A Review of APL★PLUS III for Windows

Dick Bowman
Dogon Research
2 Dean Gardens
London E17 3QP England
Tel: +44-81-520-6334
E-mail: `bowman@apl.demon.co.uk`

Manugistics' APL★PLUS III is the latest incarnation of what is possibly the longest lineage of APL interpreters for the PC; it supersedes the previous APL★PLUS II and joins Dyalog's APL/W and ISI's APLIWIN as an uncompromisingly Windows-only product.

Installation

This review centres around Release 1.1 (at the time of writing Release 1.2 had been announced but not yet received) which comprised three 3.5" disks and a two-inch thick set of manuals (User Manual and Reference Manual); installation is via the now-customary Windows install procedure, resulting in an additional 750-Kb used in my Windows directory and just over 5-Mb in the new APL★PLUS III directory. Note that APL★PLUS III adds `Win32s` files to your installation. I have since upgraded to a more recent version of `Win32s` (needed for another package) without problems; my sense is that `Win32s` is in a rather fluid state at this time.

First Impressions

Following one's natural instinct, the first thing that comes to mind with a new APL interpreter installed is to fire it up and type `1+1` (which has been known to fail with some interpreters in the past...).

What you first see is as in Figure 1; a typical Windows screen with menu, toolbar and some blank space to use. Starting up with all the defaults means that the display uses the recommended screen font (there are three more TrueType fonts included). Although better than some, this is not my favourite APL screen font, but as character mappings are different between vendors it will have to suffice.

Keyboard usage is "classic" APL in the Unified keyboard sense; an old-time APLer will be fairly comfortable with the lie of keys under fingers and thumbs; although any lingering over the `Alt` key will not activate menu options, which may confuse the non-APLer. If all you use is one version of APL, there should be no problems—anyone using two or more versions of APL may be frustrated from time to time. I wish we could agree on a single keyboard mapping.

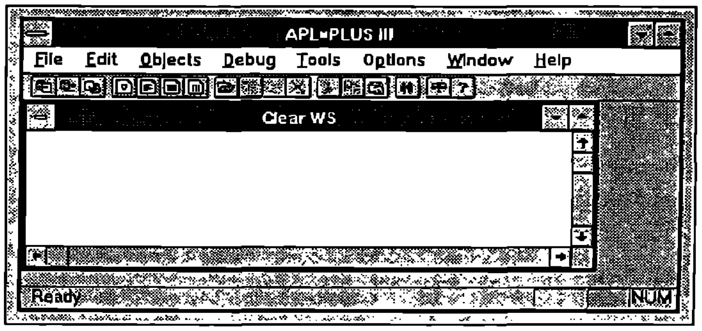


Figure 1. Initial Screen Shot

Compatibility

I don't really know where the vendors are going on the compatibility issue; as a long-term APL2 user, I found it hard to get used to the different meanings of visually-identical code when dealing with nested arrays in both APL★PLUS II (and now III) and APL/W—the problems of monadic ↑ versus ⊃, for example. I'm happy to see that this is beginning to be resolved, with APL★PLUS III adopting APL2 conventions by default (system command *)EVLEVEL* gives programmer control over precise behaviour).

But this is still some way from being a plug-and-play situation; if what the user wants to do is to download some mainframe APL2 application and run on APL★PLUS III, there's probably still a lot of work to do—even leaving aside the near-certainty that user interfaces will deserve substantial rethinking.

Some APL2 features not present in APL★PLUS III include defined operators, partitioned enclose (although this is emulated by *⎕PENCLOSE*), *⎕EC*, *)OUT* and so forth.

Manugistics provide past users with a full set of tools for upgrading from previous APL★PLUS versions; they also include tools for importing APL2 workspaces. Although there are no intrinsic facilities for importing Dyalog APL workspaces, it was a relatively simple job to create a workspace to do the job (as usual, the most time-consuming part was making sure that the character mappings were correct). Naturally—imports from other workspaces will normally need some further manual amendment.

Timing Tests

Although it's not my intention to start a shoot-out between vendors and followers of different implementations, it is instructive to look at some simple timing comparisons; the table below shows the results of running through Gregg Taylor's battery of timing tests (see *APL Quote Quad*, Vol. 21, Number 1, page 20, for a description of the tests). It seems fairer to do it this way (all tests done the same day on the same environment) than to leaf through past benchmarks. Of course, if performance is critical to your application, you will have to perform your own tests.

	Dyalog APL/W 6.3.2	Manugistics APL★PLUS III
Integer Add	16.5	14
Floating Point Add	24.5	33
Integer Multiply	43.5	49
Floating Point Multiply	33	50
Index	2	4
Character Compress	3	3
Integer Compress	3	0
Integer Plus Reduce	0	3
Integer Max Reduce	0	0
Boolean Scan	22	593
Matrix Rotate	0	6
Character Transpose	11	5
Integer Transpose	11	6
Partition [See Note]	16.5	20
Rho Each	16.5	14
Vector Comparison	0	0
Integer Sort	14	8
Boolean Comparison	0	3
Iota	44	22

Note: APL★PLUS III reports a *NONCE ERROR* for dyadic enclose, test uses ⎕*PENCLOSE*

The conclusion I draw from this is that neither APL★PLUS III nor APL/W have a decisive speed advantage, except possibly for followers of Mr. Langlet's interest in Boolean scan (this anomalous result worried me for a while, but referring back to Gregg Taylor's review cited above, we see a similar pattern holding for APL★PLUS II).

Significant Omissions

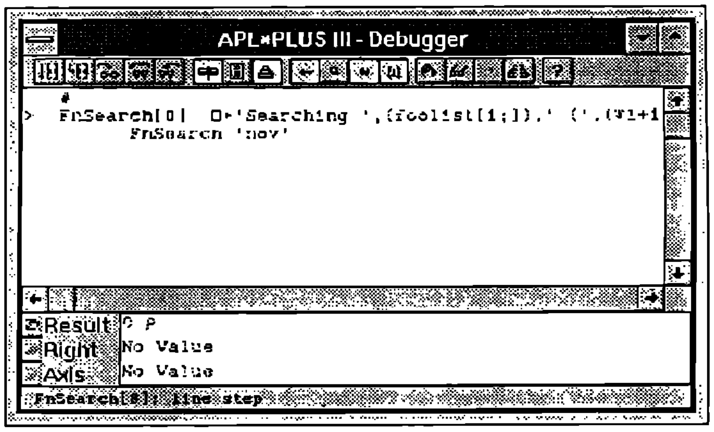
Manugistics have bitten the Windows bullet quite decisively with this version; all of the ⎕*WIN* facilities of earlier versions are omitted, as are the ⎕*GRAPHICS* features. Clearly, there are some tough decisions to be made by users of the DOS product.

What surprised me more was that ⎕*EDIT* is also omitted.

Programming Environment

APL★PLUS III is an MDI application; the window normally contains the traditional workspace log, and additional child windows may be opened to hold edit sessions, debug windows and so forth.

Function breakpoints, variable watchpoints may be set and execution traced via the debug window (which allows single-stepping, argument/variable/result inspection, and so forth). An example of the debug window is shown in Figure 2. The debug window—like the application code—operates independently of the MDI parent form.



```
 #
> FnSearch[0]  ⎕←'Searching ',(foolist[1;]),' (',(⍕1+1
       FnSearch 'nov'
```

```
Result  ⍝ P
Right   No Value
Axis    No Value
FnSearch[8]: line step
```

Figure 2. The Debug Window

One curiosity is that while it's only possible to have one workspace open at a time, any windows that you open from that workspace will remain open after a new one is loaded. So you can open a function for edit from workspace *A*, leave it on-screen while you load workspace *B* and save it there. I wonder about this....

The programming environment is a significant advance on the facilities offered by APL★PLUS II—it takes time to become familiar with it, and full benefit is only achieved when this time has been spent. The environment is also customisable—you can add your own tools (which may, of course, be APL code) to the menu bar.

Language Extensions

The big language news with APL★PLUS III is the inclusion of control structures; so far as I am aware, this is the first time such elements have been included in a widely-available APL product.

What APL★PLUS III control structures allow the programmer to do is to incorporate the "conventional" constructs of `if-then-else`, `do-while`, `do-until`, and so forth into APL functions; the mechanism by which this is implemented is by introduction of a set of keywords which begin with a colon. Here's an example:

```
:while i<ρspos
     i←i+1
     chunk←len[i]↑spos[i]↓tv
     chunk←(~chunk ⎕ss '''''')/chunk
     name←⎕def '∇',chunk
     :if (0=1↑0ρname)∧0≠2|i
          JunkIn←JunkIn,⊂chunk
     :endif
:endwhile
```

As you can see, code readability is increased, because we have an extra opportunity to discard housekeeping clutter. It's also good to see that APL★PLUS III respects white space. The experienced APL programmer clearly has a learning curve to climb here—I freely confess that I originally wrote the code fragment above in "traditional" style and converted it later as an experiment.

On the whole, I much prefer code which uses these constructs than code which doesn't— Manugistics have advanced the state of the APL linguistic art. But I have reservations. The first of these is code portability; unless and until the other vendors adopt these structures, we have reduced code portability (you can move your old-style code into APL★PLUS III, but you have to work at getting new-style code out). I note that J Release 2 offers some similar facilities, but I'm not aware of anything else; of course, the door is very much open to either consensus or the more traditional APL solution of everyone doing their own thing.

The other reservation I have about these control structures is what happens when they get into the hands of naïve users; what I fear is proliferation of scalar/looping code which does not take advantage of array operations. Agreed, they could do it before, but even the naïve found that writing explicit branch statements was tedious.

Designing and Using the Windows Interface

As we've come to expect from this product line, the Windows interface is implemented through a new quad function, ⎕*WI*, which is quite heavily loaded with options.

One of the first tasks which the curious will want to perform is to design and use a Windows form, and this is most readily done using the]*WED* user command (user commands are an APL★PLUS feature which does not seem to have been emulated by other vendors—a selection are included in the package). Invoked with the syntax "]*WED formname*", the appearance is as shown in Figure 3. Familiar enough to the Visual Basic user, if not hugely endowed with tools (what this does illustrate is an orientation of APL★PLUS toward character GUI applications—if this isn't a contradiction in terms),]*WED* offers facilities for building menus and testing forms; the end result after saving is

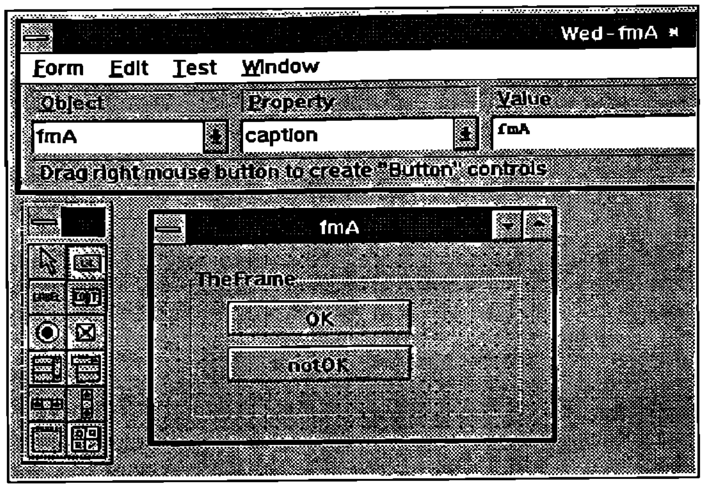


Figure 3. The Form Editor

that a variable called "*formname_def*" appears in the workspace. From here, the choices proliferate; the programmer may use the variable within applications, changing it with]*WED* until stable, or produce a function definition from it (the Manugistics manual is quite emphatic that programmers should not try to manipulate the variable directly—although some may wish to do so). Making the equivalent function is the work of another user command (]*WMAKE*), which yields the function below:

```
fmA_Make;x;⎕wself
⍝∇fmA_Make -- Created 8/31/94 at 20:01:39
'fmA' ⎕wi 'Delete'

⎕wself←'fmA' ⎕wi 'New' 'Form' 'Close'
⎕wi 'caption' 'fmA'
⎕wi 'extent' 7.75 37.25
⎕wi 'where' 9.375 20

⎕wself←'fmA.f1' ⎕wi 'New' 'Frame'
⎕wi 'caption' 'TheFrame'
⎕wi 'where' 0.5 2 5 25

⎕wself←'fmA.f1.bn1' ⎕wi 'New' 'Button'
⎕wi 'caption' 'OK'
⎕wi 'where' 1 3 1.5 10

⎕wself←'fmA.f1.bn2' ⎕wi 'New' 'Button'
⎕wi 'caption' 'notOK'
⎕wi 'where' 2.5 3 1.5 10
```

In this example, the form is quite inanimate—what we need to do (and could have done with]*WED*) is to associate code with events happening to controls:

```
fmA_Make                               ⍝ Make the form
'fmA' ⎕wi 'Wait'                       ⍝ Make it visible and
                                         wait
'fmA.f1.bn2' ⎕wi 'onClick' 'ouch'      ⍝ Run <ouch> if <bn2>
                                         is clicked
Yaroo, Bunter - that hurt              ⍝ What <ouch> says
'fmA' ⎕wi 'Close'                      ⍝ Close the form
'fmA' ⎕wi 'Delete'                     ⍝ And delete it
```

What this truly awful example shows is that (in the session) we can even choose to define event actions while the form is current and waiting; when encapsulated into a function, it's important to define the event activity before opening the form (I'm not quite sure that this is correct—it would seem useful to be able to redefine responses on the fly). Of course—normal code would define a lot of event handling (and other form specifics) inside *fmA_Make*.

Notice that APL★PLUS III is quite finicky about upper and lower case—an error message is the result of using `<close>` instead of `<Close>`, for example.

The great strength of]*WED* (apart from making form design and modification very simple), is that it has zero footprint in the workspace; when you've finished with it, absolutely nothing except the required end-product remains. This is, to me, one of the great advantages of User Commands in the APL★PLUS product line.

What the careful reader will also notice in the example above is that Manugistics are infusing a lot of object-orientation terminology (if not more) into APL; forms are objects, they have properties and respond to events. Also, notice how "real" graphics sit not quite so easily into this framework; you can't use]*WED* to put decoration onto your forms. There are graphical objects, and you can apply appropriate methods to them, but they're special and the programmer has to consider their requirements separately (for example—if a form containing both a button and a graphical image is covered by another window and then uncovered, the button is redrawn automatically, but the graphical image has to be redrawn explicitly). It can all be done, but it's harder than it needs be—I think.

DDE

Manugistics' approach to DDE is very educational in the message it sends to us about their designers' philosophy for the future direction of APL. Let's examine a very simple example of using DDE to retrieve data from an Access database:

```
z←TestSQL;fmA;junk
junk←'fmA' ⎕wi 'New' 'Form' ('visible' 0)
junk←'fmA.Ed' ⎕wi 'New' 'Edit'
'fmA.Ed' ⎕wi 'ddeTopic' 'MSAccess|audio2;
                  SQL Select [Title] from Books'
'fmA.Ed' ⎕wi 'ddeItem' 'data'
'fmA.Ed' ⎕wi 'ddeMode' 'cold'
junk←'fmA.Ed' ⎕wi 'DdeRequest'
z←'fmA.Ed' ⎕wi 'text'
junk←'fmA' ⎕wi 'Close'
junk←'fmA' ⎕wi 'Delete'
```

What's most striking here is that APL★PLUS III uses forms, and controls on forms, as the vehicle for implementing DDE. This seems artificial to me. What I feel as a long-term, hard-core APL user, is a need for a more direct path between the foreign application and my APL workspace; in other words, the DDE interface which feels right for my APL application is the one that would be provided by generalisations of the `TestSQL` function—I don't feel that I get much benefit from having to provide all of this detail.

This approach to DDE mirrors the approach found in other languages—most notably Visual Basic; what it means (probably) is that a Visual Basic programmer is likely to find APL★PLUS III a more welcoming environment than one which took a more purist (or isolationist) APL approach to DDE.

When the application is being used, this probably doesn't matter a great deal—as you can see above, the form can be made invisible. What we might guess is that this is a deliberate move towards making APL seem less alien to the "conventional programmer," a conclusion which is supported by seeing control structures implemented through keywords rather than the more symbolic approach which has been aired elsewhere.

I have not been able to stress-test the DDE interface; it feels fast and stable, but I have not (as yet) put a great deal of data through it.

Summary

There's a great deal more in APL★PLUS III than I've been able to cover in this quite brief review—I believe that the programming environment, GUI interface and DDE support are the most significant new features introduced.

Although I haven't been able to subject APL★PLUS III to a great deal of stress-testing, my sense is that this is a solid and reliable product which may be used to build sturdy, well-behaved Windows applications. I'm a little surprised to find that Manugistics have not seen fit to include such things as a help compiler in the package, but they're not alone in this (it's rather surprising to see these things omitted when they come as part of much more inexpensive "other language" products).

APL★PLUS III obviously issues a challenge to users of the earlier members of this interpreter family by being so clearly targeted at production of Windows applications.

As a programming language, we see welcome extensions such as the new control structures (although they would be much more welcome if other vendors also offered equivalents); the programming environment is also much improved over earlier versions. To derive fullest benefit from this environment, it will be necessary to spend some time investigating how the tools are best used, and unlearning old practices.

Aside from graphics, the only feature that appears to be missing is any form of OLE support;

again—if this is your requirement, you may have to look elsewhere for the solution. It's always unreasonable to try to predict the future; Version 1.2 has been announced as I write this, and includes support for VBX controls (amongst other extensions)—it seems reasonable to expect that the product will continue to be enhanced and become even more useful. ■

●

Manugistics Responds:

Paul Clements
Business Manager, APL Products
Manugistics, Inc.
2115 East Jefferson Street
Rockville, MD 20852 USA
Tel: 301-984-5324
E-mail: `paul@manu.com`

Dick Bowman's article is a good discussion of many of the technical features of APL★PLUS III for Windows and also of the design issues that went into building it. As he states, version 1.2, with VBX support is now shipping. In addition to the Text keyboard, version 1.2 includes an option for the traditional APL keyboard. We have also created a new object class, called "Picture," that provides, among other capabilities, a persistent image, so that a graphical image is redrawn automatically.

Other technical points raised in the article:

- APL★PLUS III provides the user commands]*IN* and]*OUT* to correspond to the APL2 system commands)*IN* and)*OUT*.
- The function *Wedit* in the *WINDOWS* workspace replaces ⎕*EDIT* as a developer's tool. The new graphical user interface allows programmers to provide the same functionality in an application with a Windows look-and-feel.
- Many of our users have told us they particularly like the ability to have an edit session open, load another workspace, and then save the object into the new workspace.
- It is indeed possible to redefine event handlers, like "`onClick`", on the fly. You can redefine not only responses, but object characteristics or objects themselves, on the fly.

Manugistics believes in minimizing incompatibility, as Dick mentions with our)*EVLEVEL* support. However, it is unrealistic in a competitive market environment to wait for consensus before providing new features. Plug-and-play is theoretically appealing, but restricts innovation.

Just as we unilaterally adopted some APL2 conventions, we would be pleased if our control structures become a *de facto* standard, or at least a target for consensus. The vast majority of our users love them, and we are not seeing bad coding as a result. In fact, it may be easier to detect and correct scalar/looping code written with control structures than that written without them. We are most pleased that Dick agrees we *"have advanced the state of the APL linguistic art."* That was one of our goals.

He is also correct that the VB-like approach to DDE and]*WED* are examples of using market-accepted approaches to Windows programming in order to close the gap between our community and the rest of the world. The future promises to hold even more communication and interface standards, e.g., ODBC, and the APL world cannot afford to remain isolationist, no matter how good a tool this language continues to be. ■

3-D Graphics

1995 Symposium on Interactive 3-D Graphics

April 9 – 12, 1995
Monterey Conference Center
Monterey, CA USA

Sponsored by ACM, the Association for Computing Machinery and Special Interest Group GRAPH.

For further information, contact:

Michael Zyda
Naval Postgraduate School
Code CS/Zk
Department of Computer Science
Monterey, CA 93943-5100 USA
Tel: 408-656-2305
E-mail: `Zyda@trouble.cs.nps.navy.mil`

SIGPLAN '95

ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)

June 18 – 23, 1995
Hyatt Regency
La Jolla, CA

Sponsored by ACM, the Association for Computing Machinery and Special Interest Group PLAN.

For further information, contact:

David Wall
Digital Western Research Lab
250 University Avenue
Palo Alto, CA 94306
Tel: 415-617-3309
E-mail: `wall@decwrl.dec.com`