—by Dick Bowman
London, England

THIS HAS NOT BEEN an easy review to produce. Over the years, I have watched J emerge and my reaction—as a long-standing "pure" APL application developer—has slowly changed from scepticism to curiosity. One of the intriguing aspects of J is the way in which it is perceived in a confrontational manner; a sense in which J is sometimes viewed as "not APL."

This is a controversy I want to avoid in this review. I have long been eager to look at J as a "product," or a tool for building applications and the release of J2 for Windows has made this an achievable goal.

The product reviewed is the "personal" version 2.03 (the personal edition is said to be identical to the professional version except that it does not offer facilities for building runtime applications), there have been new releases subsequent to this one which may address some of my observations, I believe the current version number is 2.05.

Installation

J is delivered as a set of four disks (one for J itself, one for ODBC and VBX setup, the other two containing ODBC drivers); there are two manuals—the "Introduction and Dictionary" and a "User Manual."

The Introduction and Dictionary is both dense and terse—while the User Manual puts quite some emphasis on building Windows applications with J.

Installation was straightforward—there is even a J font—which came as something of a surprise.

I installed J on two machines, one a 486-DX/33 with almost-ample everything, the other a 386-SX portable which is becoming almost marginal for doing real work on. Both machines run Windows 3.1 with Win32S; the main machine runs OS/2 for some of the time.

Starting to use J felt very like starting to use APL did—a great need to explore how the language elements interacted; the sense of "strangeness" is compounded by the way that some of J's nouns resemble familiar APL functions but are subtly different.

Problems

Running the standard `profile.js` script at start-up on the portable always seemed to hang; when I interrupted the process the ensuing interchange with DOS/Windows always told me that `POINTER.DLL` had stopped responding to the system. I put this down to a three-way conflict with fancy mouse software and `WIN32S`. Shutting down `POINTER.DLL` got me running again.

My other major problem was that "`<.`" in a script operating on a complex argument locked everything up solidly (this was on the main machine) to the extent that Ctrl-Alt-Del needed a reboot. I corrected the code and have not been able to reproduce the problem outside the context of the wrongly-coded application.

```
cm=.50 100$cv=.(?1000$11){' ABCDEFGHIJ'
im=.50 100$iv=.?1000$500
fm=.50 100$fv=.iv+0.01*1|.iv
bm=.50 100$bv=.?1000$2
res=.<'int add             ',":1000*(2) 6!:2 ('aa=.im+im+im+im+im')
res=.res,<'fp add            ',":1000*(2) 6!:2 ('aa=.fm+fm+fm+fm+fm')
res=.res,<'int mult          ',":1000*(2) 6!:2 ('aa=.im*im*im*im*im')
res=.res,<'fp mult           ',":1000*(2) 6!:2 ('aa=.fm*fm*fm*fm*fm')
res=.res,<'index             ',":1000*(2) 6!:2 ('aa=.bv#i.$bv')
res=.res,<'char compress     ',":1000*(2) 6!:2 ('aa=.bv#cv')
res=.res,<'int compress      ',":1000*(2) 6!:2 ('aa=.bv#iv')
res=.res,<'int plus red      ',":1000*(2) 6!:2 ('aa=.+/iv')
res=.res,<'int max red       ',":1000*(2) 6!:2 ('aa=.>./iv')
NB.  res=.res,<'boolean scan ',":1000*(2) 6!:2 ('aa=.(-.@=/\"1) 10 10{.bm')
res=.res,<'matrix rotate     ',":1000*(2) 6!:2 ('aa=.>(i.50)|.&.><"1 cm')
res=.res,<'char transpose    ',":1000*(2) 6!:2 ('aa=.|:cm')
res=.res,<'int transpose     ',":1000*(2) 6!:2 ('aa=.|:im')
res=.res,<'vector of vectors ',":1000*(2) 6!:2 ('aa=.(<"1) cm')
res=.res,<'partition         ',":1000*(2) 6!:2 ('aa=.bv <;.1 cv')
res=.res,<'shape each        ',":1000*(2) 6!:2 ('aa=.$&.>aa')
res=.res,<'vector compare    ',":1000*(2) 6!:2 ('aa=.cv *./ .= cv')
res=.res,<'integer sort      ',":1000*(2) 6!:2 ('aa=.iv/:iv')
res=.res,<'boolean compare   ',":1000*(2) 6!:2 ('aa=.bv+.1|.bv')
res=.res,<'iota              ',":1000*(2) 6!:2 ('aa=.iv i.<.  fv')
res=.>res
res 1!:2 (2)
```

Figure 1: A conversion of Gregg Taylor's APL benchmarks

Timing

A critical consideration for non-trivial applications, and a key question is whether we should apply the same benchmarks to J as to long-established APL interpreters. It seemed most valuable to do so, even though the issue is clearly raised whether the language elements exercised are the most appropriate (one thing that strikes home over and over again is that J offers a non-trivial paradigm shift for the long-established APLer).

Conversion of the established Gregg Taylor benchmarks gave the following results:

int add	165
fp add	55
int mult	140
fp mult	80
index	30
char compress	0
int compress	0
int plus red	0
int max red	0
matrix rotate	165
char transpose	25
int transpose	30
vector of vectors	25
partition	305
shape each	410
vector compare	30
integer sort	25
Boolean compare	0
iota	30

My script for this is shown in Figure 1.

Comparing these with recent reviews of Dyalog APL/W and APL★PLUS III it is possible to make some observations:

The J results are significantly poorer than the same timing tests on the APL products. J timings seem to be more variable than the APL product timings. Boolean scan has been omitted from the above as it proved dramatically worse with J.

Bear in mind that these tests have had a long history with "conventional APL" and that the code was converted into J by a J novice. My sense is that they tell us that while J offers respectable performance, it does not offer a significant advance *on these benchmarks* over long-established APL products. But, that's not what J is exclusively about.

Session manager

While using J as a developer, most of your time is likely to be spent with the session manager, and a screen shot is shown in Figure 2.

If you compare this with corresponding views of APL★PLUS III or APL/W you will see that J's designers have chosen to go for a clean and simple interface; and indeed this is an observation which is borne out by the sense of interaction between user and J. Quite simply, J feels fast to use, much faster than the benchmarks shown above.

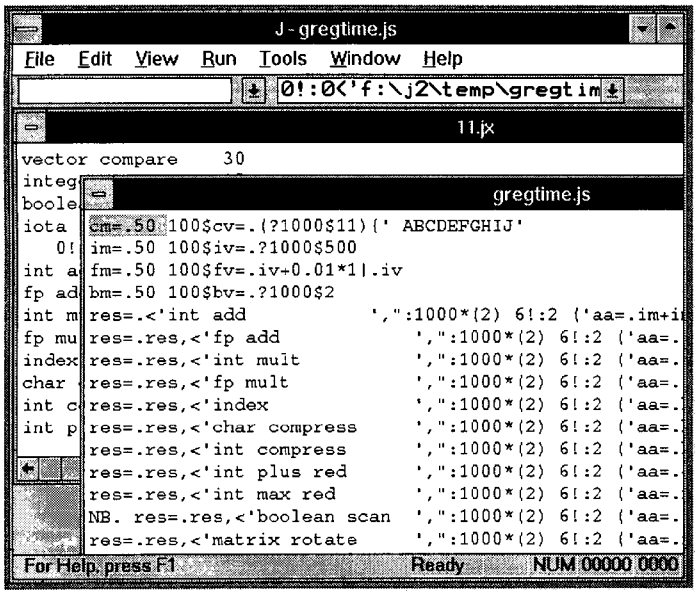

Figure 2: The session manager

The important aspects of the J Session Manager are:

- The user can open multiple execution windows, but they are not independent of one another. Nouns defined in one execution window are usable in any other execution window you have open.

- The user can also open multiple script windows; a typical application would consist of a series of scripts which define the nouns and verbs of the application. This is similar in some ways to the "function file" approach to building APL applications—but a great deal more lively.

- You will also notice the two combo boxes at the top of the shot—the one on the right is an input log and may be used to recall sentences you have type in; the left-hand one is used for searches, but I did not find it especially useful.

There were two aspects of the Session Manager which I did not like:

- There seemed to be no way of interrupting execution; something that you might need to do if your application goes into a loop, for example.

- To re-execute a line it is necessary to locate it and hit Enter— you can then type over the line and modify it. Unfortunately you can also type over the line before hitting Enter. If you do this you end up with two copies of the line—one at the bottom of the session ready for execution, and a modified version in its original location. The result of this is that the execution window does not contain a true record of your session—this is highly misleading. I believe that the J Session Manager should be changed so that it behaves in the same way as (for example) APL2's session log—the user can type over any line, the modified line is executed (command and response at the end of the log) and the original line redisplayed.

J notation

I believe that J's notation is the big divide for us all, and I can't avoid talking about it any longer. The objective of avoiding the problems which come from APL's character set on the hardware of the mid-1980's is a laudable one, although I think a little misplaced in time.

I'm sorry—I can't avoid saying it—I think the cure is worse than the problem. There are just too few characters trying to do too many jobs, and I think the end result is plain ugly.

The unwary APLer is likely to find a few surprises as they come to terms with J, but these are all learnable problems. I would like to see J's advocates publish a lot more material which contrasts how APL and J can solve the same problems. We can all benefit from some of the ideas which are in J but not (yet?) in the APL products.

Once we're away from the core language of J we meet the "foreign conjunction." This reminds me of earlier days, using languages like Coral 66, where topics such as I/O were not part of the language proper but were handled by external libraries (I hear this is still the case with some other languages). The foreign conjunction in J is `x!:y` where the same `x` defines a family of verbs and `y` defines precisely what is to be done—for example `1!:1` reads from a file, `1!:2` writes to the file, and `6!:0` tells you the time. See what I mean about overloading the character set? To be fair, J is quite well packaged to hide this from the programmer and comes provided with a wide range of script files which supply appropriately-named verbs to cover pretty well all of these. The verbs certainly get the novice out of a lot of trouble—for example I was more than happy with `dbr 1`, but leave you to contemplate

```
   dbr
┌──┬──┬─┐
│13│!:│0│
└──┴──┴─┘
 13!:0 1
|rank error
|        13!:0 1
   13!:0 (1)
   (13!:0) 1
```

The third notation of J comes with the "window driver"— which we might run from `11!:0`, but are steered into using the verb `wd` instead.

Window driver

As you know by now, there are two main approaches to offering GUI access in programming languages. The first school is typified by APL2/2 and the original APL★PLUS II Windows product—make the programmer learn the native API. Which is not easy because the vast majority of programmer documentation is in C (or, even worse, C++). But it does offer the industrious programmer total control, and there is no reason why a well-informed APL programmer should not be able to produce applications indistinguishable from those written in any other language.

The other approach is to define a GUI for the language, which is what Dyadic Systems did with APL/W, and what ISI (Iverson Software, Inc.) has done for J. The advantages of this approach are that the APL application programmer can remain productive (all the hard work was done once, by someone who had to know C to write the interpreter), and that the GUI can be portable across platforms (which Dyadic Systems are doing with Motif, and ISI are hoping to do with OS/2). The limitation of this approach is that unless the defined GUI is comprehensive the resulting applications are likely to be feature-challenged.

So, how did ISI do?

Pretty well, I think.

If you use J to develop applications in a Windows environment you get a wide range of intrinsic controls, DDE, OLE, VBX and ODBC. The only thing I failed to get working was the Word macro example described in the User Guide—there was a conflict between what J wanted to use " for and what Word wanted; I could probably have got around this anywhere else but on Crewe railway station at 9:30 on a wet Monday morning—there a some environments which are just plain not conducive to creative thinking.

The main limitation of J's GUI feature set is that (as you might expect) it is more austere than the wilder fantasies coming from downtown Redmond—your users had better be able to read the legends on buttons, because you can't put a picture on them.

Building an application

Reading the manual is all very well, and so is experimenting with the Dictionary—but everything has a purpose and I wanted to make a useful application with J.

Where to begin, and how?

At this point I began to feel a little like the bully-boy spaniel who just met a Rottweiler for the first time. The Dictionary, and most of the other literature that I had read, had me clued up on nouns, verbs, forks, gerunds and trains. The User Guide had patiently led me through a series of Windows Driver examples. But I hadn't got the foggiest idea of how to put them together.

The application I chose was a quite simple one out of electrical engineering—I knew it had to be simple because I had learnt the lesson of trying to do too much the first time with other languages. I also had some Basic source code for it—I would have preferred the raw formulae, but this never was a perfect world.

Stage One was to build a script that did the raw maths; the main struggle here was to make sure that I got the same results from J as I did from Basic (and I never have worked out what the evaluation rules are for Basic). What fooled me for a little while is that the default for J is to run without suspension; the Dictionary confirms that this is so, but does not tell you why you might choose the option—the User Guide has nothing to say on the topic (or if it does, I can't find where). I would guess that the main mistake I made at this point was of putting all of the code into a single script—it would probably have been better to break the lower-level utilities out into a script of their own; I would certainly do this in the future.

One very handy aspect of J2 for masochists determined to recode Basic is that it contains a selection of control words such as "`if.`" and "`while.`".

With the calculation script able to build me a set of useful nouns I could move on to creating a user interface with `wd`.

This went into a separate script and is a perfect example of the old adage that it is a lot easier to make something the second time. Chris Burke's User Guide is, on the whole, a model of clarity—except when I needed to use it in earnest. I knew I ought to use `wdml`, but it took several minutes head-scratching to decode the explanation. Now that I have a working application, it's obvious....

There are (so far as I can tell) no visual tools to help build the interface; the windows must be defined directly with `wd` commands.

For public ridicule, I offer the code shown in Figure 3 as my solution to handling the user interface (this is pretty much a re-elaboration of what is on pages 105–107 of the User Guide). The application is shown in action in Figure 4.

```
closewin=:      wd bind 'psel tfmrwin;pclose;'
tfmrcalc=: 3 : 0
wd 'csel busy; cn "busy";'
top=:".'e0' wdg wd 'qd;'
volts=:".'e1' wdg wd 'qd;'
curr=:".'e2' wdg wd 'qd;'
xyz =: tfmr top,volts,curr
wd 'csel r0;cn ',(":0{xyz),';'
NB.  Repetitive stuff omitted
wd 'csel e0;cfocus;'
wd 'csel busy;cn "";'
)
tfmrwin =: tfmrcalc`closewin`tfmrcalc casetable(((<'*id'),.'ok';'quit'),WDENTER)
tfwin=: 3 :0
wd 'pc tfmrwin;'
wd 'xywh 5 10 120 14; cc c0 static; cn "Topology";'
wd 'xywh 150 10 20 8;cc e0 edit;'
NB.  More boring repetitive stuff
wd 'xywh 10 170 20 10;cc ok button;'
wd 'xywh 40 170 20 10;cc quit button;'
wd 'xywh 70 170 20 10;cc busy static;cn "";'
wd 'pas 5 5;csel e0;cfocus;pcenter;pcloseok;pshow;'
wdml 1
)
```

Figure 3: Code to handle the user interface

As you see—it is quite simple with the user entering values into the upper three edit boxes and seeing the results after pressing either "Enter" or the "OK" button. Something which I had to add was a "busy" sign because the calculations take a second or two and I could find no way to change to the hourglass cursor with wd.

Making the application run directly from Program manager was the final step, achieved through another script which uses the foreign conjunction to load the necessary utility and application scripts, execute the `tfwin` noun and `(2!:55) 0` to log off at the end. Establish a program item in an appropriate Program Manager group and the application runs—presumably with the professional version the J Session Manager window would itself remain invisible.

Ground not covered

I recognise that I have certainly not explored J's locales in this review—my sense is that they are in many ways analogous to APL/W namespaces in terms of allowing the programmer greater leeway in making nouns and verbs selectively visible. Since last writing about APL/W I have converted an application to use namespaces and been very happy with the result; my sense is that J's locales would be equally helpful.

The other aspect that I am conscious of is that all my definitions have been explicit ones; there is much talk in the J literature of tacit definitions and I would like to see more explanations of why we might choose to use one or the other.

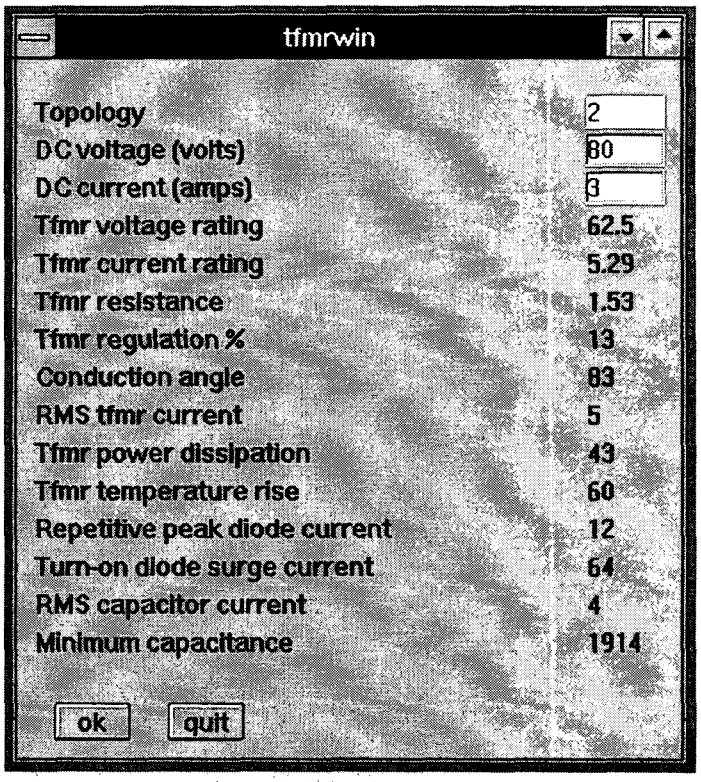

Figure 4: What the application looks like

Summary

I was able to build a small application using J for the first time without a great deal of difficulty; it certainly helped me to have spent some considerable time beforehand as an APL application developer. I think that there is quite a gap between the J Dictionary and what a developer will need to know in order to build significant application—the User Guide and the accompanying script files will help to get people started but there is a need for much more tutorial material.

My guess is that it would take a reasonably proficient APL programmer two to three months to reach an acceptable level of proficiency with J.

As an APL programmer I feel that I have gained from the experience of looking at J—it has had an effect which will continue to ripple through into my APL work.

If I had not been an APL programmer before I doubt that I would have found J a particularly appealing experience; it is very austere.

At this moment I am not sure whether I will use J further; if I did not have access to the other APL products I am sure that I would, but apart from OLE it does not appear to offer any particular advantages.

I would be very happy to see ISI deliver their OS/2 version of J; assuming that they produce a compatible wd it should be possible to take your J GUI applications and run them immediately as native applications. This is a very appealing prospect indeed.

Conclusions

There is one topic on which I have been silent so far, and it is dear to my heart.

Price.

J is an enormous bargain.

The personal edition is priced at 50 US Dollars; the professional edition (which you would need to purchase if you were making products for sale) costs 500 US Dollars. All but the most impecunious can afford to purchase a copy and begin writing Windows applications risking only their learning time. ■

***Dick Bowman** is a frequent reviewer and a regular contributor to Quote Quad. He can be reached at "bowman@apl.demon.co.uk".*

A Reply to the J2 Review

—by Chris Burke and Roger Hui

We would like to thank Dick for his detailed and careful review of the product, and the editors of Quote-Quad for an opportunity to comment.

- The version reviewed was 2.03. The current version is 2.05 and version 2.06 will be released in time for APL95. These releases include several improvements in performance and functionality.

- J for Windows is now distributed in Professional, Personal and FreeWare editions. The US prices are $495 for the Professional edition, $100 for a bundled Personal edition which includes both manuals, or $40 for the disks alone. The freeware edition is available by anonymous FTP from various servers.

- J2.05 has a new forms editor plus improved Windows message handling—overall, it is much easier to build a GUI application than with J2.03.

- The review mentions a problem a mouse driver which is attributed to a conflict with Win32s, but J does not in fact use Win32s.

- Dick describes two problems with the Session Manager:

  - he could not interrupt execution. In fact, Ctrl-Break will interrupt execution.

  - he did not like the fact that the execution window does not contain a true record of the session, unlike APL2. In this respect, J is like other windows products, in that the session log can be edited to get rid of typing mistakes, false starts, and the like.

- The Word macro example included with J requires Word 6, and will not work with Word 2.

- The hourglass can not be changed with the J window driver, but it will be displayed automatically in any application that is taking time.

- The timings in the review were obtained using J2.03. There have been significant speed improvements in J 2.04, '05, and '06. The following table compares APL★PLUS III Version 1.2 and J times obtained under Windows on a 80486⁄50, in milli-seconds averaged over 1000 trials.