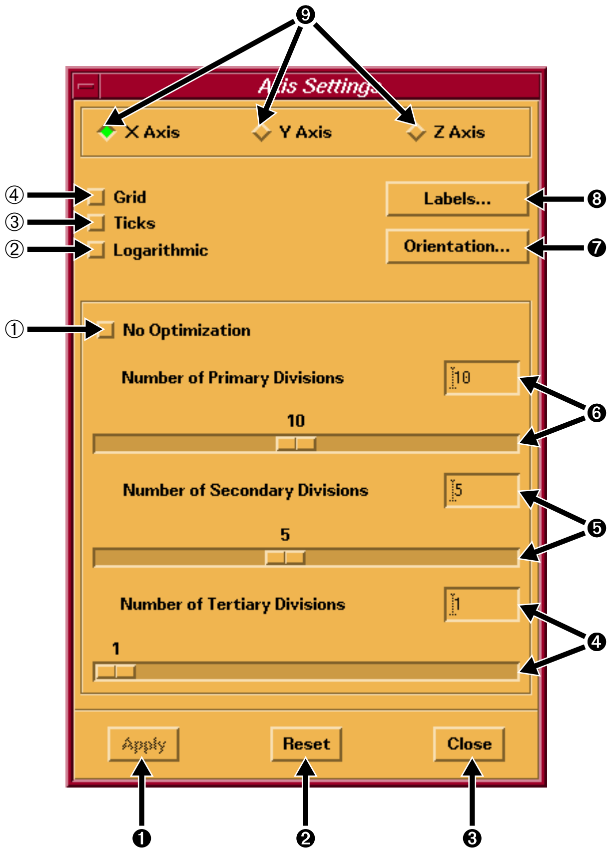


- ① Active zone.
- ② Not active zone.

- ❶ Reset to one zone.
- ❷ Close the panel.

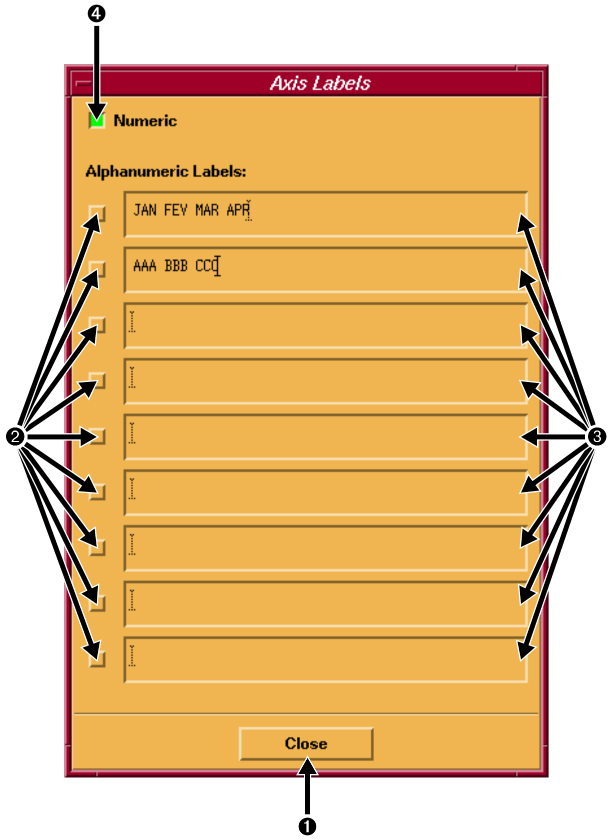
9.4.10 Axis Settings

This panel allows to define the labelling, number of divisions and axis properties (like LOG scale), of the X, Y and Z axis. This is a direct interface to the commands `SET NDVX`, `NDVY` etc ...



① Activate or desactivate the tick marks optimization.
② Activate or desactivate the Log scale.
③ Activate or desactivate the additionnal tick marks on the top and right of the plot.
④ Activate or desactivate the grid drawing.

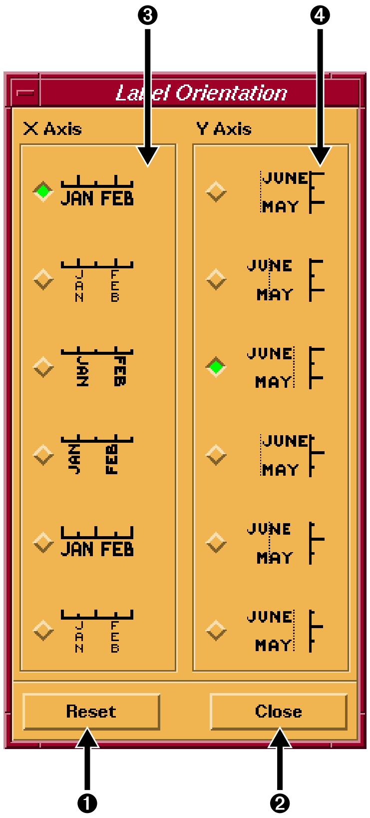
❶ Apply.
❷ Reset the defaults.
❸ Close the panel
❹ Define the tertiary divisions.
❺ Define the secondary divisions.
❻ Define the primary divisions.
❼ Display the “Labels” panel.
❽ Display the “Orientation” panel.
❾ Select on which axis the whole panel will act.



Axis Labels

The panel defines the type of label used.

❶ Close the panel.
❷ Activate one of the alphanumeric list.
❸ Define an alphanumerique list.
❹ The labelling is numeric.

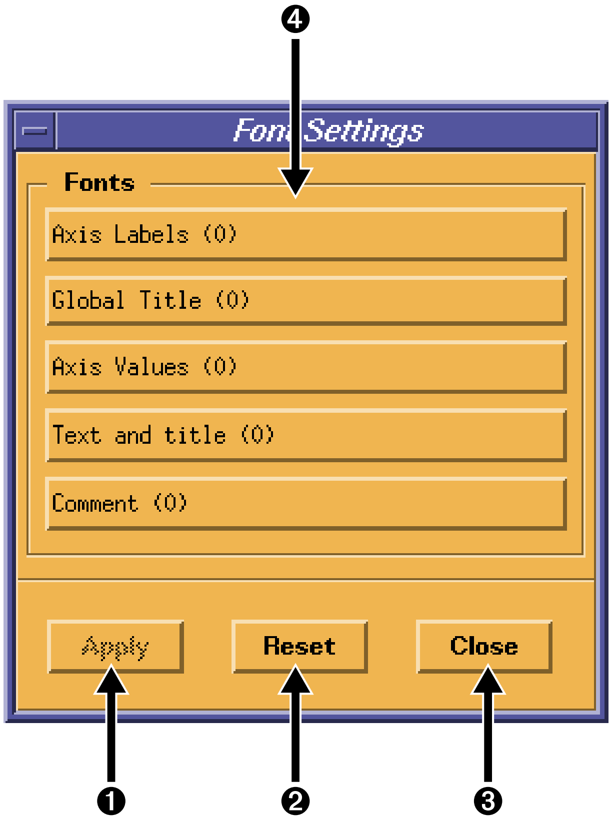


Label Orientation

Defines the labels orientation.

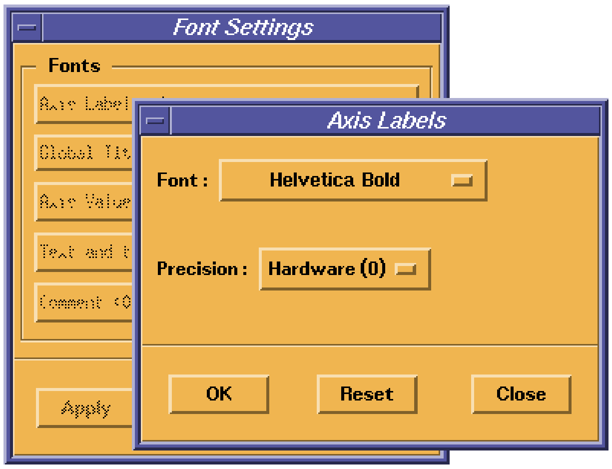
❶ Reset the default orientation.
❷ Close the panel.
❸ Define the X axis labels orientation.
❹ Define the Y axis labels orientation.

9.4.11 Font

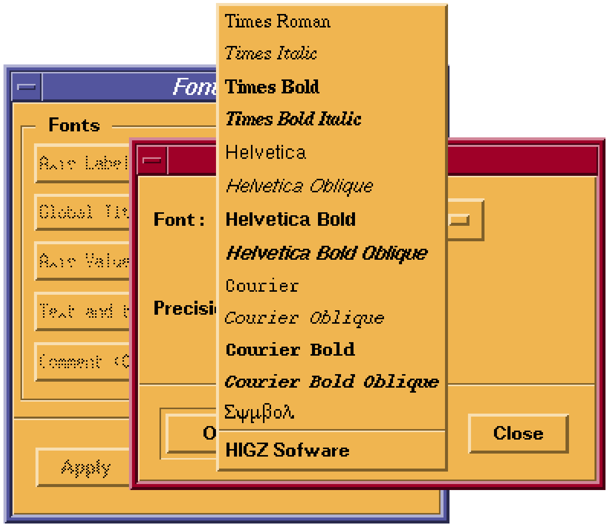


Font selector

❶ Apply.
❷ Reset the default font.
❸ Close the panel.
❹ Select the font for the various type of text.



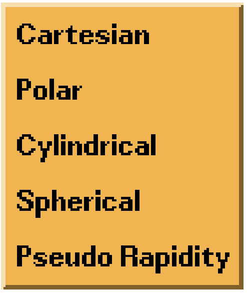
The font settings panel allows to define the font and the precision of a given type of text.



The font may be choosen amoung the standard X11/PostScript fonts.

9.4.12 Coordinate Systems

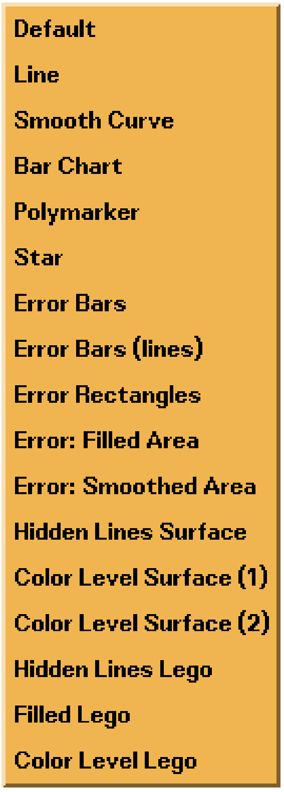
Various coordinate systems can be choosen for surface and lego plots.



Cartesian	All lego and surfaces will be in cartesian coordinates.
Polar	All lego and surfaces will be in polar coordinates.
Cylindrical	All lego and surfaces will be in cylindrical coordinates.
Spherical	All lego and surfaces will be in spherical coordinates.
Pseudo Rapidity	All lego and surfaces will be in pseudo rapidity coordinates.

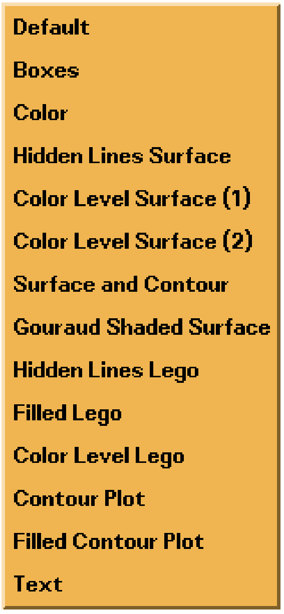
9.4.13 Plot Options

The possible plotting option for 1D histograms available in the **Histogram Style Panel** are the following:



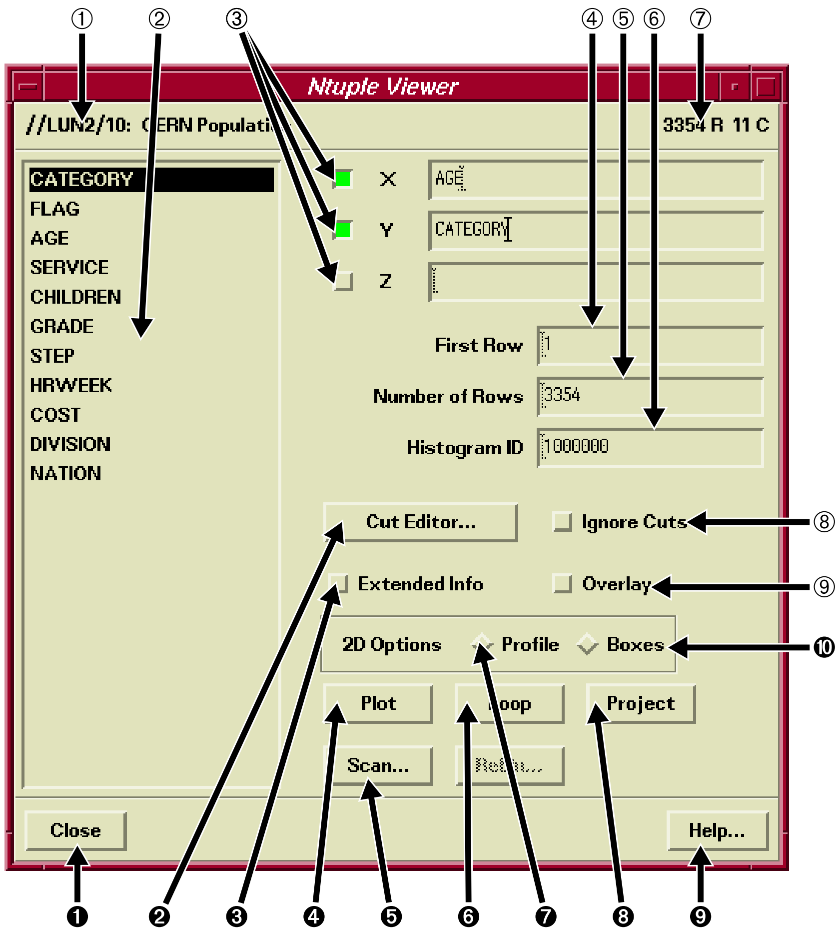
Default	Normal histogram drawing.
Line	Draw the histogram with line.
Smooth Curve	Draw the histogram as a smooth curve.
Bar Chart	Draw the histogram as a bar chart.
Star	Draw the histogram with stars.
Error Bars	Draw the histogram with error bars.
Error Bars (lines)	Draw the histogram with error bars ended with tick marks.
Error Rectangles	Draw the histogram with error rectangles.
Error: Filled Area	Draw the histogram as a filled area.
Error: Smoothed Area	Draw the histogram a a smoothed and filled area.
Hidden Lines Surface	Draw the histogram as a surface.
Color Level Surface (1)	Draw the histogram as a surface.
Color Level Surface (2)	Draw the histogram as a surface.
Hidden Lines Lego	Draw the histogram as a lego.
Filled Lego	Draw the histogram as a lego.
Color Level Lego	Draw the histogram as a lego.

The possible plotting option for 2D histograms available in the **Histogram Style Panel** are the following:



Default	Scatter plot.
Boxes	Boxes plot.
Color	Color plot.
Hidden Lines Surfaces	Surface plot.
Color Level Surface (1)	Surface plot.
Color Level Surface (2)	Surface plot.
Surface and Contour	Surface plot.
Gouraud Shaded Surface	Surface plot.
Hidden Lines Lego	Lego plot.
Filled Lego	Lego plot.
Color Level Lego	Lego plot.
Contour Plot	Line contour plot.
Filled Contour PLot	Filled contour plot.
Text	Text plot.

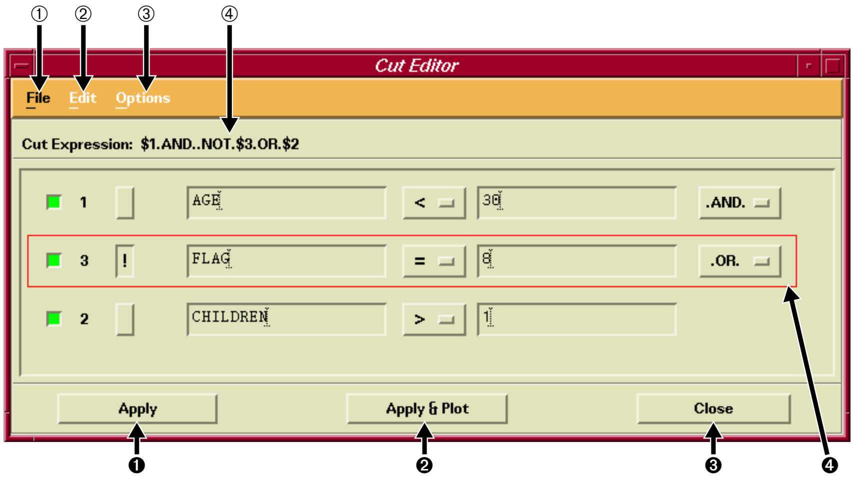
9.5 Ntuple Viewer



① Field showing the current directory and the name of the Ntuple.
② The names of the variables defined for the Ntuple. If you double click on one of the variable names a histogram showing the values of the variable will be plotted.
③ The X, Y and Z fields allow you to define which variables will be used by the Plot and Scan buttons. These fields can be filled in two ways: firstly by typing the name or an expression of a variable; secondly by double-clicking in one of the X, Y or Z fields. In the latter case the field pointed at is filled with the variable highligted in the list of variables.
④ Defines the first row used in the Ntuple when the Plot or Project buttons are pressed.
⑤ Defines the number of rows used (starting from First Row) when the Plot or Project buttons are pressed.
⑥ Defines the histogram identifier used when the Plot or Project buttons are pressed.
⑦ Fields showing the number of rows and columns in the Ntuple.
⑧ A toggle button allowing you to enable/disable the cuts defined with the **Cut Editor**.
⑨ A toggle button, which, when pressed will produce the next plot on top of an already existing one, i.e. without clearing the graphics window.
⑩ If pressed, 2D plots are drawn with boxes.

❶ Close the **Ntuple Viewer**.
❷ Invoke the **Cut Editor**.
❸ When it is pressed, the Ntuple variables types and ranges are also listed.
❹ Produce a plot using all the indications specified on the **Ntuple Viewer** panel.
❺ Invoke the Ntuple Scanner.
❻ Perform the `NTUPLE/LOOP` command.
❼ If pressed, the 2D plots produce profile histograms.
❽ Project the selected variables in the histogram specify in ⑥.
❾ Help on the **Ntuple Viewer**.

9.6 The Cut Editor



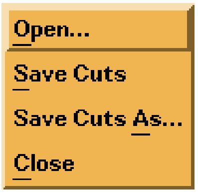
① Invoke the File menu.
② Invoke the Edit menu.
③ Invoke the Options menu.
④ Current cut expression applied.

❶ Apply the cut.
❷ Apply the cut and replot the graph.
❸ Close the cut editor.
❹ Cut definition panel. The current cut is highlighted with a red line. A cut can be activated or deactivated with the toggle button on the left. It can be negated with the push button on the right of the cut number. A “!” appears on this button when the cut is negated. Cuts are defined with the help of the two editable fields and menu choices.

9.6.1 The Cut Editor Menu Bar

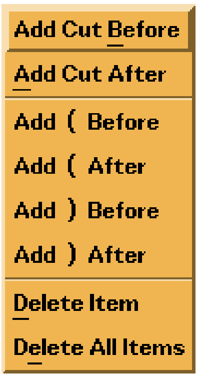
In this section, is describe the full functionality of the pull down menu available in the Menu Bar of the **Cut Editor**.

File



Open	Open a cut file.
Save Cuts	Save the current cuts on disk.
Save Cuts As ...	Save the current cuts on disk in a spsicific file.
Close	Close the panel.

Edit

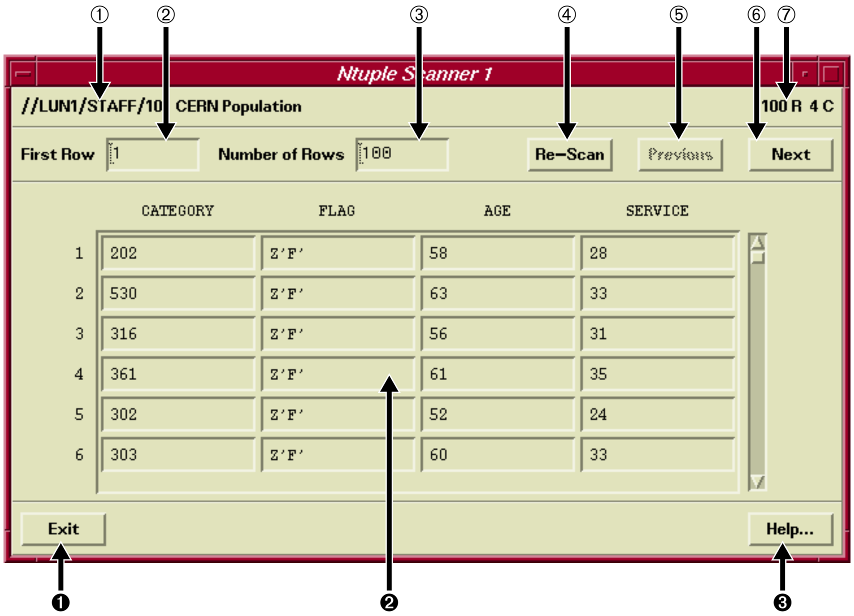


Add Cut Before	Add a cut line before the current cut line.
Add Cut After	Add a cut line after the current cut line.
Add (Before	Add a (line before the current cut line.
Add (After	Add a (line after the current cut line.
Add) Before	Add a) line before the current cut line.
Add) After	Add a) line after the current cut line.
Delete item	Delete the current cut line.
Delete All items	Delete all the cut lines.

Options

Dynamic Mode ...	The current cut can be change dynamically.
Indentation ...	Indente the cut definitions.
Activate all cuts	Activate all cuts
Deactivate All cuts	Deactivate all cuts.

9.6.2 Ntuple Scanner



9.7 KUIP/Motif **Panel Interface**

The PANEL Interface allows to define command sequences which are executed when the corresponding button is pressed (like `STYLE GP` in PAW/X11). The command sequence

```
PANEL 0
PANEL 4.06 'some string'
PANEL 0 D 'This is my first panel' 500x300+500+600
```

creates a panel with 4 rows and 6 columns of buttons. The text 'some string' should be long enough to fit the longest command Sequence which should be put onto one of the buttons. The command "`PANEL 0 D`" defines the title and the window size and coordinates in the form `WxH+X+Y`.

The panels can be edited interactively:

- Clicking with the right mouse button on an empty panel button the user will be asked to give a definition to this button.
- Clicking with the left mouse button on a panel button removes its definition.

The PANEL commands needed to recreate a panel can be saved into a macro file by pressing the "`Save Panel`" button. Panels can be reloaded either by executing the command `PANEL 0 D`' or by pressing the "`Command Panel`" button in the View menu of the **Executive Window** and entering the corresponding file name.

Appendix A: X Window resources

A.1 X resources for PAW++

This is a list of the X resources available to PAW++. Resources control the appearance and behavior of an application.
Users can specify their own values for these resources in the standard X11/Motif way (via their own .Xdefaults file or the system wide /usr/lib/X11/app-defaults/Paw++ file).
Any default values specified by PAW++ are given behind the resource name.

```
Paw++*background:
```

Specify the background color for all windows.

```
Paw++*foreground:
```

Specify the foreground color for all windows.

```
Paw++*kxtermGeometry:                550x550+5+10
```

Geometry of Kxterm, the KUIP terminal emulator (PAW++ **Executive Window**).

```
Paw++*kuipGraphics_shell.geometry:    550x550+585+10
```

Geometry of the Graphics Window(s) (if any).

```
Paw++*kuipBrowser_shell.geometry:     495x511+161+481
```

Geometry of the Browser(s).

```
Paw++*histoStyle_shell.geometry:      599x360+668+631
```

Geometry of the Style Panel.

```
Paw++*ntupleBrowser_shell.geometry:
```

Geometry of the **Ntuple Viewer**.

```
Paw++*XmText*fontList:           *-prestige-medium-r-normal-*-120-*
Paw++*XmTextField*fontList:      *-prestige-medium-r-normal-*-120-*
```

Font used by all text areas.

```
Paw++*kxtermFont:                *-prestige-medium-r-normal-*-120-*
```

Font used by Kxterm (PAW++ **Executive Window**)

```
Paw++*dirlist*fontList:          *-courier-bold-r-normal*-120-*
```

Font used for the icon labels in the browser.

```
Paw++*matrix.fontList:           *-courier-medium-r-normal*-120-*
```

Font used for the Ntuple/Scan matrix (accessible via the **Ntuple Viewer**).

```
Paw++*helpFont:                  *-courier-bold-r-normal*-120-*
```

Font used for help windows.

```
Paw++*fontList:                 *-swiss*742-bold-r-normal-*-120-*
```

Font for the menus, messages and boxes.

```
Paw++*keyboardFocusPolicy:      pointer
```

If “explicit” focus is determined by a mouse or keyboard command. If “pointer” (default), focus is determined by the mouse pointer position.

```
Paw++*doubleClickInterval:      400
```

The time span (in milliseconds) within which two button clicks must occur to be considered a double click rather than two single clicks.

```
Paw++*dirlist*background:
```

Specify the background color for the iconbox part of the browser.

```
Paw++*dirlist*<object>*iconForeground:
```

Specify the foreground color for the icons of type ¡object¿.

```
Paw++*dirlist*<object>*iconBackground:
```

Specify the background color for the icons of type ¡object¿.

```
Paw++*dirlist*<object>*iconLabelForeground:     black
```

Specify the foreground color for the labels of the icons of type ¡object¿.

```
Paw++*dirlist*<object>*iconLabelBackground:     white
```

Specify the background color for the labels of the icons of type ¡object¿. Currently the following different ¡object¿’s are defined:

```
dir     -- directory
1d      -- 1d histograms
2d      -- 2d histograms
ntuple  -- Ntuples
pict    -- Higz pictures
chain   -- Ntuple chains
entry   -- Ntuple chain entries
hbook   -- Hbook files
```

The default iconForeground and iconBackground colors for these objects are:

```
Paw++*dirlist*dir*iconForeground:      blue
Paw++*dirlist*1d*iconForeground:       DarkGoldenrod3
Paw++*dirlist*2d*iconForeground:       DeepPink3
Paw++*dirlist*ntuple*iconForeground:   SteelBlue3
Paw++*dirlist*pict*iconForeground:     green4
Paw++*dirlist*chain*iconForeground:    blue
Paw++*dirlist*entry*iconForeground:    OrangeRed
```

When using a black and white X Server use the following resource settings to make the icons visible:

```
Paw++*dirlist*<object>*iconForeground:          black
Paw++*dirlist*<object>*iconBackground:          white
Paw++*dirlist*<object>*iconLabelBackground:     black
Paw++*dirlist*<object>*iconLabelForeground:     white
```

A.2 X resources for for KUIP/Motif

This is a list of the X resources available to any KUIP/Motif based application (e.g. PAW++). Resources control the appearance and behavior of an application.

Users can specify their own values for these resources in the standard X11/Motif way (via the .Xdefaults file or a file in the /usr/lib/X11/app-defaults directory). One just has to prefix the desired resource by the class name of the application.

To customize PAW++, for instance, all the resources have to be prefixed with `Paw++` or they should be stored in the file /usr/lib/X11/app-defaults/Paw++.

Any default values specified by KUIP are given behind the resource name.

```
*background:
```

Specify the background color for all windows.

```
*foreground:
```

Specify the foreground color for all windows.

```
*kxtermGeometry:                550x550+5+10
```

Geometry of Kxterm, the KUIP terminal emulator (**Executive Window**).

```
*kuipGraphics_shell.geometry:    550x550+585+10
```

Geometry of the graphics window(s) (if any).

```
*kuipBrowser_shell.geometry:     580x450
```

Geometry of the browser(s).

```
*XmText*fontList:               *-helvetica-bold-r-normal*-120-*
*XmTextField*fontList:          *-helvetica-bold-r-normal*-120-*
```

Font used by all text areas.

```
*kxtermFont:
```

Font used by Kxterm (PAW++ **Executive Window**)

```
*dirlist*fontList:
```

Font used for the icon labels in the browser.

```
*helpFont:                      *-courier-bold-r-normal*-120-*
```

Font used for help windows.

```
*fontList:                      *-helvetica-bold-r-normal*-120-*
```

Font for the menus, messages and boxes.

```
*keyboardFocusPolicy:           explicit
```

If "explicit" (default), focus is determined by a mouse or keyboard command. If "pointer" focus is determined by the mouse pointer position.

```
*doubleClickInterval:           250
```

The time span (in milliseconds) within which two button clicks must occur to be considered a double click rather than two single clicks.

```
*dirlist*background:
```

Specify the background color for the iconbox part of the browser.

```
*dirlist*<object>*iconForeground:          black
```

Specify the foreground color for the icons of type ¡object¿.

```
*dirlist*<object>*iconBackground:          white
```

Specify the background color for the icons of type ¡object¿.

```
*dirlist*<object>*iconLabelForeground:     black
```

Specify the foreground color for the labels of the icons of type ¡object¿.

```
*dirlist*<object>*iconLabelBackground:     white
```

Specify the background color for the labels of the icons of type ¡object¿.

```
*zoomEffect:                     True
```

Turn zoom effect on or off when going up and down directories in the browser.

```
*zoomSpeed:                      10
```

Specify speed of zoom effect in the browser.
Currently the following different ¡object¿'s are defined:

```
Cmd        -- Command
InvCmd     -- Deactivated command
Menu       -- Menu tree
MacFile    -- Macro File
RwFile     -- Read-write file
RoFile     -- Readonly file
NoFile     -- No access file
ExFile     -- Executable file
DirFile    -- Directory
DirUpFile  -- Up directory (..)
```

When using a black and white X Server use the following resource settings to make the icons visible:

```
*dirlist*<object>*iconForeground:          black
*dirlist*<object>*iconBackground:          white
*dirlist*<object>*iconLabelBackground:     black
*dirlist*<object>*iconLabelForeground:     white
```

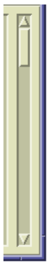
Appendix B: Editing keys in the Input Pad

"C-b" means holding down the Control key and pressing the b key.
"M-" stands for the Meta key and "A-" for the Alt key.

```
C-b:               backward character
A-b:               backward word
M-b:               backward word
Shift A-b:         backward word, extend selection
Shift M-b:         backward word, extend selection
A-[:               backward paragraph
M-[:               backward paragraph
Shift A-[:         backward paragraph, extend selection
Shift M-[:         backward paragraph, extend selection
A-<:               beginning of file
M-<:               beginning of file
C-a:               beginning of line
Shift C-a:         beginning of line, extend selection
C-osfInsert:       copy to clipboard
Shift osfDelete:   cut to clipboard
Shift osfInsert:   paste from clipboard
Alt->:             end of file
M->:               end of file
C-e:               end of line
Shift C-e:         end of line, extend selection
C-f:               forward character
A-]:               forward paragraph
M-]:               forward paragraph
Shift A-]:         forward paragraph, extend selection
Shift M-]:         forward paragraph, extend selection
C-A-f:             forward word
C-M-f:             forward word
C-d:               kill next character
A-BS:              kill previous word
M-BS:              kill previous word
C-w:               kill region
C-y:               yank back last thing killed
C-k:               kill to end of line
C-u:               kill line
A-DEL:             kill to start of line
M-DEL:             kill to start of line
C-o:               newline and backup
C-j:               newline and indent
C-n:               get next command, in hold mode: next line
C-osfLeft:         page left
C-osfRight:        page right
C-p:               get previous command, in hold mode: previous line
C-g:               process cancel
C-l:               redraw display
C-osfDown:         next page
C-osfUp:           previous page
C-SPC:             set mark here
C-c:               send kill signal to application
C-h:               toggle hold button of pad containing input focus
F8:                re-execute last executed command
Shift F8:          put last executed command in input pad
Shift-TAB:         change input focus
```

Appendix C: The Motif user interface tools

C.1 Scale



A scale can be moved with the scale button, or with the two arrows (top and bottom). It is usualy linked to some quantity which may vary continuously.

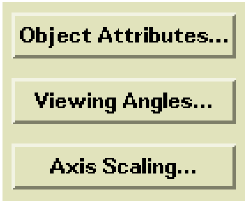
C.2 Buttons

Various kind of buttons are available in Motif: Toggle, Push and Selectionbuttons.

C.2.1 Toggle Buttons

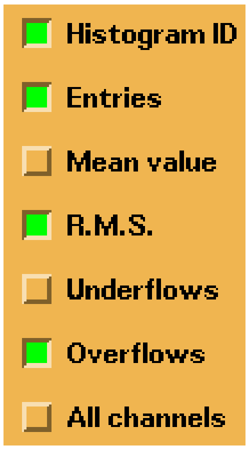
The toggle buttons are usually used for Yes/No choices. In a serie of toggle button, only one can be push.

C.2.2 Push Buttons



Push buttons are usualy used to perform a specific action. Very often they open an other panel.

C.2.3 Selection Buttons



Selection buttons are used to select an option or a special mode. They are not linked together like the toggle buttons and they can be on independently from the state of the others.

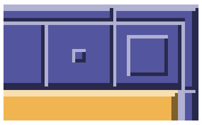
C.3 Paned Window

Paned window separate a window in several part. Each part is resizable but the global size stay the same: when a part grow an other one reduce.

C.4 Window manager buttons

This tools a present on all the Motif windows.

A double click on this button closes the window. a simple click display a pull down menu. The content of the this menu depends on the window manager used.



These two buttons allows respectively to iconise and to enlarge a window to the maximum size possible on the screen.

Bibliography

[1] CERN. *COMIS – Compilation and Interpretation System, nProgram Library L210*, January 1994.

[2] CN/ASD Group. *HBOOK Users Guide (Version 4.21), nProgram Library Y250*. CERN, January 1994.

[3] CN/ASD Group. *HIGZ/HPLOT Users Guide, nProgram Library Q120 and Y251*. CERN, 1993.

[4] CN/ASD Group. *KUIP – Kit for a User Interface Package, nProgram library I202*. CERN, January 1994.

[5] CN/ASD Group. *MINUIT – Users Guide, nProgram Library D506*. CERN, 1993.

[6] CN/ASD Group and J. Zoll/ECP. *ZEBRA Users Guide, nProgram Library Q100*. CERN, 1993.

[7] R. Brun and H. Renshall. *HPLOT users guide, nProgram Library Y251*. CERN, 1990.

[8] R. Bock et al. *HIGZ Users Guide, nProgram Library Q120*. CERN, 1991.

[9] F. James. *Interpretation of the errors on parameters as given by MINUIT, nSupplement to "CERN Program Library Long writeup D506"*. CERN, 1978.

[10] F. James. Determining the statistical Significance of experimental Results. Technical Report DD/81/02 and CERN Report 81–03, CERN, 1981.

[11] W. T. Eadie, D. Drijard, F. James, M. Roos, and B. Sadoulet. *Statistical Methods in Experimental Physics*. North-Holland, 1971.

[12] Sun Microsystems. *Network File System Version 2*. Sun Microsystems, 1987.

Index