An APL 90 Summary

Primitives and syntax
Generally, APL 90 1.05 conforms to the ISO standard. Some extensions in the direction of IBM's APL2 are implemented.

Primitive data types are numbers, characters and atoms. The prototypes of these objects are the number zero, the space character, and the null atom, or *nilde*.

Jot : the Jot (⍤) is syntactically considered as a function. However, its ⎕NC, as can be tested in a user defined operator, is 0.

Strand notation is available. However, strand specification is not currently allowed.

Function specification (such as f[⌿]) is available.

Nested arrays are fully handled by the system. Strand notation and some primitive functions allow creation of enclosed arrays. Scalar functions and most structural functions apply to enclosed arrays. Other primitive functions and operators do not presently extend to enclosed arrays.

User defined functions are ambivalent.
User defined operators also derive ambivalent functions. In both cases, the presence of the optional left parameter can be tested using ⎕NC.

New primitive functions include enclose (monadic ⊂), disclose (monadic ⊃), pick (dyadic ⊃), match (dyadic ≡), depth (monadic ≡), first (monadic ↑), type (monadic ⎕NC), enlist (monadic ∊), dex (monadic and dyadic ⍳), lev (monadic and dyadic ⍴).

New primitive operators are each (¨), exchange (⍨) and compose (∘).

Identifiers are defined as starting with an alphabetic character, optionally followed by alphanumeric characters. Alphabetic characters include upper case letters ⍺ to ⍵, lower case letters a to z, underlined upper case letters ⍶ to ⍹, underlined digits, ⍬ and ⍕, and some accentuated letters (é è à ì ù â ô î ê û ë ï ö ü). Alphanumeric characters include the above, plus the digits 0 to 9, ∆ ⍙ ‘ ’ and ^. Note that ⍺ and ⍵ are recognized as mono-characters identifiers. Finally, you can use the ASCII vertical bar (which differs from the character representing the absolute and modulo functions) as an escape character to turn every character string into an identifier : |+2 "| is a 4 characters name and can be used like any identifier.

Numeric constants. Based notation is available for integer constants. A based constant is composed of a radix, expressed as a decimal integer, the letter B, and a number expressed in the corresponding base. Here are some examples of such constants : 2B1001 (number nine), 16B124C41, 36BC3PO...

Zilde : the zilde (⍬) can be used to denote the empty numeric vector.

Nilde : the nilde (⍬⍬) is the prototype of atoms.

Brace notation is available. Brace notation is a general notation for vectors of simple, mixed, or enclosed items. The execute function applies to such constants :

```
“{⊂[1 2},3
```

defines A as the vector 1 2 3.

System variables
All the traditional system variables are available. Session parameters, such as Lnlt, Lpw, Ltz or Lsp, retain their values through successive loads and stores. Changing the character size or aspect with the TYPO menu changes accordingly the value of Lpw. Lsp is not used by the system and can receive any kind of value.

Some APL 90 system functions
Most of the traditional system functions are available. Error handling functions include Le□, Lec, Les, Let and Lem.
Ldr (monadic function) Displays the internal type of its operand : 0 (logical), 1 (8-bits char), 2 (16-bits char), 3 (32-bits char), 4 (8-bits integer), 5 (16-bits integer), 6 (32-bits integer), 7 (32-bits floating point), 8 (64-bits floating point), 9 (128-bits complex *not available*), 10 (mixed array), 11 (enclosed array), 12 (64-bits atom), 13 (table *variable length*), 14 (object *variable length*, *not available*).
Lds (monadic function) Displays the structure and the contents of its operand, and is similar to the APL2 DISPLAY function.
Llru (niladic function) returns a copy of the last computed value of the previous line.
Lbox, Ldbr, Lss are very close to their APL 68000 counterparts.

Macintosh specific system functions provided :
Lkeys : (niladic function) Gives a 128 element boolean vector representing the status of the keyboard.
Lpeek : (monadic function) Reads a 16 bit integer at the specified memory address.
Linkey n 0 : Checks during “n” seconds the keyboard, and returns -1 or a key number.
Linkey n 1 : Checks during “n” seconds the keyboard, and returns a character or an empty vector.
Lresource Reads or updates existing Macintosh resources.

Commands
Traditional APL commands are available. These include)FNS,)VARS,)LIB,)CLEAR,)WSID,)LOAD,)SAVE,)COPY,)PCOPY,)SI,)SINL,)ERASE and)OFF. Commands may also be entered in lower case (or, actually, any combination of lower and upper case).
)SIC resets the contents of the SI. A name or a number can be specified.
)SIS lists the SI together with the suspended lines.
)DEPTH lists or sets size of the SI.
)SYMBOLS lists or sets size of the symbol table. Symbol tables automatically expend as needed.
)STORE and)PSTORE act much as)COPY or)PCOPY, and allow insertion of objects of the active WS in an inactive one.
)OPS prints the names of user defined operators.
)NMS gives the names of all objects of the current WS and their classifications.
)INPUT *filename* opens *filename* and executes its contents.)INPUT without parameter, or reaching the end of the input file, closes the file. This command allows the option -NOECHO after the file name to suppress echo on the screen while reading a file.
)OUTPUT *filename* opens *filename* and keeps in it a copy of the session.)OUTPUT without parameter closes the file.
)QWITT returns to Finder, losing the virtual space.
)KWORD defines a keyword for an APL character representing a function, an operator or almost any other symbol :

```
)KWORD RHO 130
```

where 130 is the ZCODE for character ℝ.
)KWDS prints the defined keywords. You can erase or copy keywords.
Commands such as)FNS,)VARS,)LIB,)NMS,)KWDS,)OPS allow regular expressions :

```
*PL*   represents names containing the letter sequence PL.
??A    represents three character names ending with A.
```

)TIME <n> can be used to execute repeteadly APL expressions and mesure execution times. Values from 1 to 7 give different repetition factors from 1 to 10000. A value of 0 resets timing.

) or)n where n is a number can be used to edit last entered line.
In most commands dealing with files, if you use a L instead of a name, you will be prompted with the standard dialog box for creating or opening the file.

Keyboard
The layout of the keyboard is defined by resources of type ‘AKBD’. Each is a set of 9 tables of 96 bytes, giving the L□→ codes associated with keyboard key numbers. These 9 tables correspond to APL mode (normal, shift, option, shift+option), ASCII mode (normal, shift, option, shift+option), and Command mode. Six different keyboards are provided, with resource numbers 256 to 261. Keyboard configuration can be selected with the keyboard menu. Additionally, the VS “KEYB” contains a workspace that can be used to display or change the keyboard configurations.

Function editor
The "del" function editor lets you edit simultaneously up to six functions, and accepts all standard editing commands, with some extensions. Most commands act on "line_expressions", wich are of the form "n1 n2 ... np" (set of the lines n1, n2, ... np), "n-p" (lines n to p inclusive), "n-" or "-p" (from n to the end or from the beginning to p).

;L' to print contents of current object

;L line_expression' to print specified lines of current object

;H line_expression' to delete specified lines of current object (T or U can be used instead of H).

;Y line_expression' to copy at that point specified lines of current function.

; fun L line_expression' to print specified lines of object FUN.

; fun Y line_expression' to copy at that point specified lines of object FUN.

In these two commands, if FUN is the name of current object, it refers to the *previous* contents of the object.

;n.p' to include new lines after n.

;n L p' to edit line n. (uses .,/123..□b...)

;I' to renumber lines of current object.

;Q' to print incorrectly formed lines.

;-' to position at the end of current function

;] fun' to open for editing object fun

;]' or ;[' to switch to the right (or left) object in the edit ring.

Multi-workspaces experimental facility

APL 90 currently provides an experimental multi-workspaces facility. Note that in future versions of APL 90, this feature may be absent, or available in a different form.

The same VS can hold more than one active WS, and these WS are saved when you quit APL. These WS are designated by numbers. The command “)WS” prints the number of the current WS, and the set of active WS :

```
      )WS
IS  2  [2,3,5,20,21,22]
```

The command “)WS” with a parameter switches to the designated WS. If the WS does not exist, it is created :

```
      )WS  8
WAS  2
      )WS
IS  8  [2,3,5,8,20,21,22]
```

To delete a WS, use the “-OFF” option :

```
      )WS  –OFF  3
      )WS
IS  8  [2,5,8,20,21,22]
```

Available numbers for WS are 2 to 9, 12, and 20 to 29. The WS#2 is the standard WS, and is the one used when you connect to APL. The WS#12 is used by APL as a save area for the active WS when the VS has been damaged. An additional restriction applies to WS #20 to 23, which are used by the system. 20 is the LongAtoms symbol table, and contains the names of long symbols. 21 is the ClearWorkspace, 22 is the Library, and 23 is reserved for the object extension. These WS cannot be cleared or deleted.

Expressions can be executed in such WS with the Lzx function :

```
      □ Lzx e
```

executes expression E in the WS A.The left parameter is the number of a WS, the right parameter is, like the operands of “, Le□ or Lec, a character vector or a generalized array :

```
      8 Lzx Km[2 10ℝI7K
      r[ 8 Lzx {-/m}
```

The result of the function is the result of the evaluation of the expression in the alternate WS. Values can be communicated with the Lsp shared session parameter :

```
      Lsp[3 4 5 ` 8 Lzx {x[ Lsp}
```

MLDF Functions

APL 90 provides a facility that lets you define functions in machine language. Such a function (or MLDF for short) acts like a locked user defined function. It can be saved in an inactive WS, loaded, copied, or erased. Since the definitive interface of such functions with APL 90 is not currently completely defined, we do not provide the facility to define MLDF. However, some MLDFs are provided in the MLDFS folder. These functions can be loaded in the active WS with the command :

)MLDFLOAD *filename funname type*

where *filename* is the name of the external file containing the code of the function, *funname* is the APL name that you wish to give to the function, and *type* should be 1 for a niladic function, and 2 for a monadic or dyadic function. If you use a L instead of *filename*, you will be prompted with a dialog box to select a file.

Quickdraw interface

Introduction
An experimental quickdraw interface is provided under the form of a set of MLDF functions. The code of these functions is available in the folder MLDFS. These functions work exclusively inside the APL 90 window. Graphic notions are not developed here, and more explanations can be found in “Inside Macintosh” and other specialized litterature.

Definitions
We will use the following terms :
point : a X-Y position expressed as two integers.
pattern : a 8 by 8 bitmap, that can be used as a color for the background, the pen, or to fill a shape. A pattern can be expressed as a vector of 64 logical elements (or a 8 by 8 array), a vector of 8 numbers or characters, a vector of 4 16-bits integers, or 2 32-bits integers. For example, the black pattern can be expressed as 64ℝ1, or 8ℝ¯1, or ¯1 ¯1...
pen : the element used to draw inside a window (or, more precisely, a grafport). A pen has a position (a point), a size (width and height, expressed as two integers), a mode (an integer) and a pattern.

APL 90 related facilities
Some functions provide ad hoc facilities related to this window :
SCREEN (monadic function). The parameter is a numeric scalar. A value of 1 erases the screen. 2 redisplays the APL session. 3 redisplays the indicators area. 4 erases the indicators area.
CURSOR (monadic function). The parameter is a numeric scalar. Values from 0 to 4 select different cursors shapes. ¯1 calls the ObscureCursor function, ¯2 HideCursor, and ¯3 ShowCursor.
IMASK (monadic function) lets you select an APL 90 internal interrupt mask. The parameter is a numeric scalar. A value of 0 inhibits the handling of events by APL 90, a value of ¯1 restores this handling.
CLIPRECT (monadic function) lets you select a clipping rectangle for further calls to quickdraw.

Pen handling
A first set of graphic functions let you modify some characteristics of the graphic window :
GETPEN (niladic) returns the position of the pen as a 2 elements numeric vector.
GETPENSTATE (niladic) returns a 9 elements numeric vector representing the position of the pen (2 numbers), the size of the pen (2 numbers), the mode of drawing (1 number), and the pen pattern as 4 numbers.
PENNORMAL (niladic, no result) sets the size of the pen to 1 1, the mode of drawing to 8, and the pen pattern to black.
PENSIZE (monadic, no result) lets you specify the size of the pen as a 2 elements numeric vector.
PENMODE (monadic, no result) lets you specify the mode of the pen as a number.
PENPAT (monadic, no result) lets you specify the pattern of the pen as a pattern.
BACKPAT (monadic, no result) lets you specify the pattern of the background as a pattern.
HIDEPEN (niladic, no result) hide the pen. Subsequent commands (such as text or line dawing) will be executed, but nothing will be displayed on the window.
SHOWPEN (niladic, no result) is the inverse function of HIDEPEN. Note that HIDEPEN and SHOWPEN can be nested.

Line drawing
MOVE (monadic, no result) moves the pen, without drawing, with a relative displacement specified as a 2 elements numeric vector.
MOVETO (monadic, no result) moves the pen to a new point, without drawing.
LINE moves the pen and draws the corresponding line, with a relative displacement specified as a 2 elements numeric vector.
LINETO (monadic, no result) moves the pen to a new point, and draws the specified line.
Note that LINE and LINETO allow an arbitrary number of positions.
DRAWLINE (monadic, no result) with 4 operands is equivalent to MOVE on the fisrt two, followed by LINE on the last 2.

DRAWLINETO (monadic, no result) with 4 operands is equivalent to MOVETO on the fisrt two, followed by LINETO on the last 2.
Note that DRAWLINE and DRAWLINETO allow an arbitrary number of positions to be specified.

Text handling
DRAWSTR (ambivalent function without result). The right parameter must be a character string. In its monadic form, this text will be displayed in the current font, starting at the pen position. In the dyadic form, the left operand may be a point (two elements), or a vector of points.
A DRAWSTR B, when A is a point, is equivalent to MOVE A followed by DRAWSTR B. If A is a vector of points, each point of the vector will be interpreted as a position for drawing one character of the string. This allows for a fast labelling of a set of points.
TEXTFACE (monadic, no result) lets you specify the aspect of the text, as an integer parameter. Allowed values are 0 (standard), 1 (bold), 2 (italic), 4 (underlined), 8 (outline), 16 (shadow), 32 (condensed) and 64 (extensed). These values may be combined by addition : a parameter of 19 (1+2+16) specifies bold italic shadowed.
TEXTMODE (monadic, no result) specifies the mode of text drawing.
TEXTFONT (monadic, no result) specifies the font to be used as an integer. 0 is system font, 24 is the APL font, and other values depend of your system configuration.
TEXTSIZE (monadic, no result) specifies the size (in points) to be used when drawing text. For example, the APL font is provided in 9, 12, 18 and 24 points. Other sizes will be obtained by scaling one of these fonts.
FONTINFO (niladic) returns a 4 elements numeric vector describing the current font : ascent, descent, max width and leading. Note that the current line height can be determinated by :

```
-/1 1 0 1/fontinfo
```

Mouse and other functions
BUTTON (niladic) returns the current state of the mouse button as 1 for pressed, 0 otherwise.
GETMOUSE (niladic) returns the current position of the cursor as a point.

Shape handling
A last set of graphic functions handle shapes provided by quickdraw. These shapes are Rectangles, Ovals, RoundRectangles, and Arcs. A shape is defined by 4 or 6 numbers :
Rectangles (RC). A Rectangle is defined by 4 integers, the position of its upper left corner and its lower right corner.
RoundRectangles (RR). A RoundRectangle is defined by 6 integers, the position of its upper left corner and its lower right corner, and the lengths of the axes of the ellipse used to draw its corners.
Ovals (OV). An Ovals is defined by 4 integers, which are the positions of the upper left and lower right corners of the rectangle that surrounds this Oval.
Arcs (AR). An Arc is defined by 6 integers, the first four defining the oval, the last two being the starting and ending angles.
A set of functions apply to these shapes. These functions are PAINT (fill the shape with black pattern), ERASE (actually, fill the shape with the background pattern), FRAME (draw the shape), INVERT and FILL. Names of the functions provided are the concatenations of the names of the operation and the names of the shape. For example, PAINTRR is the function that paints RoundRectangles. All these functions apply to an arbitrary number of objects of the same shape, which sould be provided as a vector of 4n (or 6n) elements, or as a n by 4 (or n by 6) mattrix. All these functions are monadic without result, except the FILL functions which are ambivalent. The left argument, if provided, is a pattern. A default grey value is assumed for the monadic forms.

Other facilities
PTINRECT (dyadic function) A PTINRECT B, where A is a point and B is a set of *n* rectangles, returns a *n* elements logical vector indicating if the point is inside the corresponding rectangle.

An example of the use of these functions is provided in the WS “UPDKB” of the VS "KEYB”, which lets you see or modify the configuration of your keyboard.

The APL ScrapBook Experimental Facility

Since the memory management used inside APL is very different from the one that is built inside the Macintosh, we found useful to provide an interface between the objects of APL and those of the Macintosh. This interface is what we call the APL ScrapBook, or ASB for short.

The ASB is memory resident, uses the Macintosh memory manager, and is independent of the virtual space. The ASB can store APL and Macintosh objects. Objects of the ASB are designated by a type and a number. The type is a 4 characters vector, the number is a 16 bits integer : such an identification is therefore very close to the one used by the Macintosh resource manager. We will use the following notation :
ID : a two elements generalized vector, the first item being a 4 elements character vector, the second being an integer scalar. Example : `KpictK 25` is the object of type PICT and number 25.
IDT : a three elements generalized vector, the first item being a 4 elements character vector, the second being an integer scalar, the third an internal type specification (in the sense of the `Ldr` function). Example : `KpictK 25 5` is the object of type PICT, number 25, expressed with type 5 (16 bits integers).

Communication between APL and the ASB
Exchanging data with the ASB can be done with the ASBOBJ function :
ASBOBJ X, where X is an ID or an IDT retrieves the object from the ASB and returns its value.
V ASBOBJ X, where X is an ID or an IDT inserts the object V in the ASB. For example :

```
KhelloK □sbobj KtextK 10
```

creates the entry TEXT 10, with a default type of 1 (character).

```
□sbobj KtextK 10 4
```

reads this entry, and returns the value with type 4 (8 bits integers).
An object is destroyed when you create a new object with the same identification, or with the KILLOBJ function. For example :

```
killobj KtextK 10
```

Finally, the ASBINFO niladic function returns the identifications of the objects of the ASB as a 2 elements enclosed array.
Note that the ASB doesn't take care of the rank of the objects, and always returns vector.
ASBOBJ returns a negative scalar in case of errors.

Communication with the Macintosh clipboard.
Objects can be exchanged between the ASB and the Macintosh Clipboard. Since the clipboard can handle only one object of each type, the number you specify is used only to provide a full ASB identification. For example :

```
fromclip KtextK 30
```

reads the TEXT object of the clipboard, and saves it as the ASB object of type TEXT and number 30. Result is the size in bytes of the object, or a negative error code. Writing to the clipboard is done with the TOCLIP function :

```
toclip KpictK 10000
```

adds as PICT entry the object of type PICT and number 10000 of the ASB. Return code is 0 in case of success, or a negative error code. Finally, the niladic function ZEROCLIP empties the clipboard.
Note : displaying the clipboard with the the Edit menu will prevent correct operation of these functions.

Communication with Quickdraw
The ASB can be used as an intermediate between APL 90 and Quickdraw to handle operations on Polygons, Pictures, Regions, BitMaps and Icons. These objects are kept in the ASB with types POLY, PICT, REGN, BMAP and ICON respectively, and can be operands to calls to Quickdraw. Note that only one quickdraw object (picture, polygon or region) can be opened at the same time. Furthermore, the opened object will be automatically closed if the interpreter returns to terminal mode. Note also that for a correct operation, you should mask the interrupt handling of APL 90, and select appropriate clipping rectangle.
Operations on Polygons
Polygons are designated by a number. Polygons must be loaded in the ASB before any operation on them. Polygons can also be directly build int the ASB with the following operations : OPENPOLY (monadic function). Right operand is a number chosen by the user. For example, the following expression opens the object of identification POLY 30 :

```
openpoly 30
```

When a poly is opened, all subsequent lines drawing (by LINE or LINETO) won't be displayed on the screen, but will be saved as parts of the poly. For example :

```
lineto 120 120 140 120 120 140
```

will construct a triangle. The polygon can then be closed, by the niladic CLOSEPOLY function. Such a polygon, as long as it is kept in the ASB, can be drawn with the following functions : FRAMEPOLY, PAINTPOLY, ERASEPOLY, INVERTPOLY (monadic functions), that take as a right operand a polygon number in the ASB. FILLPOLY (ambivalent function) applies to a polygon number and takes a pattern as an optional left parameter. KILLPOLY (monadic function) can be used to delete a poly.
Operations on Pictures
Pictures are designated by a number. Pictures must be loaded in the ASB before any operation on them. Pictures can also be directly build in the ASB with the following operations : OPENPIC (dyadic function). Left operand is a number chosen by the user. Right operand is a rectangle representing the frame of the picture. For example, the following expression opens the object of identification PICT 65 :

```
65 openpic 100 100 140 210
```

When a picture is opened, all subsequent calls to Quickdraw won't be displayed on the screen, but will be kept as part of the picture definition. The picture can be closed by the niladic CLOSEPICTURE function. The dyadic function DRAWPIC can display such a picture. The left operand is a picture number. The right operand is vector that can be empty (displays the picture in its original frame), have two elements (displays the picture with its upper left corner in the specified position), or have four elements (fits the picture in the specified rectangle). The monadic function PICFRAME returns the original frame of the picture. KILLPIC (monadic function) can be used to delete a picture.
Operations on Icons
An Icon is represented by a 32 by 32 logical matrix, or a 32 long integers numeric vector. An Icon can be drawn on the screen with the PLOTICON function, as soon as it is in the ASB with type ICON. Left operand if PLOTICON is an icon number. The right operand is vector of 2 elements (plot the icon with its upper left corner in the specified position), 3 elements (the 3rd is then a magnifying factor, 32 for standard size), or four elements (fits the icon in the specified rectangle).
Operations on regions
{to be defined}
Operations on bit maps
{to be defined}

Communication with resources files
Data can be exchanged between the ASB and a resource file. You open a resource file with the monadic function OPENRF. The parameter is a character vector representing the name of the file you want to open. If you specify an empty vector, you will be prompted with a dialog box to select a file. The result is either a positive number in case of success, or a negative return code. Note that only one Resource file can be opened at one time, and that the previously opened file will be closed. The file can be closed with the niladic function CLOSERF. You can read resource with either READR (the resource will disappear when you close the file), or COPYR (the resource will be detached from the file, and will stay in the ASB). Operands of these functions are an ID or an IDT. You can use a resource editor to get an idea of the kind of resource that you can find in a file you haven't defined yourself. For example, to read the “help” files of MacPaint, you can use :

```
openrf Km□cp□intK
re□dr KpictK 2400
2400 dr□wpict Ë
closerf
```

Changing the APL 90 Configuration

APL 90 uses the resource CONF of number 0 to describe some parameters of its configuration. This resource is interpreted as a set of 16 bits integers. The following entries can be modified by the user :

• Entry 0 is the APL cache size, that is the number of 4k bytes pages to be kept in memory. This value should not be less than 16. Current value is 20 (hexadecimal $0014), so that APL 90 can run on a 512K Mac, or with the switcher. This value should be increased for better performances if more memory is available.

• Entry 1 is the indicators position preference. Indicators will be in the bottom of the screen if this value is 0, at the top of the screen if this value is 1.

• Entry 2 is the keyboard preference. This is the number of the keyboard resource (of type AKBD) to be used for keyboard interpretation. If set to 0, APL 90 will select configuration 256 for the “old” keyboard, 257 for a Mac Plus keyboard. Otherwise, this should be a number between 256 and 261. See the VS “KEYB” for more informations.

You can change these values with any resource editor, or from APL with an expression like :

```
x[Lresource KconfK 0 5
x;1 2 3'[60 1 260
x Lresource KconfK 0 5
```

This will select a cache of 60 pages, the top position for the indicators, and the keyboard configuration number 260.

Note : To be able to work with the switcher, APL 90 contains a SIZE resource for use by the switcher. APL is configured with a minimum size of 512k and a prefered size of 640k. If you use these values, the size of the cache should be kept between 20 and 40.

Features of APL 90 not available on the Macintosh version

Lnlt is not used. Error messages and command names can be changed by using a resource editor such as ResEdit or REdit (beware of “Servant” which once destroyed 3 weeks of labor!). French and English versions of APL 90 are available.

Lci is not available.

Lcc and Ltcc are not yet available.

Lul always returns 0 (should probably be 1).

Ltz is not used.

“)SH” and “)HOST” are not available.

However, APL 90 allows the definition of machine language defined functions. Such a function can be loaded with the command “)MLDFLOAD”. The MLDF folder contains a small set of MLDF functions, whith an APL executable file to load them.

Known problems with current version of APL 90

Performances

Current performances are relatively poor on this version ; for example, the LOOP function wich runs in about 7 seconds on a 68000 8MH UNIX computer takes about 18 seconds on the Macintosh, due to the overhead of event handling, although some improvements have been made since release 0.98. This should be partially corrected in next version.

Some additional problems come from the adaptation to the Macintosh, because of the C compiler used, and of the Macintosh itself. Most problems currently range in three categories :

- File i/o. This includes management of the virtual memory and file input/output.
- Floating point handling. Floating point has not been fully tested, and some internal procedures may return incorrect results.
- Screen management. Screen management is currently very slow. Next version should include a full multi window management, giving simultaneous access to multiple workspaces.

Corrected Bugs

Many internal bugs have been corrected since version 0.98. The most useful correction is the suppression of an infinite loop which occured when an error was detected in a line with parenthesed expressions.

Non Corrected Bugs

Some bugs of the current version version are already known.

- The Llx expression is not executed when you load a WS.
- Lines using the diamond separator are sometimes incorrectly displayed.
- When using a quad for file names in some commands, you may have to select twice the same file.
- Sessions variables may be erased when an error occurs in a “)VMEM” or “)VMINIT” command. Some obscur problems also occur when switching from a VS to an other.
- You may find inconsistancies in the handling of infinite numbers or NANs.

If you find a bug, try to characterize it as much as you can, and please let us know. This will help you and the others getting better and better versions.

Known features

This word is used according to Iverson, who defines one man's feature as another man's anomaly.

• User defined functions : user defined functions are “logically” duplicated when they are executed. While this corrects some inconsistancies of the language, this also prevents a function being modified if it appears in the SI. You can edit and modify such a function, but the version that appears in the SI won't be changed.

• When specifying file names, unless the file is in the same disk (MFS) or the same folder (HFS) than the APL interpreter, you must specify the *full* name of the file. For example, to read the file HEX which is in the folder SRC of the APL 90 distribution disk, you must specify :

```
)INPUT  APL90:SRC:HEX
```

Note that spaces are not permitted in these names. An an other example, to load the MLDFS using the INPUT file provided in the MLDFS folder, and since this file, LOADMLDFS, doesn't use full names, you need to first move the program APL 90 in this folder...

• The session variables Lnlt, Lsp, Lpw, are really “virtual space” variables, and are reset to a default value when you switch from a VS to an other.

• The objects A and W are recognized as mono-character identifiers. However, these names are not currently allowed as operands of the functions Lnc or Lex, or in system commands.

System errors/VS full

APL 90 detects many errors that are called “system errors”. However, most of these errors are inoccuous. Some fall in the category “WS FULL”. You need then to extend the virtual memory (with “)VMEXT”), and clean it (with “)VMCK”). Some other system errors come from damaged objects. The system will try to correct this by deleting the invalid objects. It is very unlikely that you lose much of the contents of the virtual memory, but in any case, it is a good idea to quit (remember, the active workspace will be saved), come back with a clean system, and immediately check the virtual memory (see : *Cleaning virtual spaces*).
In case of a repetitive system error, push the “shift-lock” key, select the item “Quit” of the "File” menu, and release the “shift-lock” key.

Macintosh bombs

Some real troubles may end with a Macintosh bomb. You may try the “resume” option, or, if the problem persists, use the “restart”. You may also try to push the “shift-lock” key, select the item “Quit” of the "File” menu, and release the “shift-lock” key.

Cleaning virtual spaces

As it was said earlier, it is good practice to clean up occasionally your virtual space with the command “)VMCK”. When you start APL90 with a virtual space where a VS full or a system error occured previously, when this virtual space was left without using the “)OFF” command or the “Quit” menu, or when you start APL with the option key pressed, you are prompted with a dialog box that proposes you to check your virtual space. Options of this dialog box are :
Cancel : do nothing special with this VS. This is the default option.
Check : this will run the “)CHECK” command.
Save : this will save the active WS as the WS#12, and provide you with a clear workspace. Previous WS#12 will be deleted.
Clear : this will clear both the active workspace and the WS#12.
Extend : this will extend the current virtual space. You should select this option if you got real “no space” troubles during the previous session.
Rebuild : this will try to rebuild your VS, losing as little as possible of your data.
Finder : this option lets you come back to the finder without changing anything in the virtual space.

Error messages
Some error messages are currently in french and cannot be simply localized. This includes labels of system errors, some warnings, and a few other messages. We intend to correct that sooner or later. Meanwhile, remember that, these last twenty years, we, french people, had to deal with software full of stupid english messages...

The APL 90 ZCODE

The following table displays the contents of the L□→ vector. Note that this contents may be changed in future versions of APL 90 if some kind of standardization of L□→ appears.

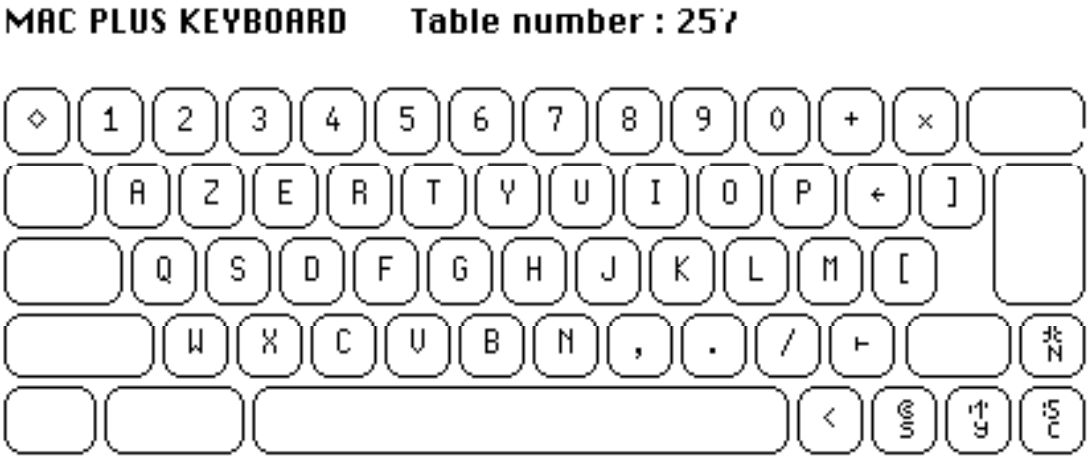
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0				0	@	P	`	p	⍵	⊂	≠	0	Δ	P	ë	¨
1			!	1	A	Q	a	q	∈	⊃	∨	1	A	Q	ê	≠
2			"	2	B	R	b	r	⍴	∩	∧	2	B	R	â	∘
3			#	3	C	S	c	s	↑	∪	×	3	C	S	î	○
4			$	4	D	T	d	t	↓	⊥	÷	4	D	T	ô	;
5			%	5	E	U	e	u	⍳	⊤	⌸	5	E	U	û	⍞
6			&	6	F	V	f	v	○	|	⍒	6	F	V	¨	⍱
7			´	7	G	W	g	w	*	←	⍋	7	G	W	ö	⍲
8			(	8	H	X	h	x	⍺	→	⌽	8	H	X	⍬	⍱
9			)	9	I	Y	i	y	⌈	⊣	,	9	I	Y	'	⍳
a			*	:	J	Z	j	z	L	⊢	⊖	≤	J	Z	Ξ	ε
b			+	;	K	[	k	{	∇	◇	/	∓	K	é	⍞	I
c			,	<	L	\	l	¦	Δ	¨	\	⍞	L	è	⍞	▒
d			−	=	M	]	m	}	∘	¯	⌽	⍞	M	à	⍟	⍠
e			.	>	N	^	n	~	'	≤	⍋	⍝	N	ù	⍛	⍎
f			/	?	O	_	o	▒	□	≥	⍱	⍒	O	ï	⍑	

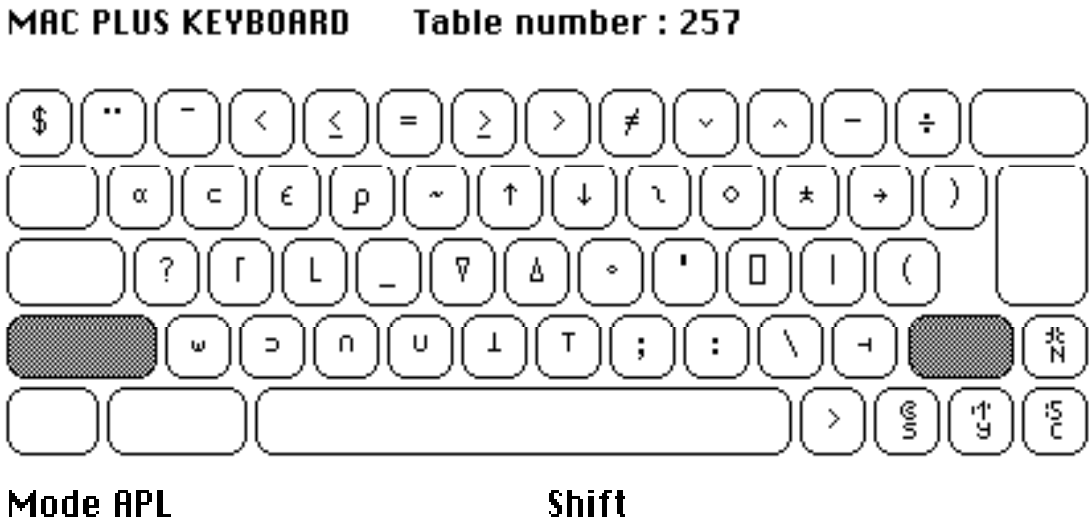
Description of the keyboards

You can see, print or modify any keyboard configuration with the workspace “UPDKB” of the virtual space “KEYB”. Connect to this VS and load the WS “UPDKB”. (This VS is provided on the distribution disk with a minimum size of 32 pages. To be able to use it, you will have to extend it to 48 pages the first time you use it. You can do that with “)VMEXT 16” after you connect to the VS).

You can type “DESCRIBE” to get a general description of the workspace. Functions with names ending in “HOW” are documentations of other functions. For example, “SHOWHOW” is the documentation of the function “SHOW”. This last function will draw the layout of the selected keyboard. You can exit from this function with the Command key, and get a MacPaint document from the screen, with Command-Shift-3. This will provide specially useful if you decide to modify the configuration of your keyboard with the “UPDKB” function.

The following tables are examples of keyboard configurations, and correspond to the layout of the APL Keyboard on a Macintosh Plus, with a french configuration, and on a Macintosh with the US configuration. Note that these tables reflect only an aspect of the keyboard, since you can also work in ASCII mode.





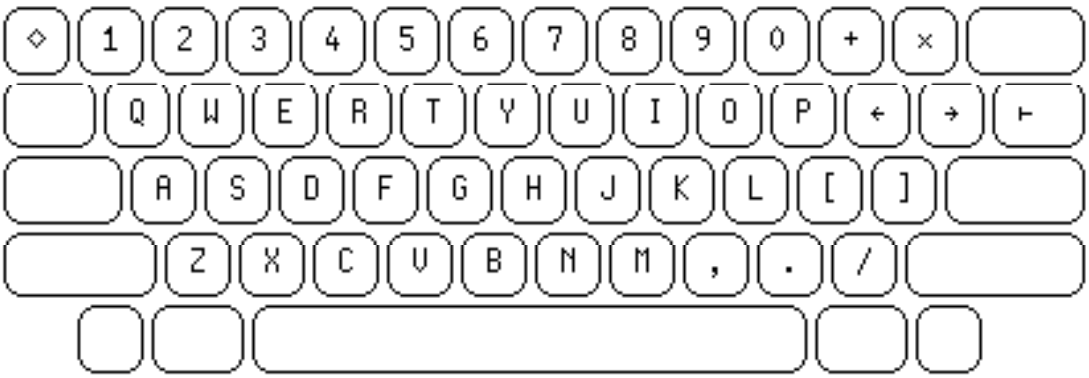
MAC PLUS KEYBOARD Table number : 257

Mode APL Option

MAC PLUS KEYBOARD Table number : 257

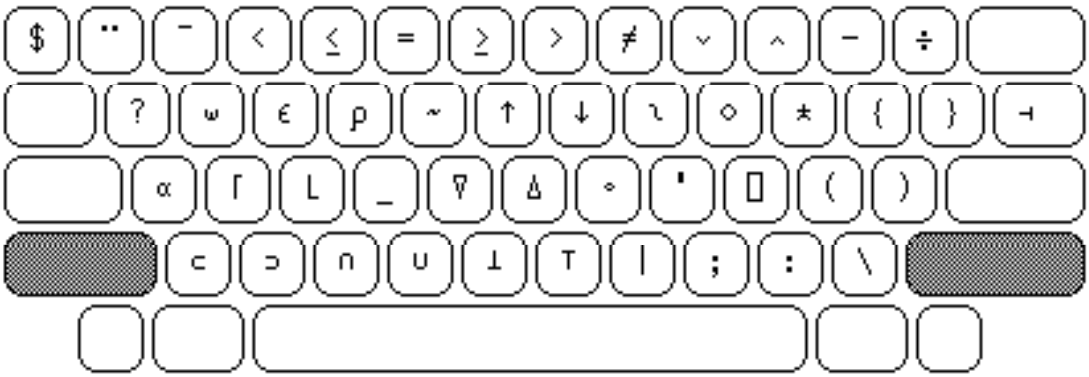
Mode APL Shift Option

U.S. KEYBOARD Table number : 259



Mode APL

U.S. KEYBOARD Table number : 259



Mode APL Shift

U.S. KEYBOARD Table number : 259

Mode APL Option

U.S. KEYBOARD Table number : 259

Mode APL Shift Option

References and documentation

- APL 90 : new directions in APL interpreters technology. J.J. Girardot, APL 85 ACM Congress.
- The syntax of APL, an old approach revisited, J.J. Girardot & F. Rollin, APL 87 ACM Congress.
- An object oriented extension to APL, J.J. Girardot & S. Sako, APL 87 ACM Congress.
- APL 90 Reference Manual, Sept. 86, 120 pages. [French]
- APL 90 Reference Manual, 1.05 update, January 86, 20 pages. [French]
- APL 90 Reference Manual, 1.10 update, March 86, 30 pages. [French]
- APL 90 : Writing MLDF for the Macintosh [to be announced]
- APL 90, Auxiliary processors for UNIX. 40 pages. [French]