PAW - TN/14
DELPHI 86-95 PROG-60
12 November 1986

PAMPA

A PANEL and MENU

MANAGEMENT PACKAGE

F.Carena, J.C.Marin, L.Pape

USER MANUAL

CONTENTS :

1. INTRODUCTION

The present package consists of routines to perform the management of :

- **Windows**, i.e. areas on the screen that can be assigned to a given task.
- **Events**, i.e. actions originating from the keyboard, the mouse, etc.
- **Menus**, i.e. lists of items that are user selectable.
- **Panels**, i.e. areas on the screen for the presentation, selection and editing of text.

The spirit of this user interface reflects very clearly the Smalltalk approach, as implemented, for instance, on the Apple Macintosh or the Apollo. An attempt was made to keep the greatest possible part of the package transportable to different terminals and operating systems. However, the part of the package which is directly related to the display will unavoidably depend on the terminal and on the operating system used. This is the case for (part of) the window and the event management. The menu and the panel management are coded in pure Fortran77 and are thus fully transportable. The main purpose of the present note is to define a Fortran77 binding for a user interface, which is a problem with existing packages as no standardization of the calling sequences exists among the various manufacturors. Hence the calling programs need to be modified in order to transport them to the different operating systems. The present package has been implemented on VAX using SMG in the window management. Future developments will include the implementation of the same calling sequences on top of GKS for graphics terminals, the Apollo GMR and the Macintosh toolbox, for instance. This will allow the calling programs to become transportable to these various environments.
It should be mentioned that, although the windows are in fact graphical objects, it is possible to implement a (somewhat downgraded) version on alphanumeric terminals (the SMG implementation is an example). The minimum requirement is that the terminal/operating system allows full page addressing and cursor control.

1.1 Structure of the paper

The basic concepts used in the interface package are described in chapter 2.
The window management routines follow rather closely the proposal of Bozzoli/Mornacchi [1] window system implemented by B.Wessels [2]. The corresponding calling sequences are specified in chapter 3. In particular, the window management package allows to use structured windows, i.e. a new window can be assigned to the screen or to another already existing window from which it will depend.
The menu and panel management routines handle the display of menus and panels at a very low

level. The contents and attributes are saved in a menu/panel data base but without hierarchical structure. The corresponding calling sequences are described in chapter 4. A menu or panel is displayed inside a window which is created via the window management routines above. In fact, a menu is then handled as a subwindow (but the latter is not noticed by the user) which allows to let the window manager handle most of the difficult (computer dependent) problems.
A limited hierarchy has been included to simplify the user interface for e.g. pull-down menus and is defined in chapter 5. A true menu structure is, however, provided only via higher level packages, like the ICI [3] or the COLA package [4] which are based on a command processor supporting hierarchy and developed in the context of the Physics Analysis Workstation project PAW [5].
An editor is provided with the package, as it is needed for the panel management routines. It is normally not called itself by the user who would rather set up a panel. Its description is nevertheless given in chapter 6, in case it would be useful for a different application.
Hints are given in chapter 7 for the implementation of the window and the event management routines in different environments than the ones mentioned above.

1.2 Convention for the description of the calling sequences

In order to simplify the description of the calling sequences in the subsequent chapters, the following conventions have been adopted:

- Arguments are either input or output, never both.

- Input parameters are not underlined. Output parameters are underlined in the description of the calling sequences.

- All parameters, except positions, are of type INTEGER*4, LOGICAL or CHARACTER, the latter being flagged by an asterisc (*) following their name in the calling sequences. Positions are given as real numbers and defined in terms of normalized coordinates (ranging from 0 to 1) with the origin in the top left corner of the screen or window to which they belong.

References :

[1] W.Bozzoli, G.Mornacchi: Window System requirements, Internal Report of DD/OC group, Window System 22 Jan 1986.
[2] B.Wessels: Window System Manager, User's Guide and Implementation Notes, Internal DD/OC Report, Window 27 August 1986.
[3] J.C.Marin, L.Pape: ICI, an Interactive Command Interface, DELPHI note in preparation.
[4] J.C.Marin, L.Pape: COLA Interactive Command Language, Version for MacIntosh, User Manual, Internal Report DELPHI 86-69 PROG-54.
[5] R.K.Bock, R.Brun, P.Giacomelli, J.C.Marin, L.Pape, A.Petrilli, P.Zanarini: Command Processor, Overall Specification, Internal Report PAW-TN/06, 6 May 1986.

2. OVERVIEW OF THE BASIC CONCEPTS

The present package provides tools for the presentation of and the interaction with information to be displayed on a terminal, ranging from a simple alpha-numeric one to a Personal Computer or a Workstation. This user interface is strongly inspired from the Smalltalk approach in particular the user interface implemented on the Macintosh or the Apollo. The problem with the existing implementations is that each manufacturor has defined his own calling sequences and that the application code thus needs to be modified in order to be transported from one environment to another. It is the main purpose of the present package to define a low level user interface which provides a sufficient functionality for the higher level packages to manipulate easily windows, panels and menus. At the same time, the window manager part contains all the functions which require computer or display dependent code. Hence the calling programs (and the menu/panel package) can remain transportable to various computers and displays. The minimal requirement is that the terminal has full screen addressing capability and that the cursor can be user controlled.

2.1 Windows and window structures

A **window** is a rectangular area on the screen which is reserved for a given task. It can be used for the direct I/O of an application program, for the display of menus, for interactive editing, for graphics display, etc.

The root window is called the "**screen**" and covers the whole surface of the display. The window manager allows to create windows at any position within this screen or within another pre-existing window, from which it will depend. This dependence provides a means to **structure the windows** such that some operations, like the removal of a window from the screen, affect not only the window itself but also all its sub-windows. The possibility to structure windows is considered as an essential feature in this package, although it has not been implemented in many existing window management systems. For example, the panel and menu packages, described below, rely heavily on this feature. The structural relations between the windows and the window attributes are held in a "window data base".

When an instruction is given to display a window, the latter becomes the front-most window on the screen. In case the area of the new window overlaps with a previous window, it will hide the overlapping surface of the old window. If the extension of a (sub-)window exceeds the boundaries of its superior window, it will be only partially visible or not at all.

Various actions can be performed or recorded when they occur within the area of a given window, like the writing or reading of text, the drawing of graphics objects, the drawing or reading of a cursor, the reading of a mouse position. All such basic operations are supported by the window management package. The present package does not save the information displayed in a window, unless the underlying manufacturors' product does so. Therefore, the user may need to remember himself any text or graphics he wants to display, in the case that the window contents need to be refreshed.

2.2 User interface for interaction

Ideally, the interaction with the user is based on the usage of a **mouse** (or a similar device, like a joystick or a track ball) to select a position or activate an object. The position of the mouse indicated on the screen is entirely dependent on the movement of the "physical mouse" and cannot be modified from the program. When the user enters a **mouse click** in a text string, the corresponding position is indicated on the screen by a "**cursor**" (normally represented by a different symbol than the mouse itself). A cursor is defined within its corresponding window, unlike the mouse which is defined with respect to the whole screen. Hence, every window can have its cursor and the cursor cannot move outside the boundaries of its window. Its displacements are software controlled. Mouse clicks to provide a position and possibly text input from the keyboard are the usual ways to interact with the presentation package.

For terminals where no provision is made for a real mouse, a "cursor" can be used instead under the control of special keys on the keyboard (with up, down, left and right arrows). The latter may, however, prove difficult to control in some terminal/computer configurations where the interaction with the machine is slow. Hence the package provides a different mode of interaction, called the "no mouse" mode, in which four operations (associated to four keys) allow to scan through the logical relations of the window structure:

- **up**, goes back to the superior window
- **down**, goes down into the first dependent window
- **next**, goes into the next window at the same level and depending from the same superior window
- **previous**, goes back to the previous window at the same level.

These operations are equivalent to the displacement of a mouse without clicking. An "**Enter**" operation is provided to simulate a mouse click. Such a mechanism allows to make large displacements of a cursor rather quickly when no hardware mouse is available. However, in some cases (like in editing text) the stepwise movement of the cursor by means of the up, down, left and right arrows remains indispensable for small displacements.

The exact definition of the keys is given in Appendix 1.

2.3 Window layout

Windows will certainly appear somewhat different on different terminals. However, the following basic layout and functionality should be respected (see figure 2.1).

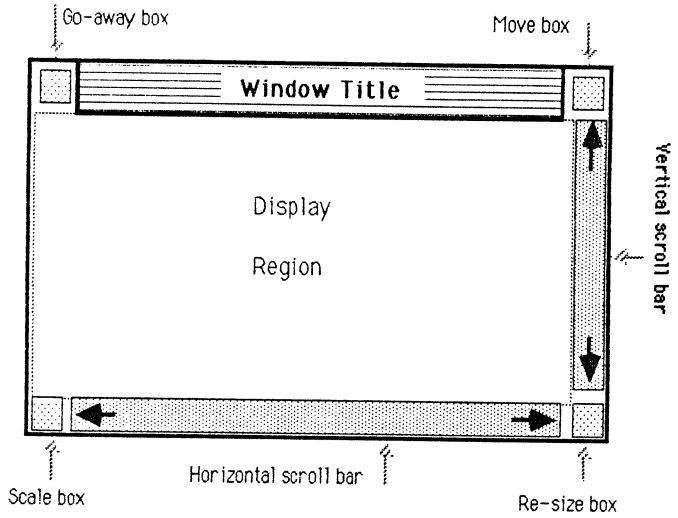

Figure 2.1. Layout of a window and functional boxes

- **Title bar**, region in which the title is displayed. In case the title does not exist or is blank, the title region is not displayed.

- **Go-away box**, when selected, the window is removed from the screen. This also removes all its sub-windows, if any.

- **Move box**, when selected, allows to move the window to another position on the screen. The sub-windows are moved together with the window.

- **Re-size box**, when selected, allows to increase or decrease the size of the window. The information displayed in the screen as well as the subwindows remain in the same absolute position and may become partly or totally invisible if the window size is decreased.

- **Scale box**, also allows to increase or decrease the size of a window, but the window contents are scaled together with the window. The latter is, however, not possible with non-graphics display of text.

- **Horizontal scroll bar**, allows to scan horizontally through the window contents in case they exceed the information displayed in the window.

- **Vertical scroll bar**, allows to scan vertically through the window contents in case they exceed the information displayed in the window.

- **Display region** is the usable region for the display of text and graphics.

- **Rectangular boundary** surrounding the whole window.

All the above functions, except the display region, are optional. In particular, a window could consist of a display region which is not even surrounded by a rectangle (this is useful in some cases for subwindows, see the panels). They are called the "**window type definition**".

2.4 Definition of coordinates

Within the screen and the windows, a **coordinate system** is defined with the origin in the top left corner of the display region. The positions are always expressed as "**normalized coordinates**" (real numbers ranging from 0 to 1), with the horizontal coordinate running from left to right and the vertical coordinate from top to bottom (see figure 2.2). The choice of normalized coordinates was made to ease the transport between displays with very different sizes and/or resolution. For instance, a window which is assigned to the screen of a small terminal, can be re-assigned to a window of the same size if the application is run on a large area terminal.

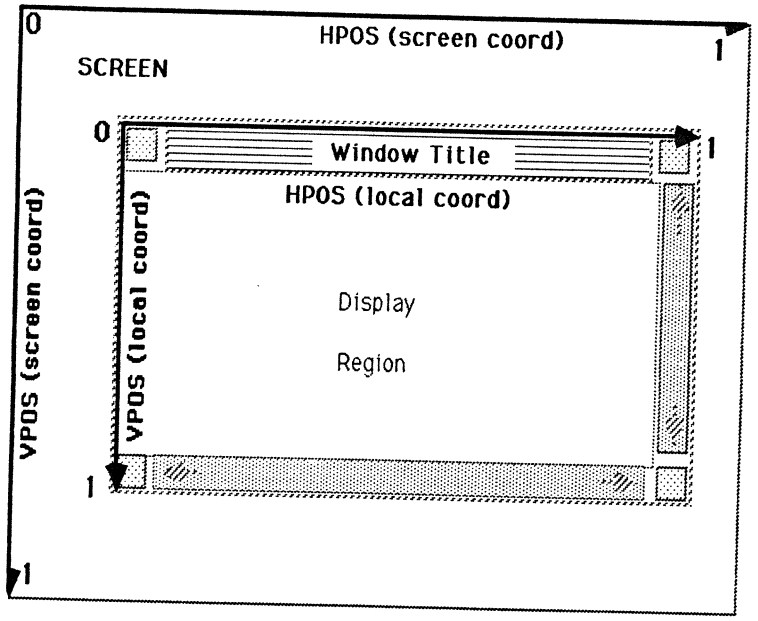

Figure 2.2. Definition of the screen and local window coordinate systems.

Every window has, attached to it, a coordinate system of its own, called "**local coordinate system**", in which the coordinates are normalized. Routines are provided to make the transformations between local and screen coordinates. To ease the manipulation of text on alphanumeric terminals, routines are also provided to convert between normalised local coordinates and columns/lines.

Note that the coordinates are attached to the whole window, not to its display region. Hence to compute the size of a window, it is necessary to include the space used by the functions associated to the window, i.e. the bars and the boxes if they exist.

2.5 Rendition attributes of a window

The rendition attributes of a window define the way in which text and graphics will appear in the window. They contain, for example:

- the background colour
- the writing colour
- the character font, size and spacing
- the line style and width
- the fill pattern for surfaces

Default values for the rendition attributes are provided by the package, but the user can modify the values at will for any window or even an individual object within a window. The default attributes are handled as follows:

- the default of the package is set for the screen at initialisation time,
- it can be overwritten by the user before any window is created,
- when a new window is created, its default attributes are taken from the attributes of the screen at the time of creation,
- the attributes of any window can be modified by the user at any time,
- objects within windows are drawn with the attributes of the window at the creation of the object.

The window attributes are subdivided into three categories (see section 3.1):

- the video attributes, applicable to both text and graphics objects
- the text attributes, for text only
- the graphics attributes, for graphics only

2.6 Functionality of the Window Management routines

The functionality of the window management routines is described in detail in chapter 3. The main functions are as follows:

- The definition of the **window** including its position, size, window type definition and rendition attributes are held in the window data base, independent of it being displayed on the terminal. Routines are provided for the definition of the layout and position of a window and installation of the window in the window data base (CWIND), the display of the window on the screen (CWSHOW), the removal of the window from the screen (CWREMO) or from the window data base (CWSCRA).
- Routines are provided to **convert** from local to screen **coordinates** and reversely and to transform local coordinates to and from column/lines in the window.
- The **mouse** position can be read out in local or in screen coordinates. Also, the **cursor** can be positioned at a given location in a window, can be moved or its current position can be read.
- **Character strings** can be written, read or edited (added or deleted) in a window. Their rendition attributes can be explicitely defined and modified (normal, bold, reverse, blink, underline).
- **Graphics objects** can be drawn, like straight lines, rectangles, ellipses, polylines and polymarkers. Their attributes can be specified by the user (e.g. highlighting). Some of these cannot, of course, be implemented on alpha-numeric terminals.
- A routine allows to signal the occurence of **events**, i.e. user or external interventions like a mouse click, keyboard input, etc.
- Windows can be **interactively moved** within their superior window or **re-sized**. In this case the corresponding routines take temporarily control over the events.

2.7 Menus and panels

A **panel** is a window in which various areas with text can be freely set-up by the user. They are defined at initialisation time and presented on the screen to prompt an interactive dialog. Within each area of a panel the text can be one of the following kinds (see figure 2.3).

- **Fixed text**, used only for display purposes (warning, error message, etc.). No interaction is possible with fixed text.
- **Editable text**, also for display purposes, but the user can interactively modify its contents. This can also be used to simply enter text in order to answer a question.
- **Selectable text**, can be used a choice among several options is required. The user selects interactively the case of his choice.
- **Buttons** are similar to selectable text but they are meant to define the exit mode at the end of the interaction with the panel (e.g. OK, NO, CANCEL).

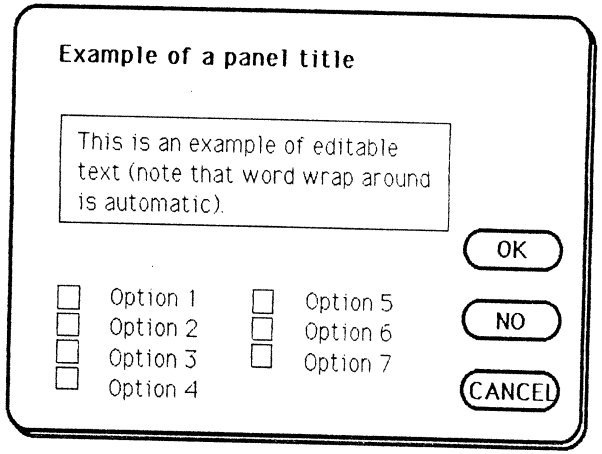

Figure 2.3 Example of a panel.

A **menu** is a particular case of a panel in which short sentences (menu items) are listed below each other (vertical menu) or next to each other on the same line (horizontal menus), as shown in figure 2.4. On presentation of a menu, the user is requested to select the menu item of his choice, i.e. menu items appear as selectable text. In some cases, e.g. to define or modify interactively the contents of the menu items, an item may also be presented as editable text.

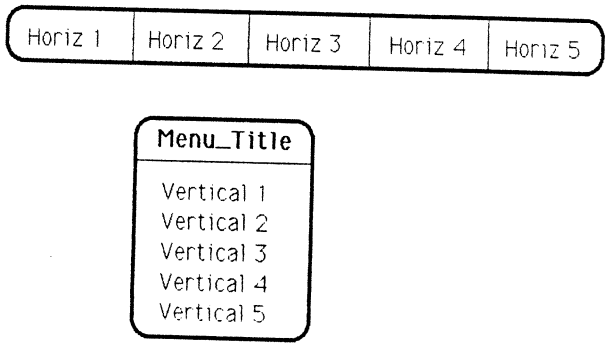

Figure 2.4 Examples of menus, one being displayed horizontally, the other vertically

As panels and menus are so closely related objects, a single package is provided to handle both of them. They are distinguished only by a few actions, as described below in the functionality of the routines.

2.7 Functionality of the Panel and Menu Management routines

A set of routines handle panel and menu display at the lowest possible level. Indeed, the panels and menus are treated as independent objects and no real hierarchical structure is supported by the package. A limited hierarchy is included to simplify the user interface in some cases. However, a true menu structure is provided only via higher level packages like the ICI or COLA packages.

Menus and panels can be in a local buffer, in their data base and displayed on the screen. Their definition and modification is always done in the local buffer. Also the display operates from the information in the local buffer. The data base is provided only for later re-use. More precisely, the definition and usage of the panels/menus follows a sequence of steps defined as follows:

- The definition of the panel/menu layout, their entries and possible attributes are defined first. Such a transaction starts by calling CMOPEN. The corresponding information is only stored in a local buffer, but it can be used for display (calling CMSHOW). It will, however, be lost as soon as another panel/menu is defined, unless it is explicitely saved in the data base.

- The panel/menu contents and attributes, held in the local buffer, can be saved in a menu data base by calling CMCLOS.

- The data base information can be restored in the local buffer for further definition or editing by calling CMREAC. After these operations, the contents of the local buffer can again be displayed. They can also be installed in the data base by calling CMCLOS, thus overwriting the previous definition.

- After a panel/menu has been opened (CMOPEN or CMREAC), its entries can be defined by calling CMENU or CMPAN. This allows to enter the text corresponding to every entry of the panel/menu. Alternatively, it is possible to specify only which entries have to be foreseen and the areas they occupy in the panel. The actual contents can then be defined at a later stage by CMFILL.

- The display of a panel/menu is entirely under user control and, for instance, **none of the above operations affect the state of the screen.** A menu or panel is displayed (CMSHOW) inside a window which is created via the window management routines. In fact, a panel/menu is then handled as a subwindow (but the latter is normally not noticed by the user) which allows to let the window manager handle most of the difficult (in particular display dependent) problems. The panel/menu to be displayed with CMSHOW is expected to be residing in the local buffer.

- In some cases a panel/menu which is being displayed needs to be partially updated. To avoid redrawing it completely, a routine CMUPD is provided to make the update of individual entries from the information in the local buffer.

- A panel/menu can be erased from the screen by calling CMREMO, which removes it from the screen leaving the local buffer and the data base unaffected.

- Panels are a convenient way to hold a dialog with a user. A routine CMDIAL is provided which, when called after CMSHOW, takes the events in charge (calling CEVENT) and returns only after the user has made a sensible reply. It specifies the entry which was affected and the exit mode from the panel. It can only be used for panels, when they are used to enter into a dialog and normally expect the user to reply before starting anything else. The routine CMDIAL should not be used with menus.

- An alternative approach is that the user keeps the events under his own control (CEVENT) and calls CMLOC to identify the entry in which a mouse click occured. This routine is useable for both panels and menus, but is of real interest only in the latter case.

3. WINDOW MANAGER ROUTINES

The basic concepts of the window management routines have been dealt with in chapter 2. The present chapter defines the calling sequences of the routines.

3.1 Global definitions of variables

Most variables keep the same meaning and abbreviation through all the calls. They are defined here. The others occur only occasionally and are described with the calling sequence.

- window identifier :

IDWIND integer number identifying a window uniquely. The value of this number is defined by the package, not by the user and is zero for the screen.

IDSUPW identifier of the window which is superior (or mother) to the current window in the window tree structure.

- position and size : positions are always given in **normalized local coordinates**

HPOS Horizontal position of a point in a window

VPOS vertical position of a point in a window

HSIZE horizontal size of a window or box

VSIZE vertical size of a window or box

- window type definition :

WTYPE character string with the window type definition

B = window is surrounded by a border line

T = window with title

A = window with go-away box

M = window with move box

R = window with resizing box

S = window with scale box

V = window with vertical scroll bar

H = window with horizontal scroll bar

several of them can be used simultaneously

as soon as one of them is set, the first one is also set automatically

- rendition atttributes :

IVATTR video attributes applicable to both text and graphics :

(1) = 0, normal

= 1, reverse (= highlight)

= 2, flicker

(the logical OR of the above values is permitted)

(2) = background colour index

(3) = foreground colour index

(the actual colours are obtained from a colour table)

ITATTR text attributes

(1) = 0, normal plain text

= 1, bold

= 2, italic

= 4, underline

= 8, outline

= 16, shadow

= 32, superscript

= 64, subscript

(the logical OR of the above values is permitted whenever it makes sense)

(2) character font

(3) horizontal character size

(4) vertical character size

(5) character spacing (includes the horizontal character size)

(6) spacing between lines of text

(7) orientation of text drawing

IGATTR graphics attributes

(1) line style

(2) line width

(3) fill pattern

(4) polymarker index (symbol obtained from a table)

3.2 Window definition routines

The window package saves the window attributes and their relationships in a "window data base". The window contents, be it text or graphics, written in the window are not saved by the package and can thus not be retrieved later. If needed, the user should save them himself. This may be necessary, e.g. to restore the contents of a window after a CWSIZE which can cause some of the contents to be lost.

CWINIT ()

Needs to be called once for the initialisation of the package. It defines the default attributes for the screen as IVATTR(1)=ITATTR(1)=0, the other settings being dependent on the display characteristics. The attributes can be modified by calling the attribute setting routines CWATT or CWSETA described below. It also sets the default window type WTYPE = 'B', i.e. a window with a border line and no other functionality.

CWIND (IDSUPW,HPOS,YPOS,HSIZE,YSIZE,IDWIND)

Creates a new window as subwindow of the window IDSUPW (the screen if IDSUPW=0). The new window is identified by a number IDWIND which is defined by the package and which the user should remember to address the window in later calls. The corresponding window record is installed in the window data base at this stage and a link is created from the record of the mother window IDSUPW to the record of the sub-window. The identifier IDWIND is used in the following calls in the same way for a mother window and for sub-windows. The default attributes of the window are set as the current attributes of the screen. The position and size, given in the call, do not need to include the window title nor border lines, but should include the space for the other functions (go-away, move, etc) if they are used. In case the window boundaries exceed the area of the superior window (except the screen), the new window is refused and IDWIND=0 is returned. Further down windows will thus be attached to the screen. Note that CWIND does not cause the window to be displayed.

CWSHOW (IDWIND)

Draws a window on the screen after it has been installed in the window data base (see CWIND). This window is made the front-most one. If the window area exceeds the boundaries of the screen only part of the window will be visible.

CWREMO (IDWIND)

Removes the window and its subwindows from the screen and makes the next window the front most one. This does not affect the window definition in the data base.

CWSCRA (IDWIND)

Scratches the window and its subwindow records from the window database and removes them from the screen. Makes the next window on the display the front most one.

3.3 Modification of the window type and attributes

The window type, title and attributes can be modified at any moment after the call to CWIND. If the following routines are called before CWSHOW the modification is made in the window data base only. If they are called after the CWSHOW, the modification is also reflected on the display. If the window identifier in the calls is zero, the screen attributes will be modified. Hence, the default will be changed for all windows created after this.

CWTYPE (IDWIND,WTYPE*)

Redefines the window type definition WTYPE for the window IDWIND. If the type is blank, the window corresponds to an area on the screen without border line nor anything else. It will, however, hide any other window parts underneath. If IDWIND =0, the default type for the screen is modified (but this does not affect its layout on the screen) and hence the default for all windows that are created after this.

CWTITL (IDWIND,TITLE*)

Defines the title contents for the window IDWIND. This call also forces the window type to include 'T', i.e. to have a title area, without the need to call CWTYPE. Note that a window with title has automatically a border line (see definition of WTYPE).

CWATT (IDWIND,IVATTR,ITATTR,IGATTR)

Sets the video, text and graphics attributes of the window IDWIND. The three arrays IVATTR, ITATTR and IGATTR have to be defined. If an element of an array corresponds to a negative number, the existing definition remains unchanged. If IDWIND =0, the screen attributes are modified and hence the default for all windows that are created after this.

CWSETA (IDWIND,ATTR*,IARR,IVALUE)

As it may be rather lengthy in some cases to define all the attribute arrays, an alternative to CWATT is provided in which attributes can be changed individually. The character string ATTR defines which attribute is addressed:

ATTR = 'V' for video attributes
= 'T' for text attributes
= 'G' for graphics attributes

IARR specifies the array element and IVALUE its new value. If IDWIND =0, the screen attributes

are modified and hence the default for all windows that are created after this.

CWUPDA (IDWIND,HPOS,VPOS,HSIZE,VSIZE,OPTION*)

Updates the video attributes within a rectangular area in the window IDWIND but leaves the contents of all it sub-windows unaffected. The character string OPTION is defined as:

OPTION = 'N' for normal
= 'R' for reverse video
= 'F' for flicker
the combination 'RF' is also accepted

CWCLR (IDWIND,HPOS,VPOS,HSIZE,VSIZE)

Clears the contents within a rectangular area in the window IDWIND and leaves all it sub-windows unaffected. The sub-windows remain unchanged in the data base and on the screen.

CWPOS (IDWIND,HPOS,VPOS,HSIZE,VSIZE,SCALE)

Modifies the position and size of the window in the data base, as well as on the screen. This routine performs a function which is similar to the CWMOVE/CWSIZE (see behind), but does not require interaction with the user. The logical variable SCALE determines how the window contents and its subwindows are affected if the window size is modified. If SCALE=.false., it leaves the window contents and all its sub-windows at the same place, except if some exceed partially or totally the new window boundaries. In this case, part of the subwindows becomes invisible. If SCALE= .true. the window contents and the dimension of all its subwindows are rescaled by the same factor as the window itself. With the alpha-numeric mode of the package, the latter can of course only be done partially (e.g. when size is reduced, the text disappears).

3.4 Window enquiry routines

CWGSUP (IDWIND,IDSUPW)

Returns the identifier of the window which is superior to IDWIND. IDSUPW=0 (zero) means that the superior window is the screen. If IDWIND is zero, the superior window does not exist and the value returned is IDSUPW=-1.

CWGDWN (IDWIND,NDWSKI,NDWMAX,NDWS,IDWS)

Returns a list of subwindows which are one level below the window IDWIND. NDWSKI specifies how many of the first window identifiers should be skipped. NDWMAX is the maximum number of identifiers the user can accept. The actual number of down windows is returned in NDWS and the list of identifiers in the array IDWS, up to a maximum of NDWMAX. If the total number exceeds NDWMAX, the routine can be called in a loop, using NDWSKI to select the next set of identifiers.

CWGSTA (IDWIND,ISTAT)

Returns the status of the window IDWIND in the integer variable ISTAT:

ISTAT = 0, window not in the data base
= 1, window exists but not displayed
= 2, window displayed but not under the mouse
= 3, is the window under the mouse

CWGTYP (IDWIND,WTYPE*)

Returns the type definition of the window IDWIND in the character string WTYPE. If WTYPE is blank, the window has no boder line nor anything else.

CWGTIT (IDWIND,TITLE*)

Returns the title of the window IDWIND in the character string TITLE. If TITLE is blank, the window has no title.

CWGPOS (IDWIND,HPOS,VPOS,HSIZE,VSIZE)

Returns the position and size of the window IDWIND defined in the local coordinate frame of the superior window IDSUPW.

CWGATT (IDWIND,IVATTR,ITATTR,IGATTR)

Returns the arrays of video, text and graphics attributes for the window IDWIND.

CWGETA (IDWIND,ATTR*,IARR,IVALUE)

Returns the value of a single attribute of the window IDWIND as an integer IVALUE. The attribute is specified by the character ATTR, as defined for CWSETA, and the array index IARR.

3.5 Character drawing

CWRITE (IDWIND,TEXT*,HPOS,VPOS,IERASE)

Outputs the character string TEXT in the window according to the rendition attributes of the window at the time of the call. If the text runs outside the window then it is cut and continues on the next line. The cursor remains positioned at the end of the text.

IERAS = 0, no erase

= 1, erase the existing text before writing

The text written in the window is not saved by the package and can thus not be retrieved later. If needed, the user should store it himself. This may be necessary, e.g. to restore the contents of a window after a CWSIZE.

The reading of character strings from the keyboard is described in section 3.10.

CWDELC (IDWIND,NCHAR,HPOS,VPOS)

Deletes NCHAR characters from the text, starting at the given position (local window coordinates). This routine does not handle word wrap around, hence the end of the line from which characters are deleted may remain blank.

CWADDC (IDWIND,TEXT*,HPOS,VPOS)

Adds the character string TEXT, starting at the given position (local window coordinates). This routine does not handle word wrap around, hence the end of the line to which characters are added may disappear if it extends beyond the window boundary.

3.6 Graphics drawing

Note that the graphics drawn in a window is not saved by the package and can thus not be retrieved later. If needed, the user should store it himself. This may be necessary, e.g. to restore the contents of a window after a CWSIZE.

CWPOLY (IDWIND,NPT,HPOSA,VPOSA)

Draws a poly-line joining the NPT points specified in the arrays HPOSA and VPOSA, the coordinates being defined in the local window frame. If some coordinates lie outside the window they are not drawn. The poly-line is drawn according to the current attributes of the window IDWIND.

CWLINE (IDWIND,HPOS1,VPOS1,HPOS2,VPOS2)

Draws a line joining the position (HPOS1,VPOS1) to (HPOS2,VPOS2), the coordinates being defined in the local window frame. If the coordinates lie outside the window, no line is drawn. The line is drawn according to the current attributes of the window IDWIND.

CWRECT (IDWIND,HPOS,VPOS,HSIZE,VSIZE)

Draws a rectangle defined by the position of the top-left corner in local window coordinates and the size of the horizontal and vertical sides of the rectangle. If a corner lies outside the window boundaries, no rectangle is drawn. The rectangle is drawn according to the current attributes of the window IDWIND.

CWRND (IDWIND,HPOS,VPOS,HSIZE,VSIZE,RADIUS)

Draws a "rounded rectangle" defined by the position of the top-left corner in local window coordinates, the size of the horizontal and vertical sides of the rectangle and the radius of the circle used to round the corners. If a corner lies outside the window boundaries, no rectangle is drawn. The rectangle is drawn according to the current attributes of the window IDWIND.

CWOVAL (IDWIND,HPOS,VPOS,HSIZE,VSIZE)

Draws an oval (ellips) inscribed in the rectangle defined by the position of the top-left corner in local window coordinates and the size of the horizontal and vertical sides of the rectangle. If a corner lies outside the window boundaries, no rectangle is drawn. The rectangle is drawn according to the current attributes of the window IDWIND.

CWMARK (IDWIND,NPT,HPOSA,VPOSA)

Draws the marker, defined by the window graphics attribute IGATTR(4), at the NPT points specified in the arrays HPOSA and VPOSA, the coordinates being defined in the local window frame. If some coordinates lie outside the window the markers are not drawn.

3.7 Mouse position routines

CWGET (IDWIND)

Returns the window under the mouse and, if several windows are superposed, the front one under the mouse.

CWGETS (HPOS,VPOS)

Returns the position of the mouse in Screen coordinates.

CWGETL (IDWIND,HPOS,VPOS)

Returns the current window identifier and position of the mouse in local coordinates (see also CWLOC in section 3.10).

3.8 Cursor control routines

CWCURS (IDWIND,FLAG)

Shows or hides the cursor in the window IDWIND according to whether the value of FLAG is .true. or .false..

CWCHOM (IDWIND)

Homes the cursor to the top left hand corner of the window.

CWCMOV (IDWIND,HPOS,VPOS)

Moves the cursor to the position HPOS,VPOS defined in local window coordinates. If the position lies outside the window the cursor is left where it was.

CWCREL (IDWIND,DELTAH,DELTAV)

Displaces the cursor by an amount relative to its current position. If the new position lies outside the window the cursor is left where it was.

CWCPOS (IDWIND,HPOS,VPOS)

Returns the position of the cursor in local window coordinates.

3.9 Conversion between position and column/line

CWLTOC (IDWIND,HPOS,VPOS,ICOL,ILIN)

Converts a position given in local window coordinates in the window IDWIND to its corresponding column and line number relative to the same window.

CWCTOL (IDWIND,ICOL,ILIN,HPOS,VPOS)

Converts a given column and line number relative to the window IDWIND to a position given in local window coordinates in the same window.

CWLTOS (IDWIND,HPOSL,VPOSL,HPOSS,VPOSS)

Converts a position given in local window coordinates in the window IDWIND to the corresponding position in screen coordinates.

CWSTOL (IDWIND,HPOSS,VPOSS,HPOSL,VPOSL)

Converts a position relative to the screen to a position given in local window coordinates in the window IDWIND.

3.10 Event Manager and interaction routines

To intercept user actions on the keyboard or from the mouse, the following routine is provided :

LOGICAL FUNCTION **CEVENT (MASK,WHAT*,WHERE*,REGION*)**

Needs to be called continuously in a loop to check whether an "event" has occured. When called, the routines keeps control until an event, acceptable to the conditions set by the mask, has occured. This allows, e.g. to put the running process in hibernation until something happens, hence freeing the CPU for other processes.

MASK allows to accept only certain types of events:

0 = all events accepted

1 = keyboard actions

2 = mouse clicks (1st click)

4 = end of selection (2nd click)

(the logical OR of the above values is allowed)

WHAT, character string defining the type of event :

'KEY' = keyboard input

'MOUSE' = mouse click (1st click)

'SELECT' = end of selection (2nd click)

WHERE, character string defining in which area the action occured :

'WINDOW' = normal window or sub-window

'SCREEN' = in the screen but outside any window

'MESSAGE' = asynchronous message

'KEYBOARD' if WHAT = 'KEY' only

REGION, character string telling in which part of a window the event took place :

'DISPLAY' = display region

'TITLE' = title bar

'AWAY' = go-away box

"MOVE' = move box

'SIZE' = re-size box

'SCALE'= scale box

'HSCROLL' = horizontal scrol bar

'VSCROLL' = vertical scrol bar

'KEYBOARD' if WHAT = 'KEY' only

CWLOC (IDWIND,HPOS,VPOS)

Returns the current window identifier and position of the mouse in local coordinates as it was at the time of the last mouse click (1st or 2nd click).

CWGETC (CHAR*)

Reads the character CHAR input from the keyboard. This should only be called after CEVENT signals a keyboard input.

CWMOVE (IDWIND)

After a move operation has been requested, e.g. by selecting the move box of the window, a call to CWMOVE allows to define the new position interactively. CWMOVE takes control over the events and waits for the next mouse click (WHAT='SELECT') which determines the new position of the move box in the screen. It displaces the window and all its sub-windows and returns control to the calling program. After a moving operation the window and some of its sub-windows may remain only partly visible on the screen. The position of the window is updated in the window data base.

CWSIZE (IDWIND,SCALE)

After a re-sizing operation has been requested, e.g. by selecting the size box of the window, a call to CWSIZE allows to define the new window size interactively. CWSIZE takes control over the events and waits for the next mouse click (WHAT='SELECT') which determines the new position of the size box in the screen. It increases or decreases the dimension of the window. The logical variable SCALE determines how the window contents and its subwindows are affected. If SCALE=.false., it leaves the window contents and all its sub-windows at the same place, except if some exceed partially or totally the new window boundaries. In this case, part of the subwindows becomes invisible. If SCALE= .true. the window contents and the dimension of all its subwindows are rescaled by the same factor as the window itself. With the alpha-numeric mode of the package, the latter can of course only be done partially (e.g. when size is reduced, the text disappears). Then, the routine returns control to the calling program. The position and size of the window are updated in the data base.

3.11 Example: creation of windows

```
      CHARACTER*32 TEXT
      INTEGER IDW1,IDW2
      REAL  HPOS,VPOS,HSIZE,VSIZE


*     Initialise the window manager package
      call CWINIT
*
*     Now the screen is created and the default attributes defined. Set the windows to include a
*        border line and a title as default.
      call CWTYPE (0,'BT')
*
*     Create a window (occupying 80% horizontally and vertically) in the screen and define its
*        title
      HPOS = 0.1
      VPOS = 0.1
      HSIZE= 0.9
      VSIZE= 0.9
      call CWIND (0,HPOS,VPOS,HSIZE,VSIZE,IDW1)
      call CWTITL (IDW1,'Title of top window')
*
*     Write text on the 3rd line of the window (this is where routines like CWCTOL are useful)
      call CWCTOL (IDW1,1,3,HPOS,VPOS)
      TEXT = 'This is some text in the window'
      call CWRITE (IDW1,TEXT,HPOS,VPOS,0)
*
*     Now create a sub-window with 30 columns and 10 lines. Assume it should be without
*        title (note that the change of attributes is in the screen, not in window IDW1) and
*        show everything.
      call CWCTOL (IDW1,5,5,HPOS,VPOS)
      call CWCTOL (IDW1,30,10,HSIZE,VSIZE)
      call CWTYPE (0,'B')
      call CWIND (IDW1,HPOS,VPOS,HSIZE,VSIZE,IDW2)

      CALL CWSHOW (IDW1)
```

3.12 Example: handling the events

```
      LOGICAL CEVENT
      CHARACTER*8 WHAT,WHERE,REGION,TYPE
      INTEGER MASK,IDWIND,IDMP,IENTRY
      REAL HPOS,VPOS


*     Set-up all window definitions, display, etc
      ...

*     Define the mask and start the loop over the events
   10 MASK = 0
      if (CEVENT (MASK,WHAT,WHERE,REGION)) then          ; wait for the event to happen

*     First test on "what" has happened
         if (WHAT.eq."KEY') then                         ; keyboard event
         ... handle keyboard input

         else if (WHAT.eq.'MOUSE') then                  ; mouse event

*           if a mouse event, test "where" the event happened
            if (WHERE.eq.'WINDOW') then                  ; inside a window
               call CWLOC (IDWIND,HPOS,VPOS)             ; get window identifier
               call CMLOC (IDWIND,IDMP,TYPE,IENTRY)      ; get panel/menu identifier
               if (IDMP.EQ.0) then                       ; normal window, not menu/pan
                  call CWSHOW (IDWIND)                   ; put window in front
```

```
*                   if inside a normal window, then test on the "region" to find how to react
                    if (REGION.eq.'DISPLAY') then              ; in the display region
                        call ... (IDWIND,HPOS,VPOS)            ; user should handle this
                    else if (REGION.eq.'AWAY') then            ; in go-away region
                        call CWREMO (IDWIND)                   ; remove the window
                    else if (REGION.eq.'MOVE') then            ; in the move box
                        call CWMOVE (IDWIND)                   ; move the window
*                       this starts interaction and expects a 2nd mouse click
                    else if (REGION.eq.'SIZE') then            ; in the size box
*                       this starts interaction and expects a 2nd mouse click
                        call CWSIZE (IDWIND,.FALSE.)           ; resize the window
                    else if (REGION.eq.'HSCROLL') then         ; in a scroll bar
                        ... ; handle the scrolling
                    end if                                     ; for normal windows
*               if mouse click is in a panel or menu
                else if (IDMP.gt.0) then                       ;in panel or menu
                    if (TYPE.eq.'P') then                      ; in panel
                        call PANELS (IDWIND,HPOS,VPOS)         ; user routine for panels
                    else if (TYPE.eq.'V') then                 ; in vertical menu
                        call MENUS (IDWIND,HPOS,VPOS)          ; user routine for vert menus
                    end if                                     ; for panels/menus
                end if                                         ; for windows or panel/menus
            end if                                             ; for where happened
        end if                                                 ; for what happened
    end if                                                     ; for event loop
    go to       10
```

4. MENU AND PANEL MANAGER ROUTINES

The basic concepts used in the panel and menu package are described in chapter 2. The present chapter defines the calling sequences of the routines.

4.1 Global definition of variables

Defined here are the variables which appear in several calling sequences and keep the same meaning throughout the panel/menu package.

IDWIND window identifier

IDMP identifier for a panel or a menu. This name is used for operations which work for both panels and menus.
IDMEN menu identifier
IDPAN panel identifier

IENTRY number of an entry in a panel or menu
TYPE type of panel/menu
= 'P' panel
= 'V' vertical menu
= 'H' horizontal menu
= 'D' pull-down menu
= 'O' overlapping menus
the combinations 'VD', 'HD', 'VO', 'HO' are allowed
KIND specifies the kind of an entry :
'F', for Fixed text
'E' for editable text
'S' for selectable text
'B' for text inside a button

HPOS horizontal position in normalized local coordinates
VPOS vertical position in normalized local coordinates

4.2 Global Panel and Menu transactions

A first phase consists in defining the layout and the contents of the menus and panels, and possibly saving them into the panel/menu data base. The complete specification of the panels or menus can be made during this phase. However, the definition of their contents can still be completed or edited at a later stage (called delayed operations).
During the definition phase, initiated by CMOPEN, the information of the current panel/menu is kept in a local buffer. It is installed in the data base only by calling CMCLOS. Delayed operations can still be performed on the panel/menu entries after their definition has been closed (i.e. installed in the data base). They are initiated by a call to CMREAC, which restores the information from the data base into the local buffer. The information defined in the delayed operations is saved in the local buffer only and can be installed in the data base by calling CMCLOS, like for the primary definition.
The display of the panels/menus is performed from the information in the local buffer, not directly from the data base. Hence, if the information does not need to be re-used later on, the user does not need to save it in the data base (i.e. the call to CMCLOS is not made).
The definition of panels and menus is independent of the window in which they are displayed and, in fact, the same panel/menu can be simultaneously displayed in several windows.
It should be noted that none of the following operations affect the information displayed on the screen.

CMINIT ( )

A call to CMINIT needs to be made once to initialise the panel/menu package.

CMOPEN (TITLE*,IDMP)

Opens the transactions to create a new panel/menu. If "TITLE" is an empty character string, no title will appear on the menu display, but the title region may exist. The routine returns an integer number IDMP which serves as a unique identifier for the panel/menu. The CMOPEN call is normally followed by the definition of the panel/menu contents (see next section). The definition is only saved locally at this stage and is not stored in the menu data base until the "CLOSE" instuction is given. If a panel/menu was previously opened without being closed, it is lost when a new "open" instruction is made.

CMCLOS ( )

Causes the currently defined panel or menu information to be installed in the panel/menu data base. After a "close" no further transactions can be made with any panel/menu, unless an existing panel/menu is reactivated by performing a call to CMREAC or a new one is opened with CMOPEN. Note that the close does not make the information to be displayed and leaves the

information which is currently displayed on the screen unaffected. Even if the definition is not closed, the display of the panel/menu is possible by calling CMSHOW, but its definition is lost as soon as a new menu (or panel) is OPENed.

CMREAC (IDMP)

Reactivates the panel or menu IDMP by restoring its information from the data base into the local buffer, hence making it available for further operations. When the local buffer already contains the information for the same IDMP, the information is nevertheless copied from the data base. This allows to re-initialize information in the local buffer which may have been modified by delayed operations. The result of the delayed operations can be saved in the data base by calling CMCLOS, hence overwriting the existent definition.

CMRENA (TITLE*,IDMP)

Requests the current panel/menu definition held in the local buffer to be renamed. A new title is associated to the renamed panel/menu. The routine returns a new identifier IDMP. This allows e.g. to save temporarily modified panel/menu information in the data base without overwriting the original definition.

CMSCRA (IDMP)

Removes a menu or panel from the menu data base. It can thus no longer be displayed unless it is re-created. If the scratched panel/menu is currently displayed on the screen, this remains unaffected.

4.3 Definition of the panel/menu contents

The operations described below allow to define or modify the entries in the panels and menus and to set or modify their attributes. They can be interleaved with the definition calls of the previous paragraph, i.e. after a CMOPEN, or they can be used during delayed operations, i.e. after a CMREAC (or CMSHOW). Note that none of them is allowed after the CMCLOS has been performed.

None of the operations below modifies the contents of the screen nor of the data base, but they act only on the information in the local buffer.

To define the entries of **menus** only :

CMENU (IPOS,TEXT*,IENTRY)

Appends a menu item behind the existing (previously defined) item with number IPOS in the menu. For IPOS=0, the item is entered in front of all the other ones. Appending to the last entry may be done by using IPOS=999 or some sufficiently large number. The routine returns the actual entry number IENTRY. Note that, when an entry is inserted between previously defined entries (or deleted), the old entry numbers following it are redefined (the entry number reflects the actual position in the menu). The character string "TEXT" specifying the menu item to be displayed (maximum 255 characters) can be entered here, although a delayed definition (see CMFILL) is equally possible. The call to CMENU automatically informs the package that the current definition is related to a menu, not to a panel.

For **panels**, not only the text, but also its position in the panel needs to be specified :

CMPAN (ICOL,ILIN,NCOL,NLIN,TEXT*,IENTRY)

The area inside the panel is defined by the starting column and line numbers (ICOL, ILIN) and by the number of columns and lines (NCOL, NLIN) to be reserved. A unique panel identifier is returned, like for the menus. A character string "TEXT" of maximum 255 characters can be specified, but a delayed definition (see CMFILL) can also be made. The call to CMPAN automatically informs the package that the current definition is related to a panel, not to a menu.

CMFILL (IENTRY,TEXT*)

Overwrites the current definition of the text associated to the panel or menu entry IENTRY

CMDEL (IENTRY)

Deletes the entry IENTRY from a panel or menu.

CMSIZE (NCOL,NLIN)

Defines the size of a whole menu or panel in number of columns and lines. If the routine is not called or if it is called with NCOL=NLIN=0 (zero) the size is computed automatically to include all areas defined by CMENU or CMPAN.

CMTYPE (TYPE*)

This routine controls the display of the menu/panel items according to the value of the variable TYPE defined in Section 4.1. The routine allows to set the value or modify a previously defined one. As it only affects the local buffer, this allows to display the same menu in different forms for different windows.

If the routine is not called, the default is that all menus are displayed vertically (TYPE='V'). A menu of type 'D' (pull-down) is defaulted as 'HD' and the type 'O' (overlapping) is defaulted as 'VO'.

CMKIND (IENTRY,KIND*,OPTION*)

Is used to specify the kind of an entry in a panel or menu. If not called, the default kind is :

- fixed text for panels
- selectable text for menus

Within a panel/menu definition a call with IENTRY=0 (zero) overwrites the current default definition for all subsequent entries of this panel/menu. If IENTRY corresponds to a defined entry number, the specification of the kind only applies to that entry. The character string "KIND" specifies the kind of entry (or leaves it unchanged if KIND is blank) and OPTION can take the following values:

OPTION = 'N' if not surrounded by anything
= 'B' if to be surrounded by a rectangular boundary
= 'L' if followed by a line

Note that a line occupies the same space in the menu/panel as a text string of one line and appears thus differently from giving the text attribute ITATTR(1)=4 in CMATT.

CMATT (IENTRY,IVATTR,ITATTR)

Allows to change the video or text attributes of a given entry of a panel or menu . If IENTRY=0 (zero), all subsequent entries of the given panel/menu are affected. The variables IVATTR and ITATTR are described in the window manager. If the routine is not called, the attributes are carried over from the screen, like any other default attributes.

CMSETA (IENTRY,ATTR*,IARR,IVALUE)

Allows to redefine the attributes individually. If IENTRY=0 (zero), all subsequent entries of the given panel/menu are affected. The character string ATTR defines which attribute is addressed:

ATTR = 'V' for video attributes
= 'T' for text atttributes

IARR specifies the array element and IVALUE its new value.

CMACT (IENTRY,FLAG)

Activates or de-activates the menu item according to whether the FLAG is set to .true. or .false.. A de-activated panel or menu entry cannot be selected by the user.

CMTIC (IENTRY,FLAG)

Puts a tick mark or removes it next to the menu item according to whether the FLAG is set to .true. or .false..

CMHIGH (IENTRY,FLAG)

Highlights or removes highlighting for the menu item according to whether the FLAG is set to .true. or .false..

CMFLIC (IENTRY,FLAG)

Causes the entry IENTRY to flicker or not depending on whether the logical variable FLAG is .true. or .false..

4.4 Display of panels and menus

CMSHOW (IDMP,IDWIND,HPOS,VPOS,OPTION*)

Displays a panel or menu, existing in the local buffer, in a previously created window or in the screen at a position defined in local window coordinates. The specified position corresponds to the top-left corner of the panel or menu. The OPTION variable is a character string :

'B' if panel/menu is to be surrounded by a rectangular boundary

The same panel or menu can be displayed simultaneously in different windows (and with different options) but only once per window. If the same panel/menu is already displayed in the given window, the old version is cleared. A call CMSHOW makes automatically this panel/menu the active one on the screen. Hence, if the position is such that it overlaps with another one, the new one will cover the previous one(s) making part of it invisible.

If the requested panel/menu, IDMP, exists in the local buffer, CMSHOW takes the definition as it

exists there. Otherwise, CMSHOW calls CMREAC to restore the given panel/menu from the data base. Note that the latter causes the existing information in the local buffer to be overwriten. A call to CMSHOW hence does not require the user to make a CMREAC and the information in the buffer can be modified after a CMSHOW has been executed. In the latter case, the screen is not automatically updated but the user is requested to generate the update by calling again CMSHOW or CMUPD.
Entries which are inactive at the time of the CMSHOW are preceded by a '-' (minus) sign in the case of menus and do not appear in the case of panels.

CMUPD (IENTRY,IDWIND)

Allows to update the display of the entry IENTRY in the panel/menu which is currently being displayed in the window IDWIND (by CMSHOW). It takes the information to be displayed from the local buffer (it does not call CMREAC). It clears the existing text for the entry IENTRY and replaces it by the new one. The layout of the panel/menu is, however, not modified. Hence, if the area associated to the previous text is too small, only part of the new text will be displayed. The panel/menu in which the entry is updated may be active or not and the call CMUPD does not change its activation status. If IDWIND=0 (zero), all versions of this panel/menu will be updated.

CMSCRI (IDMENU,IDWIND,IUD*)

Scrolls all items in the menu IDMENU up or down, depending on whether IUP= 'U' or 'D'. This routine cannot be used for panels.

CMREMO (IDMP,IDWIND)

Removes a panel or menu from the given window on the screen, leaving it in all other windows and in the data base. If IDWIND=0 (zero), it is removed from the screen, i.e. from all windows where is displayed. If IDMP=0 (zero), all panel/menus in the window are removed. If the panel/menu was covering another one, the other one becomes again completely visible. The panel/menu can be re-displayed by making another CMSHOW.

4.5 Enquiry routines

The enquiry routines look first whether the requested information exists in the local buffer. If the panel/menu IDMP is not the one in the local buffer, the search is continued in the data base.

NENTRY = ICMNEN (IDMP)

Returns the number NENTRY of entries existing in the specified panel or menu.

CMFIND (TITLE*,TYPE*,IDMP)

Searches the data base for the first occurence of a panel/menu with the same title as given in the character string TITLE. The search may be limited to a specific type and if TYPE is blank, all data are searched. The identifier is returned in IDMP. If IDMP is zero, it means that no data were found with the specified name and type.

CMGTYP (IDMP,TYPE*)

Returns the type associated to the menu or panel IDMP.

CMGET (IDMP,IENTRY,TEXT*)

Returns the text associated to the entry IENTRY in the panel/menu IDMP in the character string "TEXT". For IENTRY=0 (zero) the title is returned. The routine checks the length of the character string TEXT defined by the user and returns at most that number of characters, in order to avoid overwriting. If the string is too short, part of the text can thus be lost.

CMGKND (IDMP,IENTRY,KIND*)

Returns the kind of text held in the panel or menu entry IENTRY into the character string "KIND".

CMGATT (IDMP,IENTRY,IVATTR,ITATTR)

Returns the video and text attributes of the panel or menu entry IENTRY.

CMGETA (IDMP,IENTRY,ATTR*,IARR,IVALUE)

Allows to retrieve the attributes individually. The character string ATTR defines which attribute is addressed:

ATTR = 'V' for video attributes
= 'T' for text atttributes

IARR specifies the array element and IVALUE returns the value of the attribute.

CMGACT (IDMP,IENTRY,FLAG)

Returns the activation/de-activation status in the logical variable FLAG.

4.6 Handling of events on panels/menus

After an event has been detected (see CEVENT) to occur in a panel or a menu, the following routines allow to obtain more precise information on what has occured.

CMLOC (IDWIND,IDMP,TYPE*,IENTRY)

Defines the panel/menu identifier, its type and the entry number of the item which region contains the position of a mouse click. It also returns the identifier of the window to which this panel/menu is associated. The value returned for IENTRY can be as follows:

IENTRY = positive number for a normal entry
= 0 if mouse click in the title area, fixed text or outside the entries of panel
= -1 if corresponds to a request for scrolling up
= -2 if corresponds to a request for scrolling down

CMPOS (IDWIND,IDMENU,IENTRY,HPOS,VPOS,HSIZE,VSIZE)

Returns the position in local window coordinates of the top-left corner and the size of the rectangular area containig the specified menu entry. In case IENTRY is zero, the title region is returned.

CMDIAL (IDPAN,IDWIND,IPOS,IENTRY,BUTTON*)

Provides the possibility to interactively dialog with a panel (not with a menu) after a CMSHOW has been performed. It takes control over the events (see CEVENT). It allows to modify editable text by calling the editor (see chapter 5). Control is returned as soon as the user has selected a button (or equivalent operation). Only one item can be edited or selected at the time. IPOS forces the cursor to be positioned on the area corresponding to the item number IPOS at the start of the interaction. This is useful mainly for editable text. When returning, the routine specifies the entry number of the item, IENTRY, with which the user has interacted (zero if no interaction took place) and a character string, BUTTON, with the contents of the button used to exit from the dialog. A carriage return,<CR>, is equivalent to exiting via the first button defined, provided this is active.

4.7 Example: Panel for message display

The following is an example of using a panel for the display of a message, assuming the CWINIT and CMINIT calls have been performed.

```
      CHARACTER*32 TEXT
      CHARACTER*6 BUTTON
      INTEGER IMESBX,ITEXT,IOK,IDWIND,IENTRY
      REAL HPOS,VPOS

*     Definition of the panel
      CALL CMOPEN ('Message box',IMESBX)
      CALL CMSETA (0,'T',1,1)                  ! all entries bold
      CALL CMKIND (0,'F','N')                  ! all entries are fixed text, no border
      TEXT= 'This is an example of a message'
      CALL CMPAN (2,1,51,1,TEXT,ITEXT)         ! enter the text
      CALL CMKIND (0,'B','B')                  ! all entries from now are buttons
      CALL CMPAN (40,5,10,1,'OK',IOK)          ! make an "OK" button
      CALL CMCLOS                              ! save panel in data base
*
*     dialog with the user (this is of course very simple in this case)
      IDWIND = 0                               ! display in the screen
      CALL CWCTOL (IDWIND,10,5,HPOS,VPOS)      ! compute the position
      CALL CMSHOW (IMESBX,IDWIND,HPOS,VPOS,'B') ! display the panel
      CALL CMDIAL ( IMESBX,IDWIND,0,IENTRY,BUTTON) ! dialog with user
      CALL CMREMO (IMESBX,IDWIND)              ! remove panel from screen
```

4.8 Example: Panel for interactive editing

The following is an example of using a panel for the interactive editing of text, assuming the CWINIT and CMINIT calls have been performed.

```
*     Definition of the panel
      CALL CMOPEN ('Editing box',IEDTBX)
      CALL CMSETA (0,'T',1,1)                    ! all entries bold
      CALL CMKIND (0,'F','N')                    ! all entries are fixed text, no border
      TEXT= 'Please, edit the following text'
      CALL CMPAN (2,1,51,1,TEXT,ITEXT)
      CALL CMSETA (0,'T',1,0)                    ! all entries from now are normal
      CALL CMKIND (0,'E','B')                    ! all entries from now are editable
      CALL CMPAN (2,4,51,5,' ',ITEDT)            ! text not entered here
      CALL CMKIND (0,'B','B')                    ! all entries from now are buttons
      CALL CMPAN (20,12,8,1,'OK',IOK)            ! make an "OK" button
      CALL CMPAN (30,12,8,1,'NO',INO)            ! make a "NO" button
      CALL CMPAN (40,12,8,1,'DELETE',IDEL)       ! make a "DELETE" button
      CALL CMCLOS                                ! save panel in data base
*
*     enter the editable text and dialog with the user
      CALL CMREAC (IEDTBX)                       ! re-activate the panel
      TEXT = 'This is text to be edited.'
      CALL CMFILL (ITEDT,TEXT)                   ! define the text now
      IDWIND = 0                                 ! display in the screen
      CALL CWCTOL (IDWIND,10,5,HPOS,VPOS)        ! compute the position
      CALL CMSHOW (IEDTBX,IDWIND,HPOS,VPOS,'B')  ! display the panel with border
      IPOS = ITEDT                               ! put cursor in editable text
      CALL CMDIAL (IDWIND,IEDTBX,IPOS,IENTRY,BUTTON) ! dialog with user
      CALL CMREMO (IEDTBX,IDWIND)                ! remove panel from screen
      IF (BUTTON.EQ.'OK') THEN                   ! 'OK' button hit
         IF (IENTRY.GT.0) THEN
            CALL CMGET (IEDTBX,IENTRY,TEXT)      ! get edited text
         END IF
      ELSE IF (BUTTON.EQ.'NO') THEN              ! 'NO' button hit
         user did not want to edit
      ELSE IF (BUTTON.EQ.'DELETE') THEN          ! 'DELETE' button hit
         user wants this piece of text to be deleted
      END IF
```

5. STRUCTURED MENUS

The present package is built on top of the preceeding window, panel and menu management packages in order to provide a limited possibility to structure menus in trees. It comprises the menu structuring, the display and the interaction with such menu structures. It is restricted to the case where **all menu titles are unique**.

5.1 Menu structuring

CMCOLL (NENTR,IDMPS,TITLE*,IDMEN)

Collects a series of previously defined menus (or panels) into a new menu. The existing menus are given in the array IDMPS of their identifiers with NENTR menus in total. The character string TITLE associates a new title to the newly created menu, which identifier is returned as IDMEN. The new menu is not installed in the panel/menu data base, but remains in the local buffer (which previous information is overwritten). It can be used as a normal menu, for editing (e.g. CMENU allows to introduce new entries) and can in turn be collected in another higher level menu. To install it in the data base, a call to CMCLOS is necessary.

5.2 Display of a menus structure

Menu structures should not be displayed by means of the CMSHOW routine, as in this case the links between the menus are not recognized. A special routine is provided to display the root menu of a structure:

CMINST (IDWIND,IDMEN,WTYPE*)

Causes the root menu IDMEN of a menu structure to be installed and displayed in the window IDWIND (where IDWIND=0 means the screen). Only one menu structure can be installed in any window and a previous one, if any, is erased. The option WTYPE is a character string which allows to select various modes of display, as defined in Section 3.3 (except the option 'P'). The same defaults apply as described for the routine CMTYPE in Section 4.3.
Note that the display mode applies to the whole menu structure, not only the root menu. The same menus can be individually displayed by CMSHOW in another window and the same structure can also be displayed in a different window, possibly with other display options.

CMUNIN (IDWIND)

Un-installs, i.e. removes the menu structure from the window (and screen), making it available for other display.

5.3 Interaction with menu structures

After the root menu has been installed in a window, the interaction with the menu structure can be handled by the routine

CMEXEC (IDWIND,IDMEN,IENTRY)

As soon as a selection (mouse click) has been made in a window IDWIND in which a menu structure has been installed, the user should call the routine CMEXEC to handle the interaction further. The routine determines which menu entry was affected by the selection, highlights this entry (possibly after some cleaning up of the window contents) and displays the next down menu, if any. It then returns control to the user telling the identifier IDMEN of the selected menu and the entry number IENTRY in this menu. The way and the location in which the next menu is displayed in the window depends on the options given with the CMINST.
It should be noted that to find the down menu, CMEXEC starts from the menu item in the up menu and performs a CMFIND to get the corresponding menu identifier. This is the reason for the limitation to uniquely defined menu titles for use in the menu structures.

6. TEXT EDITING

The text editing is normally performed by setting up a panel with an area for editable text. This is described in section 6.1. The calling sequence and the functionality of the editor routine proper is described in section 6.2 in case an application needs to call it directly.

6.1 Text editing in a panel

The easiest for an application which needs to edit text (or simply read text from the screen) is to set up a panel and call CMDIAL to dialog with the user. An example of the corresponding calling sequences has been given in section 4.7.2. The interaction with the user then proceeds as follows:

- At the start, the cursor is normally positioned in the beginning of the editable area (see the example). If this is not done or if the user needs to edit another area in the same panel, a mouse click in the corresponding area will activate the editor for that area. In alpha-numeric mode (see section 2.3), the "next" and "previous" instructions can be used to scan through the areas of the panel.
- **Select**: a position between two characters is selected by moving the mouse on top of it and clicking. This positions the cursor at this place. A second mouse click ("select click") selects the whole character string between the two mouse clicks. This is indicated by displaying the string in reverse video. The first and second mouse click can also be entered via keys for the "no mouse" operation (see Appendix 1).
- **Add**: characters entered after a position has been selected by the mouse are inserted in the text at that position.
- **Delete**: the character in front of the cursor position or a whole selected character string is deleted by pressing the backspace, <BS>, key. In this case the character string is not saved and the operation can hence not be undone.
- **Cut**: the selected character string can be deleted by pressing the "cut" key. This keeps a copy of the deleted string, which thus remains available for a paste operation.
- **Copy**: copies, by pressing "copy", the selected character string in a local buffer without affecting the text. The string can thus be pasted elsewhere.
- **Paste**: inserts, by pressing "paste", the character string saved in the local buffer into the text at the position indicated by a previous mouse click. The cursor remains positioned at the end of the pasted character string.
- To exit from the editor, the user can press the carriage return <CR> key. This exits from the panel as if the first button were chosen, provided it is active. Alternatively, the user can bring the mouse on the button of his choice and click. In order to use the <CR> as exit mode, it is advisable to always define the first button as "OK", meaning that the operation has been successfully completed.

The keys corresponding to the above operations are described in Appendix A1.

6.2 The editor routine

CMEDIT (IDWIND,IDMP,IENTRY,HPOS,VPOS,TEXT*,IFLAG)

Allows to edit the text provided in as panel entry defined by IDWIND, IDMP and IENTRY. The positions HPOS and VPOS indicate the location of the mouse in local coordinates of the area IENTRY. The routine then retrieves the text in the panel and starts the interaction with the user, taking control over the events (CEVENT). It returns the edited text in the character variable TEXT and signals the exit mode in IFLAG:

IFLAG = 1, if exit with a <CR>
= 2, if exit with a backspace <BS> when no character is left in the text
= 3, if exit on a mouse click outside the editable area
<0, if the text has not been changed during the interaction

Care is taken that the number of characters returned in the variable TEXT does not exceed the length of the variable, hence the end of the text may be lost if it is too short.

7. IMPLEMENTING IN DIFFERENT ENVIRONMENTS

An effort was made to ease as much as possible the implementation of the package for different operating systems and different displays. The menu and panel management routines should be transportable without change. The environment dependent parts are all concentrated in the window manager and in the routine CEVENT. Even then, it is likely that large fractions of the code will be re-usable for other environments, as most of the logic is still pure Fortran77. In particular, the saving of the logical relations between windows, their type definition and attributes can probably be used. In-line comments are provided to help the user to understand the functionality of the routines and Appendix A2 contains a list of all the SMG routines called by the package.

7.1 Comments on the CEVENT routine

Two types of asynchronous events are controlled by 'event' subroutines :

a) Broadcast messages
b) Unsolicited keyboard input

Other asynchronous events such as CTRL/Y are left under the user control. **Broadcast messages** (for example, mail notifications or operator messages) can destroy or distort the screen image. Subroutine CMXBRO traps the broadcast of messages to the terminal and displays them in a 'message' window which becomes the front-most one. This temporary window is displayed under the mouse and is removed from the screen when the user moves the mouse out of it.

Unsolicited keyboard input (any key pressed on the keyboard while the running program does not require keyboard input) is trapped by subroutine CMXAST. Up, down, left and right arrows plus PTO the 'no mouse' keys Up, Down, Next, Previous (see Appendix A1) are used to modify the mouse position. The other keys are stored in a set of buffers together with the additional information which is necessary to process that keyboard input (see common block /EVENT/).

CEVENT is almost machine independent. It finds all the required information in buffers (filled by CWXAST), combines it with information from the window database and defines "what" has been done by the user, "where" and in "which region".

The two AST routines (CWXBRO and CWXAST) are declared as asynchronous system traps in the initialisation routine CWINIT.

APPENDIX 1: Definition of keys

The definition of the keys for the implementation on the VAX with a VT100 type of terminal is as follows:

For editing operations:

Delete	<BS>
Cut	CTRL/D
Copy	CTRL/V
Paste	CTRL/P

For the "no mouse" operation:

Click	PF1
2nd click	PF2
Up	8, on numeric keypad
Down	2, on numeric keypad
Next	6, on numeric keypad
Previous	4, on numeric keypad

APPENDIX 2 : Implementation with SMG

The following contains the list of all **SMG routines** called from the window management package or from CEVENT with the name of the calling routine. The SMG routines are listed in alphabetic order. The CW routines in parenthesis are not directly user callable routines.

SMG routine :	**called from :**
SMG$CHANGE_PBD_CHARACTERISTICS	CWINIT
SMG$CHANGE_RENDITION	CWCHOM, CWCMOV, CWCREL, CWCURS, CWUPDA, (CWXBXS)
SMG$CHANGE_VIRTUAL_DISPLAY	CWATT, CWSETA, CWTITL, CWTYPE, (CWXACT)
SMG$CONTROL_MODE	CWINIT
SMG$CREATE_PASTEBOARD	CWINIT
SMG$CREATE_VIRTUAL_DISPLAY	CWIND, (CWXBRO)
SMG$CREATE_VIRTUAL_KEYBOARD	CWINIT
SMG$DELETE_CHARS	CWDELC
SMG$DELETE_VIRTUAL_DISPLAY	(CWXACT), (CWXAST)
SMG$DRAW_LINE	CWLINE, CWPOLY
SMG$DRAW_RECTANGLE	CWRECT, (CWXBXS)
SMG$ENABLE_UNSOLICITED_INPUT	CWINIT
SMG$ERASE_DISPLAY	CWCLR
SMG$FIND_CURSOR_DISPLAY	CWGET, CWGETL, CWGSTA, (CWXAST)
SMG$GET_BROADCAST_MESSAGE	(CWXBRO)
SMG$HOME_CURSOR	CWCHOM
SMG$INSERT_CHARS	CWADDC
SMG$LABEL_BORDER	CWTITL, CWTYPE, (CWXBRO)
SMG$MOVE_VIRTUAL_DISPLAY	(CWXACT)
SMG$PASTE_VIRTUAL_DISPLAY	(CWXACT), (CWXBRO)
SMG$PUT_CHARS	CWMARK, CWRITE, (CWXBXS)
SMG$PUT_LINE	(CWXBRO)
SMG$READ_KEYSTROKE	(CWXAST)
SMG$RETURN_CURSOR_POS	CWCHOM, CWCMOV, CWCPOS, CWCREL, CWCURS, (CWXBXS)
SMG$SCROLL_DISPLAY_AREA	CWRITE
SMG$SET_BROADCAST_TRAPPING	CWINIT
SMG$SET_CURSOR_ABS	CWCMOV, CWCREL, (CWXBXS)
SMG$SET_KEYPAD_MODE	CWINIT
SMG$SET_PHYSICAL_CURSOR	CWCHOM, CWCMOV, CWCREL, CWINIT, (CWMOUSE), (CWXACT), (CWXAST), (CWXBRO), (CWXBXS)
SMG$UNPASTE_VIRTUAL_DISPLAY	(CWXACT)

Some routines of the CW package depend on SMG only because they test an **SMG variable**, but do not call SMG routines:

SMG variable :	**used in :**
SMG$K_TRM_PF1	CEVENT
SMG$K_TRM_PF2	CEVENT

The following routines of the CW package are **independent of SMG**:

CWCTOL	CWGATT	CWGDWN	CWGETA	CWGETC
CWGETS	CWGPOS	CWGSUP	CWGTIT	CWGTYP
CWLOC	CWLTOC	CWLTOS	CWMOVE	CWOVAL
CWPOS	CWREMO	CWRND	CWSCRA	CWSHOW
CWSIZE	CWSTOL			