APL 90 for the Macintosh

A Micro Manual

Experimental version 1.05

J.J. Girardot

March 87

What is APL 90 ?

APL 90 is a new implementation of APL. APL 90 was written in C, between 1982 and 1986, at the Ecole des Mines de Saint-Etienne (France), by Jean-Jacques Girardot, François Mireaux and Sega Sako (with the additional help of a lot of people). APL 90 is available under UNIX or UNIX-like systems, such as SMX on SM 90, SPIX on SPS7, ROS on SPS9, HPUX on HP9000, or Sun workstations. Current plans (still) include versions for Vax under VMS and for ATARI ST.

APL 90 Version I conforms to the ISO Standard. APL 90 Version 1.05 allows strand notation, nested arrays, error trapping, auxiliary processors, and some of the functionalities of IBM's APL 2. Next version of APL 90 will include user defined machine language functions, an APL-plus like file system, some additional goodies, and an object-oriented extension. It should be available near the end of 1987.

The Macintosh version can be used on 512K Mac or MacPlus, with or without the switcher. It has also been tested on the new Macintosh SE (although we haven't found any speed improvement over the Mac Plus), and on the Mac 2, (where it is real fast), but you will have to customize somewhat your keyboard. APL 90 is essentially an adaptation of the UNIX programs, and currently reflects little of the possibilities of the Macintosh. It is compiled using Mac C, and parts of APL 90 are © Consulair Corporation. Please consider this program as an intermediate product and a demo version. You cannot sell it or use it for commercial purposes.

For any information, contact :

François Mireaux
SYNC
12, Place Hotel de Ville
42000 Saint-Etienne France
Telephone (France : 33) 77 32 65 62

OR

J. Jacques Girardot
Ecole des Mines
158, Cours Fauriel
42023 Saint-Etienne France
Telephone (France : 33) 77 42 01 71

The future of APL 90

APL 90 Version 1.05 is the second release of APL 90 on Macintosh. We will try to provide the next version for October 1987. This version was made available partly because it was fun, and partly because we got some (limited) support from APPLE.

APL 90 on Macintosh may eventually turn into a commercial product. *In this case, all registered users will get the upgrade to this version for a very low price...*

Meanwhile, we would be glad to provide freeware versions of APL 90 for computers such as the Atari ST or the Amiga. However, this would be a lot of work, for which we have presently little time and no support. We would need for that at least a machine and a development system. Any interested constructor ? Please contact us...

Acknowledgements

APL 90 was developed with grants from the French government (Ministère de l'industrie) and from the ADI (Agence de l'Informatique).

I would like to thank people from **sync**, and specially François Mireaux and Bernard Amade, for the parts that they developed, and for their authorization for releasing APL 90 Version 1.05 on Macintosh as a freeware product. However, the APL 90 Reference Manual is the property of **sync**, and cannot be duplicated.

About this “Micro Manual”

This document is essentially a brief summary of the differences between APL 90 and traditional APL systems. It will therefore be useful only if you are proficient with APL, and should help you fill the gap between what you know and what is APL 90. You can order from **sync** the full APL 90 reference manual and additional material (however, reference manual is currently available in french only).

In case you need some introductory material to APL, we recommend “An Introduction to APL”, by S. Pommier, Cambridge Computer Science Texts #17, Cambridge University Press, ISBN 0-521-24977-5 (hard cover) or 0-521-27109-6 (paperback). This is a good and inexpensive book, which was furthermore written by the people who developed APL 90... You may also find yourself more comfortable with more expensive books, such as the classical “APL, An interactive Approach”, by Gilman and Rose, J. Wiley & Sons, ISBN 0-471-09304-1 or “APL, The language and its Usage”, by Polivka & Pakin, Prentice-Hall, ISBN 0-13-038885-8.

You may also wish to consult the APL2 Reference Manual, available as IBM publication SH20—9227—1, or “An introduction to APL2”, IBM publication SH20—9229—1.

Specific notions of APL 90

Workspace
The workspace (WS) is the place where user defined variables, functions and operators are kept. In traditional APL systems, you can save a copy of the contents of your workspace in the form of a disk file(“inactive workspace”), load such a file, or insert in your active workspace objects from an inactive one. This is the purpose of the commands “)LOAD”, “)SAVE” and “)COPY”. APL 90 extends the notion of workspace with the notion of virtual space.

Virtual space
APL 90 works with a virtual space (VS). This VS can be very large (up to 2^{53} bytes), and can hold as many workspaces as you want. Every VS contains a *root,* which holds a CLEAR WS, a LIB, and a current WS. The current WS is saved when you leave APL by the command “)OFF”, and loaded when you connect to a VS. You can save a copy of the current WS as an entry in LIB by a “)SAVE” command, or replace the contents of the current WS by a copy of a previously saved WS with “)LOAD”. Inside APL 90, you can switch to an other VS with “)VMEM *vsname*”. The contents of current VS, including the active workspace, is saved as with the “)OFF” command. Note that the disk which contains the active VS should not be write protected.

External virtual spaces
Other VS can be opened and closed with the commands “)VMOPEN” and “)VMCLOSE”. A VS is characterized by its external name, which is the name of the file that contains the VS, and which you use in the “)VMOPEN” command, and an internal library number, which is used in the commands as a reference to the virtual space, and which you choose when you create the VS. The “)LIBS” command prints the identification of opened VS, the first entry being always the active VS. Such VS are read-only, and can be used like traditional APL libraries.The commands “)LOAD”, “)COPY”, “)PCOPY” operate from any virtual memory. The commands “)SAVE”, “)STORE” and “)PSTORE” operate exclusively inside the current virtual memory. Here is an example of session :

```
      )LIBS
32    MEMAPL     32 Pages,     78432 Bytes
      )VMOPEN MLDS
100   MLDS
      )LIBS
32    MEMAPL     32 Pages,     78432 Bytes
100   MLDS

      )LIB 100
```

```
      QUICK      SYSFCT
                 )FNS  100 –WS  SYSFCT
      COMPOSE  FUN        G          TEST
                 )COPY  100   SYSFCT   TEST
      SYSFCT  (Saved  16.03.40  86/11/18)
                 )VMCLOSE   100
```

Note that you can open a VS only if its library number is different from any other currently opened VS.

Ascii files
To read or write data outside of a VS, you can also use the commands “)INPUT” and “)OUTPUT” which allow you to read and write sequential ASCII files.

Starting with APL 90 on Macintosh

Icons
The following icons correspond to files related with APL 90. The first one is the APL 90 program itself. The file named “KEYB” is a virtual memory where some APL objects have been created by APL 90. You always need at least one such virtual memory to work with. “APL Source” is a text file that can contains APL instructions or commands. Such a file if of type 'TEXT', and therefore can be created by any text editor, or from APL with the “)OUTPUT” command. The instructions can be executed with an “)INPUT” command. The last file is MLDF (Machine Language Defined Function), that can be loaded with a “)MLDFLOAD” command, and used like a user defined function. Since such a file is currently created with the MAC C development system, and you need some sources of APL 90 to produce them, we don't expect users to create such files (however, a MLDF development kit should be available in the near future). Some MLDF that provide an interface with Quickdraw can be found in the MLDF Folder, together with a text file to load them in the system.

Configuring your disks
To run APL 90, you need either a 512k “old” Mac, with two 400k drives, or a Mac Plus with a 800k drive. However, APL 90 is a rather large program, and it uses some system resources. Furthermore, you need to install the file MEMSYS in the same folder as APL 90, and *you also must have 16k available on the same disk* (this space is used when you invoke the command “)VMCK”. Here are some tips for using APL 90.
Case 1 : a 512k Mac and two 400k drives. Create one disk with APL90 and MEMSYS as the only files. Create an other disk with MEMAPL. You can use disks formated with “Fat Disk Maker” to get a few additional k bytes. The MEMAPL VS can be extended with the command “)VMEXT”.
Case 2: a Mac Plus and a 800k drive. Create a disk with APL90 and MEMSYS. You can put other files on the same disk, such a the system files or MEMAPL. Be sure to left at least 16k free. The distribution version of APL90 is configured to work on a 512k Mac, or on a Mac Plus with a 512k cache. APL90 will use only 80k bytes of memory in RAM to cache its own virtual memory. This corresponds to 20 pages of a VS. On a Mac where more memory is available, you can (and should) increase this value to at least 60 or 80 pages (240 to 320k bytes). You can do that from APL with the following statements (if you want a 80 pages APL cache) :

```
      x[Lresource KconfK 0 5
      x;1'[80
      x Lresource KconfK 0 5
   0
```

(you should get a 0 as a result of the last statement). The disk that contains the interpreter should not be write protected. Now you can “Quit”, and the next time you run APL90, it will use a 80 pages cache in memory instead of 20. Of course, such a modified system is likely to produce a bomb 15 on a 512k Mac. (What you you have done here is read the resource CONF of number 0 of APL90, change the cache size from 20 to 80, and rewrite the resource. This also can be done with any resource editor, such as ResEdit or REdit.)

Case 3: a hard disk is connected to your Macintosh. Create somewhere a folder with APL90 and MEMSYS. You should have no problem creating very large VS on your disk. If you have more than 512k of memory, it is a good idea to increase the size of the APL cache, especially if you intend to use very large VS. See “case 2” above.

Running APL
Run the application “APL90”. It first looks for a virtual memory, with a default name of “MEMAPL”. If no file with that name is present on the disk, APL 90 will prompt you with a standard dialog box to select a virtual space. You can select “Cancel” if you don't want to open a virtual space, or if none is available. However, you won't be able to work with APL until you have opened one.

You can create a new virtual space with the ")VMINIT" command. To reinitialize or create a virtual memory, use :

)VMINIT *name size* –IDEV *num*

where *name* is the name of the file which will represent the virtual memory, *size* is a number of 4k bytes pages, and *num* is the library number of the virtual space. *size* should be at least 32 and can be up to 33554432 (this would be 137438953472 bytes, but we actually never tested virtual spaces of that size ; however, 20 or 30 megabytes virtual spaces use to work correctly in the UNIX version). *num* is the library number of the VS, and is an integer between 32 and 32767, with a default of 32. For example :

)VMINIT MYMEM 96

allows you to create a 384 k bytes virtual space on a 400k empty floppy disk. Larger virtual memories can be created on 800k floppies or hard disks. Finally, you can at any time use the command “)VMEXT *size*” to add *size* pages to the current virtual memory.

To clean the current virtual memory, use :

)VMCK

You need at least a 16k bytes free space on the disk containing the interpreter.

Finally, the command “)VMEM” with no parameter will print informations about the current virtual space.

The Macintosh screen

The Macintosh screen is divided in three parts : the menu bar, the display, and the indicators. The display window is of fixed size, and cannot be moved. This window always shows the last lines exchanged with APL 90 during the session.

Menus

File
The only functions currently available are :
Quit : save current contents of the virtual space, and returns to Finder.
Abend : Try to close some files, and returns to the Finder. *Abend should not be used, unless "Quit" has repeatedly failed.*
Panic : returns to Finder. *Panic should not be used, unless the other commands have repeatedly failed.*

Edit
Undo is not used in APL 90, and is provided for desk accessories that support this command.
Cut, *Copy* and *Past* can be used to locally edit parts of the screen. They should not be used on the APL input line.
Append will add the current selection at the end of the input line, or, if the selection is empty, add a "return". It is a very convenient way to build APL expressions.
Clear will clear the screen.

Typo
The commands of the typo menu provide variations on the size and the aspect of the displayed characters.

Keyboard
This menu can be used to select a different keyboard configuration if the current one is not convenient.

Controlling the display

Typing APL expressions
The whole APL 90 character set is available on the keyboard, and most of the times, to a single key correspond different, usually related, characters : for example, the G key, when shifted, generates the character G, with the option key the character €, and with both keys the character '. The keyboard can also be interpreted in ASCII mode. The same key would then produce the characters g, g, G and '. The overstrike characters, such as € and ', can also be generated the traditional way, by typing one char, a backspace, then the other char. Backspace and space can be freely used to move the spot back and forth, and do not destroy characters.

You always enter APL expressions on the last line of the display, and your input cannot be larger than the physical line. Cut, Copy and Past do not work on the input line, and should not be used. However, you can locally edit the input line with the following control characters :

Control characters
Space : moves cursor 1 character to the right.
Backspace : moves cursor 1 character to the left.
Tab : shifts between APL and ASCII character sets.
Command+Space : inserts a space to the right of the cursor.
Command+Backspace : erases the character at the left of the cursor.
Left and Right arrows move the cursor one char to the left or to the right respectively.
Option+Left or Right arrows move the cursor 8 chars to the left or to the right.
Up and Down arrows move the cursor to the beginning and the end of the line respectively.

Command+F: adds at the end of the input line the selected area, or sends “return”.
Command+J: erases right part of the input line.
Command+N: selects APL character set.
Command+O: selects ASCII character set.
Command+S: stops temporarily the output.
Command+Q: restarts output stopped by Command+S.
Command+(“.” or “,” or “;” or “:”) : soft interrupt.
Command+Option+(“.” or “,” or “;” or “:”) : hard interrupt.
Shift-Lock, while depressed, will lock APL in the event loop. This is a convenient way to suspend a running program.

The indicators

The lower part of the Macintosh screen displays 8 indicators. From left to right :

The debug indicator
This indicators displays the debugger in use with APL 90. It can be :
APL 90 : no debugger is selected.
MacsB or *Nub* : in case of error, dialogs windows will offer the possibility to branch to the corresponding debugger.
Trace : no debugger is selected, but a dialog window will appear after each error detected by APL.
Clicking on this indicator will redraw the screen.
You can switch from one mode to another by just shift-clicking on the indicator. Of course, MacBug or Nub should be selected only if the corresponding debugger is installed in the system.

Keyboard mode indicator
This indicator displays either “ASCII” or “APL”, to indicate whether the keyboard keys are interpreted in ASCII mode or APL mode. You can switch from one mode to the other by clicking on the indicator, using the Tab key, or æo (switch to ASCII) and æn (switch to APL).

Status Indicator
This indicator displays the current status of the interpreter. It can be :
Input : the system is waiting for an input from the user.
Output : the system is currently displaying some result.
I/O : the system is doing some other kind of input or output.
Hold : output has been suspended by a æs. It can be restarted by a æq, or clicking on the status indicator.
Work : APL is doing some computation. It can be interrupted by a “soft” interrupt or a “hard” interrupt.
Wait : APL is in a wait state (execution of a ⎕dl). It can be interrupted by clicking on the status indicator, or by a “soft” or a “hard” interrupt.
EvLoop : the shift lock key is depressed, and APL is trapped inside the system main event loop.
Clicking on the indicator will interrupt the wait or release the hold status.
Shift-clicking on the indicator will generate a “soft” interrupt .
Option-clicking on the indicator will prompt you with a dialog box to select some action.

Numeric indicators
The next three indicators display numeric information : current position of the spot in the last line, maximum size of this line (or size of the selection if any), and APL code of last entered char.

Mouse position
This indicator displays the mouse position in the APL window, either in pixels or in characters. Click on the indicator to switch from one mode to the other. The indicator can be hidden by option-clicking on it.

Timer
This indicator displays the time or a timer. Click on the indicator to switch from one mode to the other. The timer can be reset by shift-clicking on it. The indicator can be hidden by option-clicking on it.

Windows

Although APL 90 uses currently only one display window, you can use desk accessories with their own window. You can select any system or application window by clicking inside its contents, and move it by clicking in its title bar and displacing the mouse.

An inactive window can be moved by holding down the command key before clicking in its title bar. A window can also be sent in the background by holding down the command and option keys before clicking in its contents. This is a very convenient way to make reappear a window belonging to a desk accessory.

Interrupts

Soft interrupts
A soft interrupt can be generated by Command-. , Command-, , or Command-: , or by shift-clicking the status indicator. A soft interrupt will halt an output, interrupt a Ldl function (processing will continue in this case, the result of Ldl being possibly less than the parameter of the function), and halt a user defined function at the end of the statement being currently executed.

Hard interrupts.
A hard interrupt can be generated by Option-Command-. , Option-Command-, , or Option-Command-: , or by Option-shift-clicking the status indicator. A hard interrupt will halt an output, and halt a user defined function at the end of the statement being currently executed.