VECTOR

Namespaces — Special Feature

The Journal of the British APL Association

A Specialist Group of the British Computer Society

ISSN 0955-1433

Vol.11 No.3 January 1995

Contributions

All contributions to VECTOR may be sent to the Journal Editor at the address on the inside back cover. Letters and articles are welcome on any topic of interest to the APL community. These do not need to be limited to APL themes, nor must they be supportive of the language. Articles should be accompanied by as much visual material as possible (b/w or colour prints welcome). Unless otherwise specified, each item will be considered for publication as a personal statement by the author. The Editor accepts no responsibility for the contents of sustaining members' news, or advertising.

Please supply as much material as possible in machine-readable form, ideally as a simple ASCII text file on an IBM PC compatible diskette (any format). APL code can be accepted as camera-ready copy, in workspaces from I-APL, APL*PLUS, IBM APL2/PC or Dyalog APL/W, or in documents from Windows Write (use the Vector TrueType font, available free from Vector Production), and Winword-2.

Except where indicated, items in VECTOR may be freely reprinted with appropriate acknowledgement. Please inform the Editor of your intention to re-use material from VECTOR.

Membership Rates 1994-95

Category	Fee	Vectors	Passes
UK Private	£12	1	1
Overseas Private	£14	1	1
(Supplement for Airmail, not needed for Europe)	£4		
UK Corporate Membership	£100	10	5
Overseas Corporate	£135	10	
Sustaining	£430	50	5
Non-voting Member (Student, OAP, unemployed)	£6	1	1

The membership year runs from 1st May to 30th April. Applications for membership should be made to the Administrator using the form on the inside back page of VECTOR. Passes are required for entry to some association events, and for voting at the Annual General Meeting. Applications for student membership will be accepted on a recommendation from the course supervisor. Overseas membership rates cover VECTOR surface mail, and may be paid in sterling, or by Visa or Mastercard at the prevailing exchange rate.

Corporate membership is offered to organisations where APL is in professional use. Corporate members receive 10 copies of VECTOR, and are offered group attendance at association meetings. A contact person must be identified for all communications.

Sustaining membership is offered to companies trading in APL products; this is seen as a method of promoting the growth of APL interest and activity. As well as receiving public acknowledgement for their sponsorship, sustaining members receive bulk copies of VECTOR, and are offered news listings in each issue.

Advertising

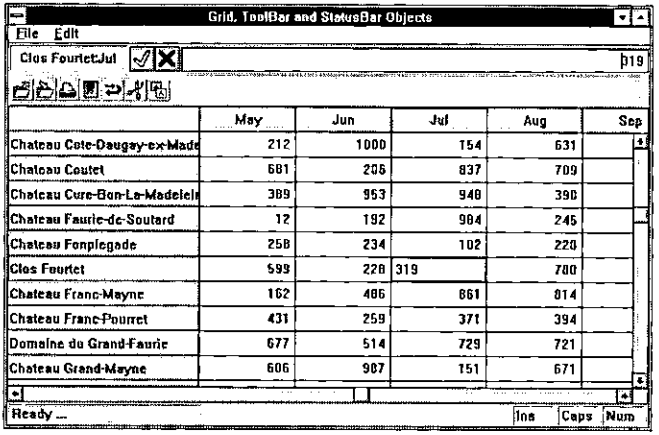
Advertisements in VECTOR should be submitted in typeset camera-ready format (A4 or A5) with a 20mm blank border after reduction. Illustrations should be photographs (b/w or colour prints) or line drawings. Rates are £250 per full page, £125 for half-page or less (there is a £75 surcharge per advertisement if spot colour is required).

Deadlines for bookings and copy are given under the Quick Reference Diary. Advertisements should be booked with, and sent to: Gill Smith, Brook House, Gilling East, YORK YO6 4JJ. Tel: 01439-788385 CompuServe: 100331,644

Contents

dyalog APL

The Definitive APL for Windows



Experience counts

Since we launched Dyalog APL/W at APL92, nearly two years ahead of our nearest competitor, our customers have developed hundreds of successful industrial-strength GUI applications. With the benefit of their experience, we have enhanced and refined Dyalog APL/W into the mature, stable and above all useful Windows development tool that it is today.

Not only does Dyalog APL/W provide the most comprehensive set of GUI facilities available for any APL, but you can be confident that your workspaces will run *unchanged* on Unix workstations under OSF/Motif.

With Version 7.0 we have added support for Visual Basic Custom Controls, a powerful built-in Grid object, Numeric, Currency, and Date fields, ToolBar, StatusBar and TabBar objects, automatic context-sensitive Hints and Tips, Metafiles, MDI, 3-D Forms and Controls, a fully customisable Session, an ODBC interface, *Namespaces* for encapsulation, and a host of other improvements.

That's why Dyalog APL/W remains the professional choice. For a free trial copy, contact Dyadic or your local distributor today.

Dyadic Systems Limited., Riverside View, Basing Road, Old Basing, Basingstoke, Hants RG24 7AL, United Kingdom. Tel:(0256) 811125 Fax: (0256) 811130 Email: sales@dyadic.com.

Guest Editorial

by Duncan Pearson

We are standing at an important point in the development of mainstream APL. The interest in namespaces or contexts or whatever we call them, from the commercial users of APL (that is, everyone who pays good money for a professional interpreter) is generating some action from the developers. Dyadic have introduced namespaces in Dyalog v7 and a great deal of interest has been shown in them. Last May James Wheeler promised that a future release of +III would have similar features, not only providing encapsulation of APL code but also integrating the GUI elements fully with the APL programming structure. Finally J, the newest professional development environment, has locales.

Whatever Manugistics produce I hope that it is different from the Dyalog implementation. This is not because I think that the Dyalog way is the wrong way. I have not used it sufficiently to judge. My point is that until a reasonable body of serious developers have spent time building big applications using these features we cannot tell whether the design is right or not. It is clear from the experience of Kimmo Kekäläinen that there is a world of difference between having a pretty demo that shows some code sitting in the button that calls it, and the reality of using namespaces to organise large chunks of utility code across multiple applications.

So let us have as many different, independently conceived, approaches to this problem as there are interpreters. Furthermore, why should we rely on the writers of interpreters to do the design work? Many people will have tried to solve this problem in their own way using local definition and assignment (the +II user command processor being an example). If you think that your approach has merits then write and explain them.

In the long term this is the feature that will decide what we are using in ten years time, and it had better be right. Let us, the users of APL, discuss freely the relative merits of whatever approaches come along, share our experience of using them and let the developers know when they get it right.

Warning:
Change to *ALL* UK Phone Numbers

For the benefit of overseas subscribers, please note that as from **16th April 1995** there will be a complete revision of the telephone dialling codes in Great Britain.

In most cases, simply add an extra '1' to the code, for example:

+44-439-788385
becomes
+44-1439-788385

The exceptions are:
Bristol: +44-272-730036 >> +44-117-9730036
Leeds: +44-532-xxx >> +44-113-2xxx
Leicester: +44-533-xxx >> +44-116-2xxx
Nottingham: +44-602-xxx >> +44-115-9xxx
Sheffield: +44-742-xxx >> +44-114-2xxx

You should start using the new numbers now.

Quick Reference Diary 1994-95

Date	Venue	Event
30 January 95	London (TBA)	GUI Workshop 'hands on'
25 March 95	Birmingham (TBA)	Beginners' tutorial
19 May 95	IEE	AGM + invited speakers
June 4th-8th 95	San Antonio, Texas	APL95
15 September 95	IEE	Vendor forum

British APL Association meetings are normally held in the IEE, Savoy Place. Nearest tube outlets: Temple or Embankment.

APL Training Courses for 1995

Training courses are offered by MicroAPL Ltd and Bloomsbury Software (formerly Cocking & Drury) - please contact the companies for details.

If you would like to have your courses or seminars listed in Vector, please contact Gill Smith with the details.

Dates for Future Issues of VECTOR

	Vol.11 No.4	Vol.12 No.1	Vol.12 No.2
Copy date	3rd March 95	2nd June 95	1st September 95
Ad booking	10th March 95	9th June 95	8th September 95
Ad Copy	17th March 95	16th June 95	15th September 95
Distribution	April 95	July 95	October 95

News from Sustaining Members

Compiled by Gill Smith

Dyadic Systems Ltd

Dyadic is pleased to announce Dyalog APL/W Version 7.1 for Microsoft Windows. This is a maintenance release that will be distributed to customers free of charge. It does however contain a significant number of enhancements.

As further evidence of Dyadic's commitment to provide greater compatibility with IBM APL2, Dyalog APL/W Version 7.1 includes the following language enhancements: Enclose with Axes, Take and Drop with Axes, Ravel with Axes, and Strand Assignment with parentheses. These enhancements do not conflict with existing Dyalog APL language conventions and are implemented at all migration levels (defined by the system variable ⎕*ML*). In addition, Version 7 offers an optional APL2-compatible partitioned enclose at migration level 3.

The Dyalog APL/W session now supports *drag and drop* editing which is implemented in a manner that is consistent with Microsoft Word for Windows. Drag and drop editing provides a fast and convenient method for moving and copying text (both whole lines and partial lines) within an edit window or between edit and session windows.

Namespaces have been extended in several ways. You may now create a GUI object, such as a Form, as a child of a namespace. Indeed, you can insert a namespace at any level in the GUI hierarchy. For example, you could create a namespace as a child of a Form and then create Buttons and other objects as children of the namespace together with any code and data that the objects need to share. Secondly, namespaces and GUI objects may be stored on component files in their ⎕*OR* form. This feature will greatly simplify the management and re-usability of complex objects and provides the basis for the implementation of *class libraries* in the future. Further enhancements to namespaces may also be included in Version 7.1.

The popular Grid object now supports the selection of rows, columns and blocks of cells. A selected block can be cut or copied to the clipboard and pasted back into the Grid. This facility also allows the user to transfer data very quickly between the Grid object and other spreadsheets such as Microsoft Excel. You may also drag a block of cells and drop them elsewhere within the Grid. Individual rows and columns may be resized by the user dragging the row and column title dividers. The user may also have the system resize a row or column

to fit the contents of that row or column, by double-clicking the mouse over a divider. All of these operations generate new events to which you may attach callback functions and you may also generate these actions under program control. A new mechanism is provided to control the input mode and the behaviour of the cursor keys. All these features have been implemented in a manner that is consistent with Microsoft Excel.

In addition to these changes, the Grid now allows you to associate Combo and Button objects with individual cells. Combos provide a very convenient means for the user to input one of a series of options and Buttons (particularly Check boxes) provide a good way of making and displaying yes/no choices. As Button objects may display bitmaps, icons and metafiles, you can also use them to display pictures in individual Grid cells.

On top of these enhancements, Dyalog APL/W Version 7 users will find that a large number of minor enhancements have been added. Dyadic intends to ship Version 7.1 during January.

Insight Systems

Insight Systems is pleased to announce general availability of the *Professional Edition* of SQAPL. In addition to being an interface from APL to SQL, the SQAPL product range allows APL to function at both ends of a Client/Server application. *SQAPL/PE* is available from Insight Systems for Dyalog APL and APL*PLUS II under Windows and Unix, and for IBM APL2 under OS/2 and Unix. It is available from Soliton for Sharp APL under Unix, and from Manugistics for APL*PLUS III under the name *APL Link Pro.*

Compared to the *Entry Level* version of SQAPL, which is now also available from Manugistics under the name *APL Link*, and is bundled with version 7 of Dyalog APL for Windows, the Professional Edition contains the following enhancements:

- Support for *SequeLink* drivers as an alternative to ODBC, giving high performance access from Windows, OS/2 and Unix to most popular SQL databases and a number of non-SQL services such as IBM CICS, AS/400 Transaction Programs, or our own SQAPL Server.

- Automatic detection of performance options supported by good ODBC drivers; most significantly block fetch modes, which can substantially increase performance (in one example with a WatCom ODBC driver, an order of magnitude increase in performance for multi-row fetches).

- Support for a number of additional APL data types, in particular date/time columns as Julian day numbers, Quad-TS vectors, or base-100 encoded integers, in addition to the ISO Standard character format.
- The ability to store any APL array in a CHAR or BINARY column, in a format which can be extracted by any other SQAPL/PE client even though data is stored in binary format. This can be used to implement a Component File system, which can be used by all four major APL systems, under Windows, OS/2 and Unix.
- Support for data sets, so that you can extract a (subset of a) table, make changes to it using APL, and then get SQAPL to generate the required SQL to apply the same changes to the underlying SQL table.
- Output data can be *grouped* according to the requirements of your application. For example, you can extract all columns of the same data type in a single cell of output, to conserve space compared to the heterogenous/mixed result returned by the Entry Level product.

For more information, contact us at the address on the back cover, or ask your APL dealer.

We are close to releasing most of the new server products mentioned in our news item in the October Vector. Make sure to get the next issue of Vector to read all about them!

HMW Trading Systems Ltd

Please note a new Email contact and a change to our phone number:

Tel: 0171-353-8900; Fax (unchanged): 0171-353-3325
Email: 100020.2632@Compuserve.com

Manugistics Inc
(UK Re-seller: The Bloomsbury Software Company Ltd)

Manugistics are now shipping *APL Link*, the fast easy-to-use interface from APL*PLUS III Windows to all of your data. Using Microsoft's ODBC, APL Link lets you access a wide variety of databases on different hardware platforms, you'll be able to write APL programs that use powerful yet simple SQL statements. And because ODBC is an industry standard, you can change databases and your programs will work with little or no modification.

There are two versions of APL Link to choose from: *APL Link* is an inexpensive yet powerful interface from APL*PLUS III to a variety of databases; *APL Link Pro*

is designed for the power user and combines increased performance with advanced functionality including: the ability to execute blocks of SQL statements simultaneously, support for more data types, and the ability to distribute APL Link as part of a run-time application.

Further details from our UK re-sellers:

The Bloomsbury Software Co Ltd.,
formerly Cocking & Drury (Software) Ltd.,
3-6 Alfred Place,
Bloomsbury, London WC1E 7EB.
Tel: 0171 436 9481;
Fax: 0171 436 0524.

Bloomsbury Software report a number of customers approaching them recently with a view to moving their VSAPL off their mainframes onto PC's running APL*PLUS III Windows — no previous version of APL having offered them both the ease of migration and the necessary performance to make this exercise a reality. Bloomsbury Software have some tools they developed to help automate this process.

See our advertisement on page 141 for more details.

THE EDUCATION VECTOR

January 1995

Editor Ian Clark

This Education Vector has been reprinted from VECTOR Vol.11 No.3. VECTOR is the Quarterly Journal of the British APL Association. For more information about the British APL Association, please contact: Anthony Camacho, 11 Auburn Rd, Redland, BRISTOL, BS6 6LS Tel: 0117-9730036.

Contents

Ian Clark
IAC/Human Interfaces,
9, Hill End, Frosterley,
Bishop Auckland,
Co. Durham DL13 2SX.

Tel: 01388-527190
Email: clark.i@applelink.apple.com or 100021.3073@compuserve.com

Editorial

by Ian Clark

One thing about being an ease-of-use consultant, you get to see a wide range of jobs. Turning my hand to a spot of supply teaching at one of the better girls schools, the headmistress confided in me "We do tend to push the weaker girls towards IT".

So there it is. IT is the Domestic Science of the 90s, fit only for cotton-heads that ought to be barefoot and babbit by the time they're sixteen, and would be if their parents weren't so well-to-do. The sort of girl that's lucky to be leaving school with any sort of qualification. So push-em towards IT. As for the more academically-minded girls, well — who can blame them if they consider anything to do with computers to be beneath their dignity?

I read in a recent issue of CUE Newsletter (Computer-Using Educators, Inc., of Alameda, California): "The dilemma in 1990: we had the technology, we could create powerful, well-designed word-processed documents, charts and graphs, you name it. What power to unleash in a classroom! Unfortunately my students and I shared the same secret — all of these skills only counted in the computer classroom." The writer of the article, entitled "A Goal Without a Plan is a Dream", goes on to recount how things have changed. "The lab had moved from the place where students were learning skills that had little relevance to their real or academic lives to a studio where tools were made available and creatively used."

Assuming that Ms Schandler is not talking through her sweatband, then by comparison we in Great Britain in the Year of Grace 1994 are stuck in a 1980s timewarp. I didn't say 1990 because at that time we were ahead of the Californians in the constructive use of computers in the classroom. But the world moves on — and it seems Britain doesn't. Chris Abbott, writing in Educational Computing and Technology (November 1994), recounts his embarrassment at having to tell erstwhile overseas visitors, who had come to this country to see what had been achieved by the network of LEA centres, that most of them have closed. "The 1993 Education Act suggests that private sector centres will develop overnight, like so many mushrooms, where LEA centres close. No such magical events have taken place." He judges that "there are only two kinds of organisation which now have the funding, the resourcing and the legal right to develop new structures: the universities and the IT industry."

Both of course have their own agenda. Industry will argue, as its running-dogs have been doing in the correspondence column of "Computing", that children must be taught on "industry standard" software and hardware. "Who'd employ somebody trained on an Acorn?" seems to clinch the matter as far as they were concerned. For "Acorn" you might substitute "APL" in the present context.

Of course there were indignant replies pointing out that children being "trained" now won't be looking for jobs for another ten years — and what price now the industry standards of ten years ago? (8-bit computers, 5 inch floppies, 64kb of memory, CIS-COBOL seen as the only way to program a serious commercial application on a PC, if you're silly enough to sidestep the mainframe). Chris Abbott again: "The only definition of industry standard which has long-term credibility is something like 'fitness for purpose at lowest possible cost'".

If people really believed that when purchasing for the classroom, then they would not buy fashionable industry standard systems which "trained", but time-proved, time-proof ones which "taught". Out would go expensive packages which are supposed to exemplify, as closely as the budget will allow, what is out there in the Real World. In would come modelling media in which the mechanisms of a word-processor or a financial package (or genetic engineering or an atomic pile) could be modelled, in terms which the pupil (and even the teacher) could grasp. So it boils down to the choice of a good, cheap durable modelling medium.

Some people build models out of matchsticks. Especially prisoners, who have all the time in the world. Presumably they would use a low level programming language to build a software model. Those of us for whom time (and patience) is in short supply need to model with larger components and subassemblies we could in principle build ourselves — or at least take apart and understand. More like Lego than Lucifers.

Who can manufacture these goodies for us? Universities? When I worked in a university it was academic suicide to be caught making things easy for people with IQ<100. And as for industry — well! Who‡s paying? What are they buying?

I'm not being cynical. Both parties play the game by rules which are handed down to them. It's up to our rulers to make rules which are productive and beneficial, supposing they feel sufficiently motivated to do so. Education of the next generation — isn't that sufficiently motivating? Not if your mentality is straight out of "Chitty-Chitty-Bang-Bang".

Jot-Dot-Floor

by Ian Clark

Here's a quote from the June 1994 editorial:

> "...my first innovation, a column of jottings on rock-bottom educational matters. Since it's simply got to have a techie APL title, what better than 'Jot-Dot-Floor'?"

I didn't really want an answer to this. But I got one. A week or so ago the following flame from cyberspace tracked me down, node-by-node, like King Tut's ghost:

> "This has been bugging me for a while, so I thought I'd better fix it now. Your column title 'jot dot floor' is catchy, but wrong. For someone doing education stuff you're misleading the public. Floor is a monadic verb. Min is a dyadic verb. Both inner and outer products take dyadic verbs as their right operands. Hence your column should properly be titled: 'jot dot min' This is not only correct, but looks nicer [3 characters of 3 each].
>
> jot
> dot
> min — Bob (Bernecky)"

Well, what can I say? Bob's right, of course. I could point to the absence of arguments altogether, which makes it niladic, sort of, but that only draws attention to its being syntactically as well as semantically wrong, besides making people wonder if there are any valid arguments in the body of the text, let alone the title.

I could hide behind a symbolic rendering: ∘.⌊ but that's obscurantist and just bemuses the public. The Editor-In-Chief was no comfort. He said I should have consulted the ISO standard which gives the proper English names for all the primitives (now he tells me!).

But I was thinking of a plan to digitise myself enunciating the names of the primitives in my beautiful BBC English (mummy used to leave me alone in the house with the radio on). My son's already done it in broad Werdle. I could find others to read them for me in Brummie, 'Merkin, perhaps even Strine. Then onto French, and other languages.

I thought some more about it. Do the Finns have standardised names for the APL verbs — and can you type them on a single line? Do the Russian names for the

verbs have perfective and imperfective aspects? Bearing in mind who hosted APL'94, do the Belgians — and do they have twice as many standard names as everybody else — one set in Walloon and one in Flemish?

Who else can I think of? What's Spanish for Floor, or Min? Is it the same in South America? Do Californian schools have to teach three names for every APL verb, in English, Spanish and Vietnamese like their public signs? Have the French expelled the last remaining *soupçon* of Franglais from their APL nomenclature? Is APL usage governed by the *Académie Française*? What about APL in Hebrew — do the verbs decline and the nouns conjugate? Has the Islamic world even begun to think of names for the contents of ⎕*AV*? — or were their scholars calligraphing them from right to left in flowing Naskh during the 11th century? Did the Crusaders actually bring APL back from the Holy Land, only to have it branded as heretical by the official dogma?

Do the Chinese use the same written names as the Japanese, but pronounce them differently? Do the Eskimos have 127 different names for Rho? And what, oh what, are they doing to APL on the Pacific Rim? Do the Ozzies care an ××××?

Let's come nearer home. Do the APL primitives have names in Welsh, and why not, man? Would the acceptance of Gaelic names by the whole Irish people help or hinder the peace process? North of the Border, would the SNP demand different names on the PC and the Macintosh? If Cornish is an extinct language, would Cornish-spoken APL bring it to life again, or might the other thing happen?

I began to fantasise about touring the world on an APL scholarship, armed with a tape recorder and a copy of I-APL, discovering how different primitive tribes pronounced the APL primitives and release my findings into the public domain just in time for APL 2000. The talking part's very easy on the Mac (I've already got one that speaks numbers) and Windows says "Me-Too" nowadays — if you install a Sound-Blaster — but you've spotted the snag, of course. It would need a built-in syntax analyser just to determine whether *'?'* is Roll or Deal, or just a plain query inside a message string. It might be one of those recursively-unsolvable problems when taken across the whole ensemble of possible APL interpreters. In Dyalog APL you can define a new function like this:

```
MYFUN←⍬.⍳
```

- so what's the machine to say when it reaches the end of the expression and finds no right argument?

Other mathematico-philosophical movements have foundered on their nomenclature, especially when you supply not just one, but two or more new names for things your audience already has names for, like good old question-mark. I spent half the seventies trying to get people to call their files "relations" and their records "tuples". Not singlets, doublets and triples, mind, but 1-tuples, 2-tuples, 3-tuples, 4...

Needless to say, our band of high fliers ran into a lot of flak, even from academics, who really ought to have known better. Isn't the whole of academic life all about learning to call everything by its right name (Augustine of Hippo, I believe)? The Company cherished us, like the Mikado, as a source of innocent merriment, but I chucked it in and spent the rest of the seventies researching why people found computers so difficult.

Eh, what's that? Did I discover the reason? Well... no, not entirely. But there are things you can do to be helpful, and things which hinder. Introducing a lot of new names and new concepts with no apparent one-to-one mapping between them is not one of the helpful things to do.

Yes, read my lips. What I'm saying is that the strange characters of APL aren't the problem with the language — that's if you accept there is a problem. It's the names for them. Who complains about code-page 437, I ask you? Yet everyone uses it, everyone still using DOS that is, and it's full of the most bizarre glyphs — Wingdings comes nowhere by comparison. And they all have names, every last jot and sigil of them.

So I think I'll stick with the present title for now, until I can think of a better one. Or a Spanish one, perhaps? Or in one of those Tintin-esque East European languages. It would be fun to see the actual names of the primitives decorated with slashes and backslashes, jots and dots, tildes and carets, all liberally laced with each-pepper.

J-ottings 4

by Norman Thomson

J-ottings is about learning J rather than about J itself — that is left to those more expert. J is much more tantalising than APL ever was. Somehow it is much more difficult to get properly started, and yet the rewards of having done so are great. The J literature is in some respects too polished, which can lead to the feeling of running in a race where the leaders keep disappearing out of sight. It thus seemed worth while to record an account of some failures and wrong avenues encountered on the path to writing a simple J verb.

Eugene McDonnell in "At Play with J" (Vector Vol.10 No.3) articulated the fact that in learning new computer languages, there is a need to have as a handhold the confidence of being able to write simple multi-line programs in the style of more primitive languages. He described a nine-liner to compute primes — I propose to do something much simpler, namely emulate in J the Basic program:

```
10 i=1
20 if i=11 then exit
30 print i
40 i=i+1
50 goto 20
```

and to record a catalogue of intermediate failures. Of course `1+i.10` can achieve my objective at a stroke, but that is not the point. The object is to generate the feeling of security that comes from being able to do it in a step-by-step multi-line program, or as it is called in J, a multi-line verb (mlv).

Multi-line verbs come little and late in the *J Introduction and Dictionary*. A first reading leaves the vague feeling that they have something to do with something called suite (`$.`) which counts lines, and is somewhat similar to $\square LC$. (Suite has in fact been removed from the more commercially oriented J Release 2, however I judge that readers of this section of Vector are more likely to continue to be users of the earlier shareware versions.)

In APL a user-defined function is an entity whose roots are well grounded in traditional programming. However, in J a multi-line verb is a table (or possibly pair of tables in the ambivalent case), where a table is a character matrix. By analogy with APL it is as if the Canonical Representation IS the function. The analogy of suite with $\square LC$ is quite strong in that suite is a vector of row numbers referencing the table, and represents the list of statement numbers which will be

executed in sequence provided that this sequence is not interrupted by explicit assignment to suite. Suite is initially set to `i.n` where `n` is the number of rows in the table. When the value of suite becomes an empty vector, this is a signal to exit the verb.

A table is built up from its component rows using link (;). Suppose these rows are the character strings `a,b,c,...` Then define

```
table=.a;b;c
```

followed by

```
mlv=.table : ''
```

if the verb is monadic, or

```
mlv=.'' : table
```

if it is dyadic, or

```
mlv=.table1 : table2
```

if it is ambivalent.

`a`, `b` and `c` are NOT program variables within `mlv`; they are temporary names used to store the program lines as the verb is built up.

In editing simple tables I find it convenient to edit a line, then redefine `table` and `mlv`, since this is made very convenient by the line recall feature of the J interpreter.

Here is my first attempt at reproducing the Basic program above (remember rows are numbered in origin zero) :

```
a=.'$.=.(1+y.=10),i. y.~:10' NB. ~: is not equal
b=.']y=.y.+1' NB. y. is right argument
c=.'y.'

table=.a;b;c
mlv=.t : ''
```

The idea is that, assuming an argument of less than 10, suite will be set to 1 0 in line 0, so that `y.` is incremented and displayed following execution of line 1, then the 0 in suite restores control to the top line. This process is then repeated until eventually `y.=10`, suite becomes 2, 10 is displayed, and execution terminates. Before reading further see if you can spot the flaw.

The reason for it is stated clearly by Eugene, viz. the result of a verb is the result of the sentence executed last. Execution is thus silent in the sense that a verb such as the above does not produce a line-by-line result. Also, since all variables including `y.` and suite are local, it is not possible to work out after the event what happened within the verb. It is possible to write a verb

```
write=.1!:2&2
```

which uses one of the foreign conjunctions to transmit its argument to the screen, and so replacing `]` in the second line with `write` helps, but now the 10 is displayed twice, once by the trace verb `write`, and once by virtue of the "result-is-last-sentence" rule.

Educated by my failure so far here is a second attempt at the verb (The intermediate stages of building up the table are omitted):

```
$.=.(10-y.)#1
y.=.y.,y.+1
```

This time I calculate in the first line the appropriate number of times the second line has to be repeated. At every stage the newly incremented value of `y.` is catenated, until last time round the full vector from start point to 10 is printed. Again try to spot the flaw before reading on.

Consider `f 9`. This indeed has the value 9 10 as anticipated. Now consider `f 8`. First time round `y.` becomes 8 9. Next time round 8 9 is joined to `y.+1` to give 8 9 9 10 and so on.

A successful verb is:

```
$.=.(10-y.)#1
y.=.y.,1+{:y. NB. {: is tail
```

The above example illustrates a simple way do deal with if/then logic. Extension to the case statement follows in an obvious way:

```
a=.'$.=.y.'
b=.'''one'''
c=.'''two'''
d=.'''three'''

t=.a;b;c;d
f=.t : ''

f 2 two
```

J has labels which use) where APL uses :, and so if/then logic can be expressed:

```
a=.'$.=.(y.=0)}lab2,lab1'
b=.'lab1)''zero'''
c=.'lab2)''not zero'''

t=.a;b;c
f=.t : ''
f 0 zero
f 7 not zero
```

This is used in a simple recursive verb to calculate triangular numbers.

```
a=.'$.=.(y.=0)}lab2,lab1'
b=.'lab1)r=.0'
c=.'lab2)r=.y.+f y.-1'

t=.a;b;c
f=.t : ''
f 5 15
```

In writing multi-line verbs it is not necessary to name each row explicitly in the table build-up phase. For example the above verb could be written:

```
a=.'$.=.(y.=0)}lab1,lab2' ; 'lab2)r=.0' ; 'lab1)r=.y. + f y.-1'

f=.a : ''
f 5 15
```

f in either form is of course a travesty of J style, nevertheless I consider it important to be ABLE to do it this way even although one wouldn't! An acceptable J verb definition would use agenda(@.), tie(') and $: which means "self-reference":

```
f=.0:'(+$:@<:)@.* NB. <: is decrement by 1, * is signum
f 5 15
```

This says take the signum of the right argument. If it is zero use the verb 0: to initialize to 0. Otherwise add (+) the value of f used recursively ($:) after decrementing its argument by 1.

The Common Mean and APL

by Joseph De Kerf

The classical definitions of the mean of two non-negative real numbers are the harmonic mean h(x,y), the geometric mean g(x,y) and the arithmetic mean a(x,y):

$$h(x,y) = 2xy/(x+y)$$
$$g(x,y) = \sqrt{xy}$$
$$a(x,y) = (x+y)/2$$

with $\min(x,y) \le h(x,y) \le g(x,y) \le a(x,y) \le \max(x,y)$. For example, let x=1 and y=99. We obtain respectively:

$$h(x,y) = 1.98000000$$
$$g(x,y) = 9.94987437$$
$$a(x,y) = 50.00000000$$

As we see, there may be a serious gap between the geometric mean g(x,y) and the arithmetic mean a(x,y). This gap may be filled by the concept of *common mean* [1] – a not very familiar concept from the literature. For convenience, let x_0 be the smaller of two non-negative real numbers x_0 and y_0. The geometric mean x_1 and arithmetic mean y_1 are:

$$x_1 = \sqrt{x_0 y_0} \quad \text{and} \quad y_1 = (x_0 + y_0)/2$$

If this procedure of forming alternatively geometric and arithmetic means is repeated indefinitely:

$$x_{i+1} = \sqrt{x_i y_i} \quad \text{and} \quad y_{i+1} = (x_i y_i)/2 \quad \text{with} \quad i = 1,2,3\ldots$$

one obtains:

$$x_0 \le x_1 \le x_2 \le \ldots \le x_i \le \ldots \le y_i \le \ldots \le y_2 \le y_1 \le y_0$$

x_i and y_i converging to the same value. We define this as the common mean $c(x_0,y_0)$ of the numbers x_0 and y_0.

For the example x=1 and y=99 for instance, with an accuracy of 10 digits, we get successively:

9.94987437	and	50.00000000
22.30456721	and	29.97493719
25.85687532	and	26.13975220
25.99792902	and	25.99831376
25.99812139	and	25.99812139

such that c(x,y) = 25.99812139.

Finally, we have:
$$g(x,y) = 9.94987437$$
$$c(x,y) = 25.99812139$$
$$a(x,y) = 50.00000000$$

with $9.94987437 < 25.99812139 < 50.00000000$.

In fact the order in which x and y are treated and the order in which the sequences of geometric and arithmetic means are calculated is not relevant and c(x,y) = c(y,x) (commutativity). In addition c(x,x) = x (idempotency). Finally, c(x,y) = 0 if and only if x = 0 or y = 0 (or both).

Programming the algorithm for calculating the common mean can be somewhat complicated in most programming languages. In APL however, it is very simple. A function to do the job is:

```
      ∇ R←X CMEAN Y
[1]     R←X,Y
[2]    LAB:R←(0.5×+/R),(×/R)*0.5
[3]     →(≠/R)/LAB
[4]     R←0.5×+/R
      ∇
```

which for the chosen example gives:

```
    1 CMEAN 99
25.99812139
```

Accuracy is determined by the current, i.e. the default value of comparison tolerance ⎕CT. It may be changed by defining the comparison tolerance as a global or local variable.

Note: a special case is the common mean of the numbers 1 and $1\sqrt{2}$:

```
      1 CMEAN ÷ 2*0.5
0.8472130848
```

which is known in the literature as the "ubiquitous constant U" since it turns up all over the place. Finally, the common mean is very useful in the design of simple and efficient algorithms for calculating the complete elliptic integrals of the first kind K(p) and of the second kind E(p). More details may be found in [1].

Reference

[1] J Spanier and K B Oldham: *An Atlas of Functions.* Hemisphere Publishing Corporation, New York. New York 1987.

Word-Search Squares in I-APL

by Bill McLean and Ian Clark

I needed a Word-Search making program, since there are a lot of good teaching points involved. The program is written using APLomb, which is a Macintosh screen interface construction set based on I-APL, but any port of I-APL should work, although you won't get to see the square being built up and you won't get the fancy buttons to control it. However the working functions don't care whether they're running on APLomb or not, so simply make all the functions dealing with the interface, viz. *APLOMB*, *BUTTON1*, *BUTTON2* and *REFRESH*, into trivial functions that do nothing when called and it should work with any APL (*I'm going to try it with Dyalog – Ed*).

LIST is a 2D char matrix containing the words to be matched. You can input *LIST* by assigning to it the result of *MAT* `10`, say, supposing 10 is the maximum width of word you want. *MAT* will then accept successive words typed-in, stopping when you just press <Enter> without typing anything.

Define *GRID* to be the size you want, e.g. a 10 by 10 array of asterisks (or anything, they'll get turned to asterisks), Enter in turn:

```
BEGIN
TRY
FINALISE
```

TRY will output into the session log what it's doing as it runs. This listing also happens to tell you the solution to the finished square, something you'll need unless you're very clever at solving these things. You can halt it at any stage

(sometimes it doesn't manage to fit in all the words you give it in *LIST* since it's possible to give it an impossible set) and then run *FINALISE*. This fills in all the remaining asterisks in *GRID* with random letters. Hey presto! — there's your finished square for the school newsletter.

And don't forget, Konky Puzzles made a lot of money selling books of things like this.

Listing of Workspace *WDS*

(A Macintosh version of this workspace is available. Send a blank disk and SAE to the Editor, EV. Other versions by arrangement.)

```
ABANDON: YB[;]←0
APLOMB: 100 ⎕MC ''
BUTTON1: BEGIN
BUTTON2: FINALISE
FINALISE: LOSE GRID[;]←GRID SUBST RANDCHAR ⍝sneaky way to update global
G1: GRID[;]←'*'
IF: ⍵/⍺
IFALL: (∧/⍵)/⍺
INDOWN: GRID[SET;COL]←WORD
INHORIZ: GRID[ROW;SET]←WORD
INRIGHT: GRID[ROW;SET]←WORD
LOSE: : 0 : ⍵ ⍝ suppresses output from a direct definition
MAT: (⍵↑V),[1] MAT ⍵ : 0=ρV←⎕ : (0,⍵)ρ''
NEXT: ((N←N+1)⊖LIST)[1;]
RANDCHAR: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'[?(ρGRID)ρ26]
REFRESH: 101 ⎕MC '' ⍝ maintains kompos in mid-execution
SUBST1: : ×ρρZ←(,⍺),[.5],⍵ : (ρ⍺)ρ((,B)⊖Z)[1;] ⍝how DOES it work??
TB: (¯1↓(' '≠⍵)/⍳ρ⍵)↑⍵

    ∇ BEGIN
[1]   ⍝ START THE APLOMB VERSION OF WORD SEARCH
[2]   RUNNING←←0
[3]   CONTINUE←←1
[4]   GRID[;]←'*'
[5]   LORIENT←WORD←'<empty>'
[6]   TEMPGRID←''
[7]   MAX←ROW←COL←U←0
[8]   N←1 ⍝--use first word in LIST
[9]   APLOMB
    ∇

    ∇ BIT;WORD
[1]   WORD←⎕
[2]   →(1 2∈⍳2)/NORMAL,REVERSE
[3]  NORMAL:
[4]   ⎕←WORD
[5]   →0
[6]  REVERSE:
[7]   ⎕←⌽WORD
[8]   →0
    ∇

    ∇ Z←BYHAND
[1]   'INPUT A WORD'
[2]   WORD←⎕
[3]   L←ρWORD
[4]   'WHICH ROW DOES THE WORD START ON?'
[5]   ROW←⎕
```

```
[6]   D←0,⍳(L-1)
[7]   'WHICH COLUMN?'
[8]   COL←⍞
[9]   SET←ROW+D
[10]  'DO YOU WANT (D)IAGONAL, (H)ORIZONTAL OR (V)ERTICAL?'
[11]  SELECT←⍞
[12]  →('DHV'∈SELECT)/DIAGONAL,HORIZONTAL,VERTICAL
[13] DIAGONAL:
[14]  TEMPBOX←GRID[(ROW+D);(COL+D)]
[15]  NOS←(ROW⍴0),(⍳(L-1)),((10-(ROW+(L-1)))⍴0)
[16]  TEMPGRID←NOS⌽GRID
[17]  TEMPGRID[ROW+D;COL]←WORD
[18]  GRID←(NOS×¯1)⌽TEMPGRID
[19]  ''
[20]  GRID
[21]  →0
[22] HORIZONTAL:
[23]  INRIGHT
[24]  ''
[25]  GRID
[26]  →0
[27] VERTICAL:
[28]  INDOWN
[29]  ''
[30]  GRID
[31]  →0
      ∇

      ∇ Z←X SUBST Y
[1]   ⍝ substitutes Y elements into X as identified by 1 in boolean B
[2]   ⍝ X, Y and B must be the same 2D shape.
[3]   ⍝ Here B is assigned internally, but you can remove the B← line
[4]   ⍝ and compose B before calling SUBST.
[5]    B←X='*' ⍝--optional line, see above.
[6]    Z←(,X),[.5],Y ⍝--Form 2×n array, X on top, Y below
[7]    Z←((,B)⍋Z) ⍝--Rotate Z vertically using B
[8]    Z←(⍴X)⍴Z[1;] ⍝--Take row 1 of Z, reshape like X and return it.
      ∇

      ∇ TRY;Z;SIZE;MAX;J;L;LABEL;LAB;ROTATE
[1]   ⍝ fill GRID randomly with WORD chosen from LIST
[2]    LABEL←DOWNRIGHT,HORIZONTAL,DOWNLEFT,VERTICAL
[3]    SIZE←''⍴⍴GRID ⍝--height of GRID, assume=width
[4]   ⍝ N←0 ⍝⍝⍝ N made global to allow restart
[5]   ⍝
[6]   NEXTWORD:
[7]   ⍝--Use existing N, finish at end of list
[8]    →0 IF N>1↑⍴LIST
[9]   ⍝--clear trailing spaces from next WORD
[10]  L←⍴WORD←TB LIST[N;]
[11] ⍝--keep within MAX row/col for WORD to fit GRID
[12]  MAX←1+SIZE-L
[13]  D←¯1+⍳L ⍝--vector of indexes, 0 1 2... for WORD
[14] ⍝--optionally reverse WORD at random
[15]  ⍎'WORD←⌽WORD' IF 2=?2
[16]  REFRESH
[17]  LABEL←LABEL[4?4] ⍝--scramble order of labels
[18]  J←1 ⍝--indexes LABEL[]
[19] ⍝
[20] NEWORIENT: ⍝--try another orientation
[21]  →ABANDON IF 4<J←J+CONTINUE←+1
[22]  LAB←LABEL[J] ⍝--the label to be used
[23]  YB←(MAX,MAX)⍴1 ⍝--flag array used by UNTRIED
[24]  ⍎'YB←(SIZE,MAX)⍴1' IF LAB=HORIZONTAL
[25]  ⍎'YB←(MAX,SIZE)⍴1' IF LAB=VERTICAL
[26] ⍝
```

```
[27] NEXTCELL: ⍝--find next untried cell
[28]  Z←UNTRIED
[29] ⍝--if no cells left try new orientation
[30]  →NEWORIENT IFALL (Z=0)∨~CONTINUE
[31]  ROW←Z[1]
[32]  COL←Z[2]
[33]  TEMPGRID←''
[34]  REFRESH ⍝--update the APLomb kompos
[35]  →LAB ⍝--go to the randomly chosen label
[36] ⍝
[37] DOWNRIGHT:LORIENT←'down/right'
[38]  ROTATE←D
[39]  Z←1 1⌽TEMPGRID←GRID[(ROW+D);(COL+D)]
[40]  →DIAGONALLY
[41] ⍝
[42] DOWNLEFT:LORIENT←'down/left'
[43]  ROTATE←⌽D
[44]  Z←1 1⌽⌽TEMPGRID←GRID[(ROW+D);(COL+D)]
[45] ⍝
[46] DIAGONALLY:
[47] ⍝--select a block and make Z the diagonal
[48]  REFRESH ⍝--to show TEMPGRID
[49]  →NEXTCELL IF 0∈(Z='*')∨(Z=WORD)
[50] ⍝--insert WORD in the diagonal of TEMPGRID
[51]  TEMPGRID←ROTATE⌽TEMPGRID
[52]  TEMPGRID[;1]←WORD
[53]  TEMPGRID←(-ROTATE)⌽TEMPGRID
[54]  REFRESH ⍝--to show new TEMPGRID
[55] ⍝--replace TEMPGRID in GRID
[56]  GRID[(ROW+D);(COL+D)]←TEMPGRID
[57]  →NEXT
[58] ⍝
[59] HORIZONTAL:LORIENT←'horizontally'
[60]  TEMPGRID←Z←GRID[ROW;SET←COL+D]
[61]  REFRESH
[62]  →NEXTCELL IF 0∈(Z='*')∨(Z=WORD)
[63]  INRIGHT
[64]  →NEXT
[65] ⍝
[66] VERTICAL:LORIENT←'vertically'
[67]  TEMPGRID←Z←GRID[SET←ROW+D;COL]
[68]  REFRESH
[69]  →NEXTCELL IF 0∈(Z='*')∨(Z=WORD)
[70]  INDOWN
[71] ⍝
[72] NEXT:WORD,' inserted ',LORIENT,' at ',⍕ROW,COL
[73]  →NEXTWORD,N←N+1
[74] ABANDON:WORD,' abandoned'
[75]  →NEXTWORD,N←N+1
     ∇

     ∇ Z←UNTRIED;B;I;J
[1]  ⍝ chooses a random YB=1, sets it to 0
[2]   U←+/,YB ⍝--the total of 1s in YB (U global for inspection)
[3]   →EX IF U=I←J←0 ⍝ return 0,0 if there are no 1s left in YB
[4]   Z←(⍴YB)⊥(,YB)\(?U)⌽U↑1 ⍝--one of the 1s in YB selected at random
[5]   YB←YB∧~Z ⍝--turn it off in YB
[6]  ⍝--find coords of the 1 in Z
[7]   I←(∨/Z)/⍳1↑⍴YB
[8]   J←(∨⌿Z)/⍳1↑⍴YB
[9]  EX:Z←I,J
     ∇

       ALPH
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

APL Product Guide

compiled by Gill Smith

VECTOR's exclusive APL Product Guide aims to provide readers with useful information about sources of APL hardware, software and services. We welcome any comments readers may have on its usefulness and any suggestions for improvements.

Pressure on space occasionally prevents us from printing the complete guide, however updates will always be listed. We do depend on the alacrity of vendors to keep us informed about their products. Anyone who is not included in the Guide should contact me to get their free entry — see address below.

We reserve the right to edit material supplied for reasons of space or to ensure a fair market coverage. The listings are not restricted to UK companies and international suppliers are welcome to take advantage of these pages.

For convenience to readers, the product list has been divided into the following groups ('poa' indicates 'price on application'):

- Complete APL Systems (Hardware & Software)
- APL Interpreters
- APL-based Packages
- APL Consultancy
- Other Products
- Overseas Associations
- Vendor Addresses

Every effort has been made to avoid errors in these listings but no responsibility can be taken by the working group for mistakes or omissions.

We also welcome information on APL clubs and groups throughout the world.

All contributions and updates to the APL Product Guide should be sent to Gill Smith, at Brook House, Gilling East, York, YO6 4JJ. Tel: 01439-788385, Email: 100331.644@Compuserve.com

COMPLETE APL SYSTEMS

COMPANY	PRODUCT	PRICES(£)	DETAILS
Dyadic	IBM RS/6000 MD320	11,736	APL POWERstation (Greyscale) 27.5 MIPS, 7.4 Mflops RISC Processor 8Mb RAM, 120Mb Disk 19" 1280x1024 Greyscale Graph Display AIX, OSF Motif, Dyalog APL (1-user)
	IBM RS/6000 MD320	13,817	APL POWERstation (Colour) 27.5 MIPS, 7.4 Mflops RISC Processor 8Mb RAM, 120Mb Disk 19" 1280x1024 Colour Graphics Display AIX, OSF Motif, Dyalog APL (1-user)
	IBM RS/6000 MD320	22,656	Advanced APL POWERstation 27.5 MIPS, 7.4 Mflops RISC Processor 16Mb RAM, 320Mb Disk, 150Mb Tape 19" 1280x1024 Colour Graphics Display AIX, OSF Motif, Dyalog APL (1-user)
	IBM RS/6000 MD520	37,114	APL POWERsystem (8-users) 27.5 MIPS, 7.4 Mflops RISC Processor 16Mb RAM, 320Mb Disk, 150Mb Tape CD-ROM Drive, 16 Ports AIX, Dyalog APL (2-8 user licence)
	IBM RS/6000 MD530	72,054	APL POWERsystem (16-users) 34.5 MIPS, 10.9 Mflops RISC Processor 32Mb RAM, 1.34Gb Disk, 2.3Gb Tape CD-ROM Drive, 16 Ports AIX, Dyalog APL (8+ user licence)
	IBM RS/6000 MD540	122,842	APL POWERsystem (32-users) 41 MIPS, 13 Mflops RISC Processor 64Mb RAM, 1.7Gb Disk, 2.3Gb Tape CD-ROM Drive, 32 Ports AIX, Dyalog APL (8+ user licence)
Interprocess Systems	APL2 Dev't Workstation	poa	Mainframe APL2 supported on a PS/2 via a co-processor card with 16Mb of memory running VM/ESA (370 mode). A complete system includes a PS/2, a P/370 co-processor card, and software licenses for VM/ESA, APL2, GDDM and the full line of Interprocess APL2 enhancements.
MicroAPL	IBM RS/6000	12,000+	POWER range of RISC systems running AIX. Dumb terminal or graphical interface.
	Aurora	20,000+	Multi-user APL computer using 68020 CPU. Std. configuration 2Mb RAM, 16 RS232 ports, 68 Mb hard disc, 720K diskette
Optima	IBM Compatible	poa	Complete PC-based station, APL interpreters & all support eq't

APL INTERPRETERS

COMPANY	PRODUCT	PRICES(£)	DETAILS
APL Software	APL*Plus/PC Release 10	450	STSC's APL for IBM PCs & compatibles.Upgrades from earlier releases also available.
	Run-time	poa	Closed version of APL*Plus/PC which prevents user exposure to APL.
	APL*Plus II	1,395	All the features of mainframe APL*Plus for your 386PC!
	Run-time	poa	
	Dyalog APL	1000-10,000	2nd generation APL for Unix systems
	APL2/PC	poa	IBM's APL 2 for the PC.
Atlantis Software	Analytic Platform (*K*)	poa	*K* is an APL-like language
The Bloomsbury Software Company			
(was Cocking/Drury)	APL*PLUS PC Rel 11	250	STSC's full featured APL for IBMs and compatibles - Version 11 gives free runtime.
	APL*PLUS III Windows	949	The new 32-bit native Windows APL*PLUS. Develop in Windows, and distribute APL applications with no runtime charges. Reasonable migration charges from APL*PLUS/PC and APL*PLUS II.

	APL*PLUS II for DOS	750	Now that APL*PLUS III for Windows is available, the facility for creating Windows applications in PLUS II has been removed, and the price reduced.
	APL*PLUS II for UNIX	poa	STSC's 2nd generation APL for all major Sparc and Risc Unix workstations.
	APL*PLUS VMS	poa	2nd generation APL for DEC VAX computers under VMS.
	APL*PLUS Mainframe	poa	Enhances VS APL with many high performance, high productivity features. For VM/CMS and MVS/TSO offers simple upgrade from VS APL.
Dyadic	Dyalog APL for DOS/386	995	Second generation APL for DOS.Runs in 32-bit mode, supports very large workspaces. Unique "window-based" APL Development Environment and Screen Manager. Requires 386/486 based PC or PS/2, at least 2Mb RAM, EGA or VGA, DOS 3.3 or later.
	Dyalog APL/W for Windows	995	As above, plus object-based GUI development tools. Requires Windows 3.0 or later.
	Dyalog APL for Unix	995-12,000	Second generation APL for Unix systems. Available for Altos, Apollo, Bull, Dec, HP, IBM 6150, IBM RS/6000, Masscomp, Pyramid, NCR, Sun and Unisys machines, and for PCs and PC/2s running Xenix or AIX. Oracle interface available for IBM, Sun and Xenix versions.
IAC/Human Interfaces			
	I-APL/Mac	13	Macintosh version of I-APL
I-APL Ltd	I-APL/PC or clones	8 - 11	ISO conforming interpreter. Supplied only with manual (see 'Other Products' for accompanying books).
	I-APL/BBC Master	8	As above
	I-APL/Archimedes	11	As above
	I-APL/Macintosh	13	As above
	Iverson Software Inc		I-APL is the UK agent for all ISI products, including APLIWIN and JWIN for PC and many other machines.
I-APL/ISI	APLIWIN/386	50	Windows APL including manual
I-APL/ISI	JWIN/386	16	Including *Dictionary of J* and *Introduction to J* **Please note the packing charge of £3 per order.**
IBM APL Products	TryAPL2	free	APL2 for educational or demonstration use. Write, fax or Email to APL Products; specify disk size desired.
	APL2 PC (US Version)	$630	Product No. 5799-PGG. PRPQ Number RJ0411. Order from 1-800-IBM-CALL
	APL2 PC (European Version)	£348	Product No. 5604-260. Part number 38F1753. From all IBM dealers, including MicroAPL.
	APL2 for OS/2 Entry Edition	$185	Part No 89G1556.
	APL2 for OS/2 Advanced Edition	$650	Part No 89G1697. Contains all facilities of the Entry Edition plus: DB2 Interface; co-operative processing TCP/IP Interface; tools for writing APs; TIME facility
	APL2 for Sun Solaris	$1500	Product No. 5648-065.
	APL2 for AIX 6000	poa	Product No. 5765-012.
	APL2 Version 2	poa	Product No. 5688-228. Full APL2 system for S/370 and S/390
	APL2 Application Envt Vn2	poa	Product No. 5688-229. Runtime environment for APL2 packages
Insight Systems	APL*PLUS/PC	poa	APL systems marketed and supported ...
	Dyalog APL	poa	from: Dyadic, Manugistics, IBM
	APL2	poa	under: Windows, OS2 and Unix
Iverson Software Inc.	APLI386	$30	Sharp APL Release 20 for PC 386, 486 with graphics, and ability to operate under Windows.
	APLIPC	$30	For PC under DOS
	APLIWIN	$30	For 386/PC under Windows 3.1
	APL Reference Manual	$30	Documentation for all the above.

	J System Kit	$24	J 6.2 diskette with manual "J:Introduction and Dictionary"
	J Source Code	$90	Full C source code plus 100-page book
MicroAPL	APL.68000 Level I	2000	First generation APL with numerous enhancements. Multi-user version (Unix, Mirage, MCS).
	APL.68000 Level II	2500	Second generation APL. Nested arrays, user defined operators, selective specification etc. Multi-user version (Unix, Mirage, MCS)
	APL.68000/X	1500-6000	Second-generation APL. Nested arrays, user defined operators, selective specification, etc. Multi-user AIX version with full OSF/Motif support.
	APL.68000 Level I Mac, ST, Amiga	87	First generation APL. Single user, full windowing interface, software floating point support.
	Mac, Amiga	250	First generation APL. Single user, full windowing interface, hardware floating point.
	APL.68000 Level II ST	170	Second generation APL. Full windowing interface, software floating point support.
	Amiga	250	Second generation APL. Full windowing interface.Hardware and software floating point support.
	Mac	520	Second generation APL. Full windowing interface.Hardware and software floating point support.
	APL*PLUS Rel 10 APL*PLUS II V 4.0	450 1395	
Optima	APL*PLUS/PC APL*PLUS II APL*PLUS II PC Developers Kit Dyalog APL	359 950 poa 999	
RE Time Tracker Oy	APL*PLUS/PC	poa	Complete APL*PLUS and Statgraphics product range and user support for Finland
	APL*PLUS II/DOS		
	APL*PLUS III/WIN		
	APL*PLUS/UNIX		
Soliton Associates	SHARP APL for MVS	poa	for IBM MVS mainframes
	SHARP APL for Unix	poa	for IBM RS/6000 and Sun SPARC
Uniware	APL*PLUS/PC	495	STSC's full feature APL for IBM PC/XT/AT, Compaq, Olivetti.
	Run-Time	call	Closed version of APL*PLUS/PC which prevents user exposure to APL.
	APL*PLUS/UNX	call	STSC's full feature APL for UNIX based computers
	APL*PLUS II	call	STSC's full feature APL for 386 machines.

APL PACKAGES

COMPANY	PRODUCT	PRICES(£)	DETAILS
Adaptable Systems	FLAIR	poa	Finite loader and interactive rescheduler. Customisable full-function scheduling system. (Available outside Australia by special arrangement only.)
APL-385	APL-385 FSM-385 DRAW-385 DB-385 GEN-385	50(PC),125(mf)	Including ... Screen development Screen design Relational W.S. Miscellaneous Utilities
The APL Group	Qualedi	$1500-4000	Electronic Data Interchange (EDI) translation software for the PC, with strict compliance checking.

APL Software Ltd (mainframe)	RDS	poa	Relation Data Base System
	IPLS	poa	Project Management System
	REGGPAK	poa	Regression Analysis Package
(microcomputer)	POWERTOOLS	295	Assembler written replacement function for commonly used CPU-consuming APL functions, includes a Forms Processor.
	REGGPAK	poa	Regression Analysis Package
	RDS	990	Relational Database System
The Bloomsbury Software Company (was Cocking/Drury) (for VSAPL)	Enhancements & Sharefile	poa	Component files, quad-functions & nested arrays for VSAPL under VM/CMS & MVS/TSO
	Compiler	poa	The First APL compiler!
(for APL2)	Sharefile/AP	poa	STSC's shared access component file system for APL2. Comparable to all APL*PLUS file systems: multi-user storage of APL2 arrays with efficient disk usage.
Cinerea AB	ORCHART	250	Organization chart package for IBM APL2/PC. Full & heavily commented source code included - free integration into other applications. NB: ASCII output with line-drawing (semi-graphic) characters for boxes.
CODEWORK	HELM	poa	Decision Support system for top management. Developed in Italy over 7 years. Requires APL mainframe or APL*PLUS/II. Optional modules: EIS, Excel Interface, DTP output via LATEX, output on map background.
CYBEX AB	APL Graf/PC	290	Presentation graphics for APL*PLUS/PC (CGI)
	APL Graf II/PC	390	Presentation graphics for APL*PLUS II/PC (CGI).
	Utility Functions APL2	1900	For APL mainframe; incl. a very fast search.
	Utility Functions II/PC	130	Same package for APL*PLUS II/PC.
H.M.W.	4XTRA	poa	Front-end Foreign Exchange dealing / pos keeping
	Arbitrage	poa	Arbitrage modelling
	Basket	poa	Basket currency modelling
	Menu-Bar	poa	pull-down menu for APL*PLUS/PC
HRH Systems	APL Utilities	poa	Software to transfer workspaces between APL*PLUS and Sharp, and between APL*PLUS and I-APL. Software to import IBM .ATF files to APL*PLUS.
	APL*PLUS Utilities		Public domain software, unlock locked fns, a user-friendly alternative to locking, fns of mathematical physics, menus, and others.
IAC/Human Interfaces	IAC/Graf	15	Graph plotting for I-APL/Mac
	IAC/Vox	15	Spoken APL characters for I-APL/Mac
I-APL Ltd	Educational workspaces	5	PC format disks with the examples from: Thomson, Espinasse (Kits 1-4), Kromberg, Jizba & FinnAPL. All the examples to save your fingers!
Impetus Ltd	*Impetus*	poa	Corporate Modelling and Reporting System.
INFOSTROY	APL*PLUS/Xbase Interface (II/386 Version 2)	$198	Complete package written in C. Comparable with the data, index & memo files of FoxPro, dBASE, & Clipper. Multi-user support. No DBMS license required.
	(PC Version 2)	$98	As above for APL*PLUS/PC.
	(DLL Version 1)	$198	The same in a DLL form! Gives your Windows applications all advantages of DLLs.
Insight Systems	IUTILS/XP	20-95	Cross-platform utility library including simple OS calls (DIR, COPY, DEL, RENAME) and DATE functions. For APL*PLUS II, APL2 and Dyalog APL under Windows, OS/2 and Unix.

	ASI	95	APL Spreadsheet Interface. "Device-Independent" spreadsheet driver supporting Excel, 123 and Quattro-Pro for Dyalog APL/W
	WinCom	95	Asynchronous comms package for Dyalog APL/W
	S2D,22D,X2X	poa	Advanced APL syntax analysis and conversion packages from Sharp and APL2 to Dyalog, and between any two APLs
	SQAPL Client	poa	Interface from APL*PLUS II, APL2 and Dyalog (Windows, OS/2 or Unix) to most SQL databases over most networks.
	SQAPL Server	poa	Makes APL*PLUS II, APL2 or Dyalog APL (Unix) available as SequeLink servers. Can be called from SQAPL clients or other applications such as Excel, C++, Smalltalk, Visual Basic.
Interprocess Systems	APL2 Development Workstation	poa	
	IEDIT	$3000-5000	Full screen APL2 editor with immediate APL execution, and full-screen debugger
(mainframe)	AFM	$6500-15300	High performance component and keyed file system (VS APL and APL2)
(PC)	AFM	$175	Single user component and keyed files for APL2/PC.
	Enhanced Format	$2575	A QuadFMT data formatter for VS APL and APL2
	PowerCode	$2000	External functions for APL2
	CALL/AP	$4700	For calling non-APL programs (VS APL and APL2)
	WSORG	poa	Full-screen Workspace Organizer for APL2.
JAD Software	JAD SMS	150-500	Software management system for APL*PLUS II based on hierarchical databases; includes full-screen interface and stand-alone functions. Price depends on number of users.
Lingo Allegro	FRESCO Business Graphics	$250	Fast and Easy Business Graphics DLL
	GDDME	$1000	AP126 GDDM graphics emulation for Dyalog/W
	AP127	$250	ODBC interface for Dyalog/W
	FACS	$1000	EMMA-like functions for SQL tables
	TOPR	poa	APL Code and Application Management for Dyalog APL/W
	Rumba Connection	$250	Connect Mainframe APL to Dyalog APL/W using Rumba
	IRMA Connection	$250	Connect Mainframe to Dyalog APL/W using Irma for Windows
Mercia	LOGOL 92	poa	Logistics management system for 386/486 & RISC computers. Sales Forecasting, Inventory Management, Master Scheduling, Distribution Requirements Planning, Sales & Operations Planning.
	TWIGS	poa	A modular library of tools to teach and explore state-of-the-art materials management concepts. Developed by R.G. Brown.
MicroAPL	MicroTASK	250	Product development aids
	MicroFILE	250	File utilities and database
	MicroPLOT	250	Graphics for HP plotters etc
	MicroLINK	250	General device communications
	MicroFORM	250	Full screen forms design
	MicroSPAN	250	Comprehensive APL tutor
	MicroPLOT/PC	250	For APL*PLUS/PC product
	MicroSPAN/PC	250	APL self instruction for APL*PLUS/PC
	STATGRAPHICS Rel 5	590	
RE Time Tracker Oy	UIT/W	poa	TMT-Team Oy's User Interface Toolkit for APL*PLUS II and PLUS III under Windows. Comprehensive spreadsheets, replicated fields, special field types, etc.
	DB+	poa	TMT-Team Oy's database interface for APL*PLUS II & PLUS III under Windows. Interfaces to almost twenty different databases.
Soliton Associates	LOGOS	poa	Application Development Environment

	MAILBOX	poa	Electronic Mail
	VIEWPOINT	poa	Report generator with interfaces to DB2 and MVS data
UNIWARE (for mainframe)	STSC's ENHANCEMENTS	poa	Quad-functions & nested arrays for IBM VSAPL
	STSC's SHAREFILE	poa	component files for IBM VSAPL and for IBM APL2
	TOOLS & UTILITIES	poa	Including FILEPRINT, FILESORT, FILECONVERT FILEMANAGER(EMMA) STSC's database package
	EXECUCALC	poa	Mainframe spreadsheet compatible with VISICALC and part of LOTUS 1-2-3 under VSAPL(VM or TSO)
(for APL*PLUS/PC)	APL Debugger 2.1	FF1950 FF9750	A visual APL debugger to help develop applications (site license)
	Menus 3.0	FF2450 FF12250	Complete set of hierarchical menu utilities (site license)
	ETATGEN 2.0	FF1950 FF9750	Page layout report generator (site license)
	UNITAB 2.0	FF4550 FF22750	An APL*PLUS spreadsheet-like data entry and validation system (site license)
	UNIASM 3.0 (site license)	FF4950	Assembler utilities to speed up APL*PLUS/PC applications
	UNISTAT 5.1	FF2900	Data analysis add-on module for Statgraphics
(for APL*PLUS II)	UNIWARE Toolkit II 4.1	FF39000	(site license only). Relational database system and complete set of utilities for APL*PLUS II development
	APL Debugger II 2.1	FF2950 FF14750	A visual APL debugger to help develop applications (site license)
	Menus II 4.0	FF3950 FF19750	Complete set of hierarchical mouse-driven menu utilities (site license)
	ETATGEN II 2.0	FF2950 FF14750	Page layout report generator (site license)
	UNITAB II 2.0	FF6950 FF34750	An APL*PLUS spreadsheet-like data entry and validation system (site license)
	UNISTAT Plus 5.2	FF4300	Data analysis add-on module for Statgraphics
Warwick University	BATS	250	Menu driven system for time series analysis and forecasting using Bayesian Dynamic modelling. Price is reduced to £35 for academic institutions.
	FAB	free	Training program for the above.
Zark	APL Tutor (PC)	$299	APL computer-based training. Available for APL*PLUS PC & APL*PLUS II. Demo disk $10.
	APL Tutor (MF)	$5000	Mainframe version.
	Zark ACE	$99	APL continuing education, APL tutor news and hotline phone support.
	APL Advanced Techniques....	$59.95	468pp. book, (ISBN 0-9619067-07) including 2-disk set of utility functions (APL*PLUS PC format).
	Communications	$200 pc, $500 mf	Move workspaces or files between APL environments.

APL CONSULTANCY

COMPANY	PRODUCT	PRICES(£)	DETAILS
Active Workspace	APL Programming	poa	Short or long-term consultant available.
Adfee	Consultancy	poa	Development, maintenance, conversion, migration, documentation, of APL products in all APL environments
Andrews	Consultancy	poa	APL programming and analysis, specialises in tree-processing algorithms.

APL People	Consultancy	poa	Consultants available at all levels. Expertise in APL system design, project management, prototyping, financial applications, decision support systems, MIS, links to non-APL systems, documentation, etc.
Bloomsbury Software	Consultancy	300-750+VAT	
Camacho	Consultancy	poa	Manuals; feasibility reports and estimates; analysis and programming; APL and MS Windows applications; Sharp, ISI APL, APL*PLUS, APL2/PC and other APLs spoken. Fixed price systems a speciality
Rav Cannon	Consultancy	poa	APL, C, Assembler, Windows, Graphics: PC and mainframe
Paul Chapman	Consultancy	poa	24-hr programmer: APL, C, Assembler, Graphics; PC, mini, mainframe and network.
David Crossley	Consultancy	poa	Broad experience in many APL environments
Peter Cyriax	Consultancy	100-150 120-200 160-300	Junior Consultant Consultant Senior Consultant
Dogon Research	Consultancy	poa	APL Systems consultancy, design, implementation, support, documentation and maintenance. All dialects with special emphasis on APL2 and Dyalog APL/W.
Dyadic	Consultancy	poa	APL and Unix system design, consultancy, programming and training.
E & S	Consultancy	poa	System prototyping: all types of information system, engineering software, graphics and decision support systems APL*PLUS/PC, APL2, Dyalog APL
Evestic AB	Consultancy	poa	Excellent track record from 10+ years of APL applications in banking, insurance, and education services. All dialects, platforms and project phases. SQL expertise.
General Software	Consultancy	from 120	
Greymantle Assoc Ltd	Consulting	poa	Company reporting, business graphics, Windows applications with Dyalog APL/W.
H.M.W.	Consultancy	poa	System design consultancy, programming. HMW specialize in banking and prototyping work.
Michael Hughes	Consultancy	poa	Consultant with 10+ years experience with various APL interpreters and C.
IAC/Human Interfaces	Consultancy	350	APL on Macintosh & PC. HCI design. VDU ergonomics: EC/Health & Safety compliance.
	Documentation	100-200	On-line assistance, product demos & mock-ups, manual writing; foreign language software localization.
	Training	poa	Using I-APL for courseware & distance learning materials; Mac programming in C, APL & HyperCard.
INFOSTROY	Consultancy	poa	APL*PLUS & Windows consultancy. Porting of software written in C into APL*PLUS.
Insight Systems	Consultancy	poa	Experts in APL conversions between any combination of: APL*PLUS, APL2, Dyalog APL and Sharp APL. We are also experienced right-sizers, comfortable with networks and relational databases (that also means when NOT to use SQL) and client/server development in APL, C and Visual Basic.
Intelligent Programs	Consultancy	poa	Systems development, enhancements, support.
	Documentation	poa	Preparation of new manuals, rewriting of existing materials.
	Training	poa	Training for APL experts through to non-technical system users.
JAD Software	Consultancy	poa	Systems design and development, project management, technical manuals, financial and actuarial expertise in APL.
Kestrel	Consultancy	poa	All APLs, all environments. Design, analysis, coding, maintenance, documentation, training, interfacing.
Lingo Allegro USA	Consultancy	poa	General APL consultancy specializing in Prototyping, Migration, Mainframe to PC Downsizing, Performance Analysis, Troubleshooting, and Graphics.

MicroAPL	Consultancy	poa	Technical & applications consultancy.
Ellis Morgan	Consultancy	250-500	Business Forecasting & APL Systems.
Optima	Consultancy	poa	A range of consultants with 3-15 yrs APL PC and mf experience.
Parallax Systems Inc	Consultancy	$750	Introductory APL, APL for End-user & Advanced Topics in APL
QB On-Line	Consultancy	350	Specialising in Banking, Financial & Planning Systems.
RE Time Tracker Oy	Consultancy	poa	Specialised in comprehensive APL Windows user interfaces, APL Multimedia, APL to API level interfacing for Windows, Windows applications, DLLs & databases. Also franchising consultancy
Rex Swain	Consultancy	poa	Independent consultant, 20 years experience. Custom software development & training, PC and/or mainframe.
Rochester Group	Consultancy	poa	Specialise in MIS using Sharp APL
Sykes Systems Inc	Consultancy	poa	Complete APL services specialising in audit, optimisation and conversion of APL systems. Excellent design skills. All dialects and platforms. 17-23 years experience.
Unlware	Consultancy (Senior)	FF/day 5000	Consultancy from people with at least 8 years APL experience.
	Consultancy (Senior)	FF/day 7500	Advice and training in Windows programming with APL*PLUS II
	Training	FF10000	5-day class on Windows programming with PLUS II version 4.0
Wickliffe Computer	Consultancy	poa	System design, consultancy, programming and documentation. Especially project management and decision support systems

OTHER PRODUCTS

COMPANY	PRODUCT	PRICES(£)	DETAILS
Adfee	Employment	poa	Contractors and permanent employees
APL People	Employment Agency	poa	Employees placed at all levels.
Bloomsbury Software	Training	poa	Contact the company for details.
HMW	Employment	poa	Contractors and permanent employees placed.
HRH Systems	APL lessons		On-screen interactive APL lessons for APL*PLUS, TryAPL2, Sharp and I-APL — in English or French.
	The BBS\APL:	$24 p.a.	703-528-7617, 1200-14400b, N-8-1, 24 hours. APL educational material is downloadable free. An additional 30 megs of APL software for APL*PLUS, PLUS II, IBM, Sharp & I-APL is available to subscribers (cost is $24/yr). Selection available on disk for $15 post-paid. Free on-disk catalogue.
I-APL Ltd	An APL Tutorial	3	45pp by Alvord & Thomson
	An Encyclopaedia of APL (2d Ed)	6	228pp by Helzer
	APL in Social Studies	3	36pp by Traberman
	I-APL Instruction Manual (2d Ed)	3	55pp by Camacho & Ziemann
	APL Programs for the Mathematics Classroom (Springer-Verlag)	16	185pp by Thomson
	J Dictionary	16	by Ken Iverson
	Programming in J	10	75pp by Ken Iverson
	Arithmetic	12	118pp by Ken Iverson
	An Introduction to J	8	47pp by Ken Iverson
	Tangible math	8	36pp by Ken Iverson
	Sharp APL Reference Manual	18	349pp by Berry
	APL Press Books	poa	A comprehensive selection of early APL literature
	Please note there is a packing charge of £3 per order		
Iverson Software Inc.	Programming in J	$15	76pp.
	Tangible Math	$12	34pp.
	Arithmetic	$18	123pp.
Kestrel	Employment	poa	Permanent and contract, home and abroad. From individual placement to supply of complete project teams.

	Software Library	poa	Low-cost software distribution service; call for details.
Renaissance Data Systems	Booksellers		The widest range of APL books available anywhere. See Vector advertisements.
Soliton Associates	MVSLINK	poa	Interface from Sharp APL (Unix & MVS) to non-APL data and software in the MVS environment.
	SSQL	poa	High-performance DB2 interface for Sharp APL (Unix and MVS).

OVERSEAS ASSOCIATIONS

GROUP	LOCATION	JOURNAL	OTHER SERVICES	Ann.Sub.
ACM/SIGAPL	International	Quote Quad		
APL Bay Area	USA N. California	APLBUG	Monthly Meetings (2nd Monday)	$15
APL Club Austria	Austria	-	Quarterly Meetings	200AS(indiv), 1000AS(corp)
APL Club Germany	Germany	APL Journal	Semi-annual meetings	DM60
APL Interest Group	South Africa	-	-	
Ass. Francophone pour la promotion d'APL	France	Les Nouvelles d'APL		
BACUS	Belgium	APL-CAM	Conferences & Seminars	£18 ($30)
Chicago SIG	Chicago			
Hartford Group	Hartford, Connecticut, USA			
CPCUG APL SIG (Capital PCUG)	Washington, D.C.	Monitor	Monthly meetings, occasional classes	free
Danish SIG	Denmark			
Dutch APL Assoc.	Holland	-	Mini-congress, APL ShareWare Initiative	
FinnAPL	Helsinki, Finland	FinnAPL Newsletter	Seminars on APL	100FIM(private), 30(student), 1000 (Co)
Japan APL Assoc.	Tokyo, Japan			
Melbourne APLUG	Melbourne, Australia		Quarterly meetings	free
New York SIG	New York, USA			
Potomac SIG	Washington DC, USA		Free monthly meetings	
Rochester APL	Rochester, New York			
Rome/Italy SIG	Roma, taly			
SE APL Users Grp	Atlanta, Georgia	SEAPL Newsletter		
SOCAL	Southern California		Seminars	$15 ($5 students)
SovAPL	Obninsk, Russia			
SwedAPL	Sweden	SwedAPL Nytt	Semi-annual meetings, seminars	SEK 75
SWAPL	Texas, USA	SWAPL		$18
Swiss APL (SAUG)	Bern	Part of Qtly SI-Info		SF60 (SI) + SF20 (SAUG)
Sydney APLUG	Sydney, Australia	Epsilon	Monthly Meetings	
Toronto SIG	Toronto, Canada	Gimme Arrays!	Monthly Meetings, APL skills database, J SIG, Toronto Toolkit	$25

VENDOR ADDRESSES

COMPANY	CONTACT	ADDRESS & TELEPHONE No.
ACM/SIGAPL	Donna Baglio	ACM, 1515 Broadway, New York, NY 10036 USA Tel:+1 (212) 626-0606 Email: baglio@acm.org
Active Workspace Ltd	Ross D Ranson	Moulsham Mill Centre, Parkway, Chelmsford, Essex, CM2 7PX. Tel: 01245-496647; Fax: 01245-496646.
Adaptable Systems	Lois & Richard Hill	49 First Street, Black Rock 3193, Australia. Tel: +61 3 589 5578 Fax: +61 3 589 3220
Adfee	Bernard Smoor	Dorpsstraat 50, 4128 BZ Lexmond, Netherlands. Tel +31-3474-2337, Fax: +31-3474-2342
Andrews	Dr Anne D Wilson	23, The Green, Acomb, YORK YO2 5LL Tel: 01904-792670
APL-385	Adrian Smith	Brook House, Gilling East, York. Tel: 01439-788385 Fax: 01439-788194 Email: 100331.644@compuserve.com
APL Bay Area Users Group APLBUG	Lewis H. Robinson (Sec)	1100 Gough St, Apt 14A, San Francisco, CA 94109, USA Tel: +1 (415) 928-2058 Email: frgp21a@prodigy.com
APL Club Austria	Erich Gall	IBM Österreich, Obere Donaustrasse 95, A-1020 Wien, Austria
APL Club Germany	Dieter Lattermann	Rheinstrasse 23, D-69190 Walldorf, Germany. Tel: +49 6227-63469 Compuserve: 100332,1461
The APL Group Inc	Stuart Sawabini	644 Danbury Road, WILTON, CT 06897 USA. Tel: +1 (203) 762-3933 Fax: +1 (203) 762-2108
APL Interest Group, South Africa	Mike Montgomery	Private Bag X11, Rivonia 2128, South Africa Tel: +27 (11) 803-7200 Fax: +27 (11) 803-9134 Email: mikemont@spl.co.za
APL People / Software	Jill Moss	The Old Malthouse, Clarence St, BATH, BA1 5NS. Tel: 01225-462602
Association Francophone pour la promotion d'APL	Dr. Gérard Langlet	SCM, C.E. Saclay, F-91191-Gif sur Yvette, France. Fax:+33 1 69-08-79-63
Atlantis Software	Arthur Whitney	1105 Harker Avenue, Palo Alto, CA 94301 USA
BAGUS	Joseph de Kerf	Roolnberg 72, B-2570 Duffel, Belgium. Tel: +32 15 31 47 24
The Bloomsbury Software Co Ltd	Peter Day	3-6 Alfred Place, Bloomsbury, London WC1E 7EB. Tel: 0171-436 9481; Fax: 0171-436 0524
Camacho	Anthony Camacho	11 Auburn Road, Redland, Bristol BS6 6LS. Tel: 0117-9730036. email: acamacho@dix.compulink.co.uk Reutemet (Sharp): ACAM
Ray Cannon		21 Woodbridge Rd, Blackwater, Camberley, Surrey GU17 0BS Tel: 01252-874697
Paul Chapman		51B Lambs Conduit Street, London WC1N 3NB. Tel: 0171-404 5401. Compuserve: 100343,3210
Chicago SIG	Larry Mysz	836 Highland Drive, Chicago Heights, IL 60411 Compuserve:73040,3032
Cinerea AB	Rolf Kornemark	Skyttegatan 25, S-193 00 Sigtuna, Sweden.
CODEWORK	Mauro Guazzo	Corso Cairoli 32, 10123 Torino, Italy. Tel: +39 11 885188 Fax: +39 11 812 2652
CPCUG	Lynne Startz	Capital PC User Group, 51 Monroe Street, Suite PE-2, Rockville, Maryland 20850, USA. Tel: +1 (301) 762-9372.
David Crossley		187 Le Tour du Pont, Quartier Le Mourre, 84210 ST DIDIER, France Tel: +33 90-66-08-87
CYBEX AB	Lars Wentzel	Gruvgatan 35B, S-421 30 V. Frölunda, Sweden. Tel: +46 31-45 37 40. Fax: +46 31-45 24 23.
Peter Cyrlax Systems	Peter Cyrlax	22 Hereford Road, London W2 4AA. Tel: 0171-229 5344
Danish User Group	Per Gjerlof	Email: gjerper@inet.uni-c.dk
Datatrade Ltd.	Ian Tomlin	1 & 2 Sterling Business Park, Salthouse Road, Brackmills, Northampton, NN4 0EX. Tel: 01604-760241
Dogon Research	Dick Bowman	2 Dean Gardens, London E17 3QP Tel: 0181-520 6334 Email:bowman@apl.demon.co.uk
Dutch APL Association	Bernard Smoor (Sec)	Postbus 1341, 3430BH Nieuwegein. Tel: +31 3474-2337
Dyadic Systems Ltd.	Peter Donnelly	Riverside View, Basing Road, Old Basing, Basingstoke, Hants RG24 0AL. Tel: 01256-811125 Fax: 01256-811130

E & S Associates	Frank Evans	19 Homesdale Road, Orpington, Kent BR5 1JS. Tel: 01689-824741
Evestic AB	Olle Evero	Bertellusvagen 12A, S-146 38 Tullinge, Sweden Tel&Fax: +46 778 4410
FinnAPL		Suomen APL-Yhdistys RY, FinnAPL RF, PL 1005, 00101 Helsinki 10, Finland
General Software Ltd	M.E. Martin	22 Russell Road, Northholt, Middx, UB5 4QS. Tel: 0181-864 9537
Greymantle Associates	George MacLeod	Bartrum House, Ravens Lane, Berkhamsted, Herts, HP24 2DY Tel: 01442-878065 Email: 100412,1305@compuserve.com
Hartford CT Group	Bob Pomeroy	Mass Mutual Life, 1295 State St, Maildrop F465, Springfield, MA 01111 Tel: +1 (413) 788-8411x2838
H.M.W.Trading Systems Ltd	Stan Wilkinson	Hamilton House, 1 Temple Avenue, Victoria Embankment, London EC4Y 0HA. Tel: 0171-353 8900; Fax: 0171-353 3325; Email:100020.2632@ compuserve.com
HRH Systems	Dick Holt	3802 N Richmond St, Suite 271, Arlington, VA 22207 Tel: +1 (703) 528-7624; Email: dick.holt@acm.org
Michael Hughes		28 Rushton Road, Wilbarston, Market Harborough, Leics., LE16 8QL. Tel: 01536-770998
IAC/Human Interfaces	Ian A. Clark	9 Hill End, Frosterley, Bishop Auckland, Co. Durham DL13 2SX Tel: 01388-527190. Email: clark.i@applelink.apple.com CS: 100021,3073
I-APL Ltd	Anthony Camacho (for queries, order forms)	11 Auburn Road, Redland, Bristol BS6 6LS. Tel: 0117-9760036 email: acamacho@clx.compulink.co.uk Reutemet (Sharp): ACAM
	J C Business Services (for pre-paid orders only)	56 The Crescent, Milton, Weston-super-Mare, Avon, BS22 8DU
IBM APL Products	Nancy Wheeler	APL Products, IBM Santa Teresa, Dept M46/D12, 555 Bailey Avenue, San Jose CA 95141, USA. Tel: +1 (408) 463-APL2 (=2752) Fax: +1 (408) 463-4488 Email: APL2@vnet.ibm.com Cserve: GO IBMAPL2
Impetus Ltd	Cedric Heddle	Rusper, Sandy Lane, Ivy Hatch, SEVENOAKS, Kent TN15 0PD Tel: 01732-885126
INFOSTROY	Alexei Miroshnikov	3 S. Tulenin Lane, St. Petersburg 191186 Russia. Tel:+7 812-3111611 Fax:+7 812-3153321 Email:alm@infostroy.spb.su
Insight Systems ApS	Morten Kromberg	Nordre Strandvej 119A, DK-3150 Hellebæk, Denmark Tel:+45 42 10 70 22 Fax: +45 42 10 75 74 Email: Insight@inet.uni-c.dk
Intelligent Programs Ltd	Mike Bucknall	9 Gun Wharf, 130 Wapping High St, London E1 9NH Tel: 0171-265 1120
Interprocess Systems Inc.	Stella Chamberlain	11660 Alpharetta Highway, Suite 455, Roswell, Georgia 30076, USA Tel: +1 (404) 410-1700. Fax: +1 (404) 410-1773 Cserve: 70373,2676
Iverson Software Inc.	Eric Iverson	33 Major Street, Toronto, Ontario, Canada M5S 2K9 Tel: +1 (416) 925-6096; Fax: +1 (416) 488-7559
JAD Software	David Crossley	580 Eyer Drive, #81 Pickering, Ontario, Canada L1W 3B7 Tel: +1 (905) 837-1895 Fax: +1 (905) 831-5172
Japan APL Association		23-2-302 Hiromichi, Adachi-ku, Tokyo 120, Japan
Kestrel Consulting	Mark Harris	Business & Technology Centre, Bessemer Drive, Stevenage, Herts. SG1 2DX Tel: 01438-310155 Fax: 01438-310131
Lingo Allegro USA Inc.	Walter G. Fil	113 McHenry Road, Suite 161, Buffalo Grove, IL 60089 USA Tel:+1 (312) 203-4928 Fax:+1 (708) 459-8501 Cserve: 71303,3224
Melbourne APL Group	Harvey Davies	CSIRO Div Atm Res, Private Bag No.1, Mordialloc, Victoria 3195, Australia Tel: +61 3 586 7574 Fax: +61 3 586 7800 Email: hld@dar.csiro.au
Mercia Software Ltd.	Gareth Brentnall	Holt Court North, Heneage Street West, Aston Science Park, Birmingham B7 4AX. Tel: 0121-359 5095. Fax: 0121-359 0375
MicroAPL Ltd.	David Eastwood	South Bank Technopark, 90 London Road, LONDON SE1 6LN Tel: 0171-922 8866 Fax: 0171-928 1006
Ellis Morgan	Ellis Morgan	Myrtle Farm, Winchester Road, Stroud, Petersfield, Hants. Tel: 01730-263843
New York SIG APL	Nestor Nelson	PO Box 138, NY 10185-0002, USA
Optima Systems Ltd	Paul Grosvenor	Airport House, Purley Way, Croydon, Surrey CR0 0XY Tel: 0181-781 1812 Fax: 0181-781 1999
Potomac APL SIG	John Martin	Computer Sciences Corp, 1100 West St, Laurel, Maryland 20707-3587 Tel: +1 (301) 497-2698 Fax: +1 (301) 498-8260 Email:jam@acm.org
QB On-Line Systems	Philip Bulmer	5 Surrey House, Portsmouth Rd., Camberley, Surrey, GU15 1LB. Tel: 01276-855880 Fax: 01276-855301
Renaissance Data Systems	Ed Shaw	P.O. Box 20023, Park West Finance Station, New York, NY 10025-1510, U.S.A. Tel: +1 (212) 864-3078

RE Time Tracker Oy	Richard Eller	PO Box 363, FIN-00101 Helsinki, Finland. Tel: +358-0-400 2777
Rochester APL	Gary Dennis	Soliton Associates, 1100 University Avenue, Rochester, NY 14607 Email: gsd@ipsalab.tor.soliton.com
The Rochester Group Inc.	Robert Pullman	50 S.Union St., Rochester NY 14607-1826, U.S.A. Tel: +1 (716) 454-4360. Fax: +1 (716) 454-5430
Rome/Italy SIG	Mario Sacco	Casella Postale 14343, 00100-Roma Trullo, Italy Email: marsac@vnet.ibm.com
SE APL Users Group	John Manges	991 Creekdale Drive, Clarkston, GA30021 USA
Shandell Systems Ltd.	Maurice Shanahan	Chiltern House, High Street, Chalfont St. Giles, Bucks., HP8 4QH.
SOCAL (South California)	Roy Sykes Jr	Sykes Systems Inc, 4649 Willens Ave, Woodland Hills, CA 91364-3812 Tel: +1 (818) 222-2759 Fax: +1 (818) 222-9250
Soliton Associates	Laurie Howard	Soliton Associates Ltd, Groot Blankenberg 53, 1082 AC Amsterdam, Netherlands Tel: +31 20 646 4475 Fax: +31 20 644 1206 Email:ljh@soliton.com
SovAPL	Alexander Skomorokhov	PO Box 5061, Obninsk-5, Kaluga Region, Russia Email:askom@apl2.obninsk.su
Rex Swain	Rex Swain	8 South Street, Washington, CT 06793, U.S.A. Tel: +1 (203) 868-0131 Fax: + (203) 868-9970
SWAPL	Stuart Yarus	PO Box 210367, Bedford, Texas 76095, USA Tel: +1 (817) 577-0165 Compuserve: 73700,2545
SwedAPL	Glan Medri	Box 16181, S-103 24 Stockholm, Sweden Tel:+46 (8) 96 09 47
Swiss APL User Group		Swiss APL User Group, CH-3001, Bern 1, Switzerland Email: sl@ifi.unizh.ch
Sydney APLUG	Rob Hodgkinson	PO Box 1511, Macquarie Centre, NSW 2113, Australia Tel:+61 2 257 5313
Sykes Systems Inc	Roy Sykes Jr	4649 Willens Ave., Woodland Hills, CA 91364, USA Tel: +1 (818) 222-2759 Fax: +1 (818) 222-9250
Toronto SIG	Marc Griffiths	PO Box 384, Adelaide St Post Office, Toronto Ontario M5C 2J5, Canada Tel:+1(416) 532-0843 Email: marcg@utcc.utoronto.ca Cserve: 76260,3314
Unlware	Eric Lescasse	Tour Neptune, Cedex 20, 92086 Paris la Defense 1, France. Tel: +33 (1) 47-78-78-00. Fax: +33 (1) 40-90-04-11
Wickliffe Computer Ltd	Nick Telfer	76 Victoria Rd., Whitehaven, Cumbria, CA28 6JD. Tel: 01946-692588
Warwick University	Prof. Jeff Harrison	Dept of Statistics, University of Warwick, Coventry, CV4 7AL Tel: 01203-523369
Zark Incorporated	Gary A. Bergquist	23 Ketchbrook Lane, Ellington CT 06029, USA. Tel: +1 (203) 872-7806

The Compass group is a recognised and rapidly expanding international consultancy with a prestigious client base extending across Europe, North America and the Far East. Compass provides detailed action plans to improve IT efficiency by benchmarking against top performing companies. In 1993 alone, the implementation of Compass recommendations reduced our clients' IT costs by more than $600 million world-wide.

Based in Guildford, Compass R&D is responsible for the development of a range of software products used by the group and our clients. Compass has been committed to APL since 1984 and a number of key software products have been developed using APL*PLUS/PC and APL*PLUS II. We are now looking for an accomplished specialist to join our high calibre team.

Senior Software Engineer (APL)

You will assist in the development and support of our existing APL systems including a large database and modelling system running on PCs at sites throughout the world. Plans for 1995 include integrating this system with Windows using APL*PLUS III, ODBC, DDE and MicroSoft Office products.

Accordingly, you will require good APL development experience using APL*PLUS II/III or Dyalog APL and the ability to deliver quality software on time. Familiarity with the Windows development environment and Visual Basic or C would be an advantage.

Compatible with the expertise and commitment of our employees, Compass R&D offers a highly competitive remuneration and benefits package.

If you are interested in this challenging position and would like to join our successful and highly motivated team, please reply to Ruth Ramsay, Compass R&D, 10, Frederick Sanger Road, Surrey Research Park, Guildford, Surrey GU2 5YD. Tel: (01483) 302249. Fax: (01483) 302279

RECENT MEETINGS

This section of Vector documents all British APL Association meetings, and any other events of interest to the APL world. If you have recently attended any gathering which you feel would be interesting to Vector readers, please let the Editor have a brief note, and we will include it here.

In this issue, we continue the documentation of APL94 with the complete set of foils from Gérard Langlet's presentation on human vision. We also have in full one of the papers from the meeting at Frankfurt organised by Dittrich and Partner, and a brief résumé of Adrian Smith's seminar to FinnAPL on the Causeway platform.

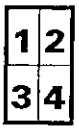
APL 94 at Antwerp
The APL Theory of Human Vision

by Gérard A. Langlet

We are glad to be able to bring our readers the foils from Gérard Langlet's presentation of this paper at Antwerp.

These foils can be read like a book if you take them in the right order. They are reduced to an eighth of their original size, but we hope they will be legible enough.

The order to read them in is:



What does "by" mean ?

What does "-" mean ?

What does "+" mean ?

"-" is the **presence** of 1 electron
e^-

"+" is the **absence** of 1 electron

i.e.

"+" is the **presence** of 0 electron

"+" is a VOID, a HOLE

APL94, Antwerp

The APL Theory of Human Vision

Gérard A. Langlet, CEA/DSM/DRECAM/SCM/LIT

Centre d'Etudes de Saclay

F-91191-Gif sur Yvette, France

An APL game for the electrons

```
+ by + is +

+ by - is -

- by + is -

- by - is +
```

Benjamin Franklin (1706-1790)

1 is ONE **QUANTUM** of *MASS*

1 is ONE **QUANTUM** of *CHARGE*

0 is ONE **QUANTUM** of NO-*MASS*

0 is ONE **QUANTUM** of NO-*CHARGE*

then...

APL transcription :

```
0 ≠ 0 is 0

0 ≠ 1 is 1

1 ≠ 0 is 1

1 ≠ 1 is 0
```

For a couple (A,B) of Quanta

```
A ∈ 0 1

B ∈ 0 1
```

Inital state : (A,B)

Next state : (A,A≠B)

i.e.

Next state : (≠\A,B)

APL2 economy : (≠\A B)

more generally for a CHAIN

i.e. a VECTOR V←A B C ...

```
≠\V
```

What is Vision ?

Optics, Biochemistry & Electricity

What is Biochemistry ?

Chemistry of "living entities"

What is Chemistry ?

The Science of Bonds

What are the Bonds made of ?

Electrons

The Least-Action Principle

for INFORMATION

"*Nature is thrifty in all its actions*" (1744)

Maupertuis (1698-1759)

Elementary NO-ACTION : $\omega \leftarrow \omega$

Elementary ACTION : $\omega \leftarrow \sim\omega$

Controlled Least-Action (Decision Theory) :

IF α THEN $\omega \leftarrow \sim\omega$
[ELSE $\omega \leftarrow \omega$]

APL transcription :

Inital state : (α , ω)

Next state : $(\alpha , \alpha \neq \omega)$

i.e.

Next state : $(\neq\backslash\alpha , \omega)$

Restoration
(*reversible* computation) :

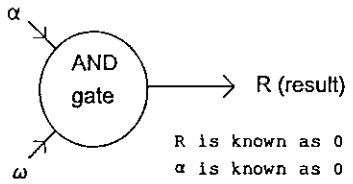
$(\alpha , \omega) \equiv (\neq\backslash\neq\backslash\alpha , \omega)$

Reversibility warrants
isentropy
i.e.
NO NOISE, NO LOSS
of INFORMATION

"*We only perceive Differences*"

Kristiaan Huygens (1678)
(Traité de la Lumière)

Example of Kurt Gödel's
undecidability :



R is known as 0
α is known as 0
What was the value of ω ?
0 or 1 ?

Undecidability may occur
with all Boolean functions
except ~ = and ≠

Only 3 functions warrant
absolute *isentropy* or
non-Gödelian behaviour :

NOT

EQUAL

UNEQUAL

NOT ω is $1\neq\omega$

α EQUALS ω is $\neq/1 , \alpha , \omega$
i.e.
the last item of $\neq\backslash 1 , \alpha , \omega$

with arguments 1 α ω

in ANY order

Gödel's theorem (1931) cannot
be proven anymore if one
removes the axiom of
ordered sets

The result of $\neq/$
which composes every item of
a result given by $\neq\backslash$
is independent from the order
of the arguments
(commutativity)

Nature is non-Gödelian :

It always decides.

Quantum Mechanics is
Gödelian

Parity logic with $\neq\backslash$ is not

The Theory of NO-VISION

Louis Braille (1809-1852)'s Alphabet for the Blind (1825, 1829)

I S O A P L

The 3D integer-modulo2 code is *difference-scanned* in parallel from left to right in 3 rows by the sensitive fingertips

Information is seen and understood at the same time

Receptors in retina (rods and cones) are arranged in a close-packing with 3-fold (triangular) symmetry

An example of neurobit-packing

Every horizontal row is the row above, difference-scanned with a half-position shift

Every row, parallel to the sides of a Δ character is its neighbouring parallel row, difference-scanned

≠\ is, mathematically, the modulo2 equivalent of undefined integration for continuous functions and of discrete numeric cumulation +\

≠\ is also the modulo2 equivalent of −\

≠\ is a difference-scanner which produces differences without damaging information

≠\ ciphers and deciphers information within itself, contrary to scalar ≠ which requires a cryptographic key with the same length in bits as the message

Every row, parallel to the sides of the Δ character contains, in bits, the modulo-2 equivalent of a Fourier (1768-1830)-transform

≠\ performs two very fast FFT-like transforms without difficulty or truncations, in modulo2 integer arithmetics, directly in the neurobit-network

One is the FCT : Fast Cognitive Transform

The other one is the FHT : Fast Helical Transform

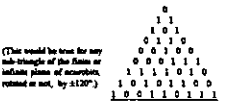
at the speed of electron jumps

Typos in the paper

(Quote-Quad, Vol. 25 No 1)

Page 109 near the bottom of the first column :

!The nine bits or pixels, underlined below **Fig. 1** about the retina topology : 1 0 1 1 0 1 1 1 form an *APL* vector, which, right-matrix-multiplied by matrix M above, produce 1 1 1 0 0 1 1 0 i.e. the left side of the following finite-difference triangle, extracted from the same figure with black/light pixels :



!

Two 0s (in bold below) are missing :

The nine bits or pixels, underlined below **Fig. 1** about the retina topology : 1 0 0 1 1 0 1 1 1 form an *APL* vector, which, right-matrix-multiplied by matrix M above, produce 1 1 1 0 0 0 1 1 0 i.e. the left side of the following finite-difference triangle, extracted from the same figure with black/light pixels :

Fractal Matrix Recipe

Take a *primordial* parity sequence B in bits, e.g. 1 followed by 15 zeros :

```
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Iterate 16 times : B←≠\B

Fill the rows of a G-matrix, a 16-geniton with the 16 integrals of B :

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0
1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0
1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0
1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Use this matrix three times as an *ante-factor* or a *post-factor* for the modulo2 matrix product with any sequence S of 16 bits e.g. your initials in the PC's ⎕AV character code

```
GS1← G ≠.∧ S
SG1← S ≠.∧ G

GS2← G ≠.∧ GS1
SG2← S ≠.∧ SG1

GS3← G ≠.∧ GS2
SG3← S ≠.∧ SG2
```

First, both vectors GS3 and SG3 should reproduce your initial sequence S because the fractal matrix G is a 3-fold symmetry operator :

It is always equivalent to a rotation matrix which performs what you can hardly do with double-precision complex arithmetics using Euler's factor

$$e^{2\pi i/3}$$

In other words, for any size, from 2 to infinity, this matrix acts as an exact binary code of j the complex cubic root of 1.

And the inverse matrix is also its modulo2 matrix square

```
G≠.∧ G
```

or its hypercomplex conjugate which is obtained by symmetry - so without computation - as either :

```
Φ⊖G
```

or :

```
⊖ΦG
```

How simple is *neuro-computing*, expressed in APL !

But do not try to use this matrix method to transform a sequence the shape of which would reach a Gigabit..., unless your APL WS is large enough to contain the G matrix :

```
      ρG
1073741824 1073741824
```

Rather use the *magician*'s algorithm, given in the paper and in Les Nouvelles d'APL N°11 whose execution time is in $(O)^N$ while the matrix product is in $(O)^{N^2}$

Fortunately, our left eye is wired to the right brain and conversely :

When analysed in detail, symmetries induced by ≠\ appear to be able to replace the whole of classical computing...

If the shape of B is a power of 2, then the geniton is a symmetric matrix so that a left- or a right- matrix product returns the same result :

GS1 (or SG1) is the FHT, reversed (mirrored) while GS2 (or SG2) is the FCT, reversed (mirrored)

Recollection

If applied to sequence S directly , 16 successive iterations of ≠\ produce P a fractal PARITON matrix the last row of which is

```
S≡,P[ρS;]
```

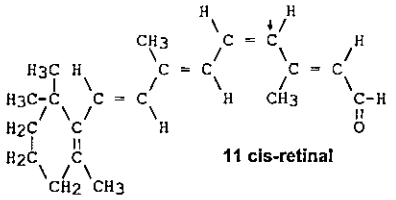
The FCT will then be automatically found in the *last* column on the right (⎕IO≡1) :

```
FCT≡,P[;ρS]
```

while the FHT will be automatically found as the second diagonal of P :

```
FHT≡1 1 ⌽⊖P
```

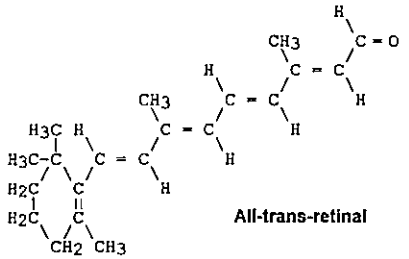
(a) the organic molecular computer at rest



11 cis-retinal

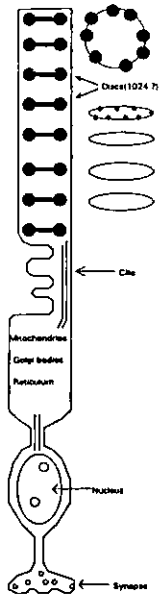
The small down-arrow ↓ points to the carbon atom (number 11) around which the carbon chain jumps when a photon activates the sleeping computer : symmetry changes from *cis* (bending on the same size) to *trans* (maximum linear extension) :

(b) the same computer, in extension, ready to operate



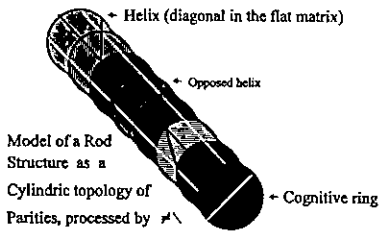
All-trans-retinal

Now, the symmetry of bonds is ternary around each node (carbon atom) in the chain of the pigment.



0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0

0	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0
0	1	0	1	0	1	0	0	0	1	0	0	0	0	0	0
0	1	1	0	0	1	1	1	1	0	0	0	0	0	0	0
0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1
0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1
0	1	1	0	0	0	0	1	1	0	0	1	1	0	0	1
0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1
0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	0
0	1	0	1	0	1	0	0	0	0	0	1	0	1	0	0
0	1	1	0	0	1	1	1	1	1	1	0	0	1	1	1
0	1	0	0	0	1	0	1	0	1	0	0	0	1	0	1
0	1	1	1	1	1	0	0	1	1	0	0	0	0	1	0
0	1	0	1	0	0	0	1	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0
0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0



Model of a Rod Structure as a Cylindric topology of Parities, processed by ≠\

Postulatum

The Law(s) which govern(s) the evolution of what we are able to perceive should be the same as the Law(s) which govern(s) the evolution our perceptions themselves

in particular HOW we SEE,

and the same as the Law(s) which govern(s) the evolution of ourselves.

APL, as a complete (with the mathematical meaning) language for the description of dynamical processes, i.e. sequences of actions, i.e. algorithms,

becomes the best TOOL for studying (scanning), modelling, propagating (teaching) the KNOW-HOW, with simple & efficient expressions, which, by their existence as very-short-although-fast programs, will bring the necessary proofs to the theory.

Double-Helix Data Structures

```
      ⎕←D←DIAGO⍳8⍴2   ⍝ DIAGO is 2⍴⎕IO
0 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0
⇐             ⇒ ⇐             ⇒

      ⎕←DD←DIAGO⍳8⍴8⍴2
0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 0
⇐             ⇒ ⇐             ⇒
```

one has extracted a double helix from the periphery of the rod (cylinder)

```
D[⍳8] ≡ DD[⍳8]
```

(Typos in the text p. 115, right column : remove APL-character ";" from the expressions)

```
      2 8⍴D                  ≠⌿2 8⍴D
0 1 0 1 0 1 0 0  │  0 1 0 0 0 1 0 0
0 0 0 1 0 0 0 0  │

      2 8⍴DD                 ≠⌿2 8⍴DD
0 1 0 1 0 1 0 0  │  0 0 0 1 0 0 0 0
0 1 0 0 0 1 0 0  │
```

In all such double-helix data structures, any half among the 3 different halves, is the ***logical difference*** (or the integer modulo2 sum) of the other two, automatically

The main Law of Genetics :

If a gene is *dominant*, it acts,

If a gene is *recessive*, it does not act.

At the Quantum level of Action, this is the SAME law as the APL formulation of the Least-Action Principle (LAP) for information processing :

```
≠⌿
```

A major question about helix chirality (i.e. *either* left-handed *or* right-handed double helices) is now solved :

Only one diagonal of the Pariton matrix, the second diagonal,

has the property of containing

a Fast Transform of the information contained in the last row,

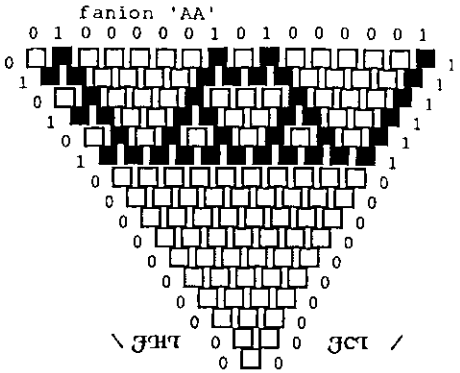
and also a Fast Transform of the information contained in the last column (the Cognitive Ring in the cylindric topology)

"Living" DNA/RNA, as well as the 7-helix molecule (*rhodopsine*) which fixes retinal in the rod, are always **right-handed** helices

The equivalent of *Maxwell*'s law, now for information processing, in right-handed *corkscrew* structures, becomes, for any sequence S :

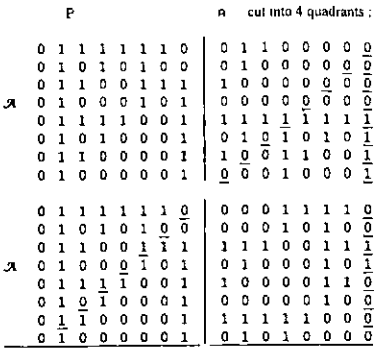
(COG S) ≡ ΦHELΦS
(HEL S) ≡ ΦCOGΦS

with COG & HEL *APL* fns which compute both transforms



The "neurobit-system" detects periodic replications and *compresses* them, filling both the ⊃⊂⊤ and the ⊃⊣⊢ with 0s.

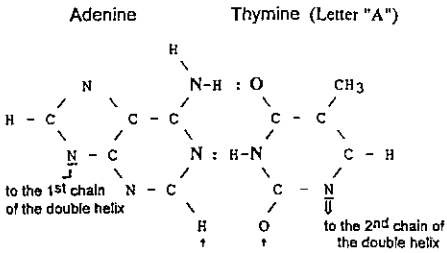
The "neurobit-system" detects palindromes : the ⊃⊂⊤ and the ⊃⊣⊢ become symmetric of one another.

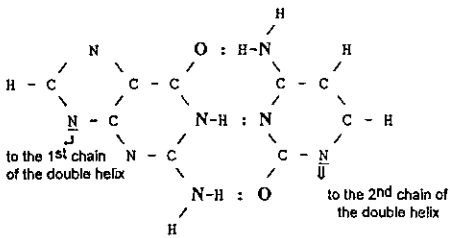


$\mathcal{A} \equiv \mathcal{B} \neq C$
$\mathcal{A} \equiv C \neq \mathcal{B}$
$\mathcal{B} \equiv C \neq \mathcal{A}$
$\mathcal{B} \equiv \mathcal{A} \neq C$
$C \equiv \mathcal{A} \neq \mathcal{B}$
$C \equiv \mathcal{B} \neq \mathcal{A}$

$\mathcal{A}$ is itself a P matrix, etc...

The Code of *Essential* Information





General isomorphism of information in modulo2 integer matrices

In G2 : 1 1 any row or column is ≠\ of the other one
1 0

Sex matrix

X X
X Y

Spin-state matrix

↑ ↑
↑ ↓

Agglutinogen matrix

A A
A O (same for O B)

DNA (A-T) matrix

N N
N O

Synapse ion matrix

Cl^- K^+
Cl^- Na^+

Microtubule matrix (*)

α α
α β

(*) After Michael Zaus, Berichte aus dem Institut für Kognitionsforschung, No 18 : "*Theoretische und andgewandte Paritätslogik*", Carl v. Ossietsky Universität Oldenburg (1994), p. 6.

G2 is the *growth matrix*; it is Fibonaccian, (so are its rotations in the plane; e.g. in classical algebra, the Golden section Φ is an *eigenvalue* of ⊖G2 the other one being $-1/\Phi$, this also holding for all the matrix powers of ⊖G2).

ΦG2 and ⊖G2 are the involutive modulo2-self-inverse operators (matrices) for the FHT and the FCT which compress and analyse information, already the modulo2 equivalents, for a two-bit information, of a FFT. ⊖G2 and G2 are the modulo2 square and matrix inverse of one another (j and j^2 in complex algebra).

- Human language, music, electroencephalograms, are all 1/f signals, of which Paritons offer a plausible mathematical model.

- Topologies which process or keep information in biological entities are not random : *ternary bit networks* (retina patterns), *cylindric data structures* (rods), *conic data structures* (cones), *double-helix data structures* and *rings* (cognitons) have very special properties which immediately appear thanks to the symmetries induced by the ≠\ motor.

- The *genetic code* itself is a paritonic data structure (so are all parity data codes for electron spins, sexes, agglutinogens, microtubules, ions in synapses).

- ≠\ is the correct formulation of the least-action principle for elementary, *isentropic* then fiable data processing.

- No Natural Process is Random.

Main Conclusions

- Our senses (inter alia vision) process information in bits, NOT as continuous signals (with potentials, fields and smooth derivatives).

- The responsible mechanism may be described by ≠\ (ISO8485, Geneva, 1989).

- ≠\ is the quantum description of electrodynamics, the correct algebra for the interpretation of quantum states (*either* 0 *or* 1) being modulo2 integer algebra, in which ≠ (XOR of the isomorphous logical algebra) corresponds to ⊕ (PLUS or MINUS modulo 2).

- Inventors reproduce the way they are themselves programmed : e.g. the braille code for the blind, which looks like DNA code, expressed in bits with *either* 0 for Oxygen *or* 1 for Nitrogen.

Acknowledgements

Victor Besson (software testing)

Claude Cortet (discussions, pure mathematics, proofs of theorems)

Jean Delaunay (Macintosh &video support)

Dr Jean Hainaut (medicin)

Dr Jacqueline Langlet (Thesis on retinal, theoretical-chemistry work on DNA base-pair stacking).

Bernard Mailhol (discussions, APL2, OS/2 & SHARE support)

Louis Métayer (discussions, mathematics, APL*PLUS II)

Miss Moulft, Institut des Jeunes Aveugles, Paris, (for Braille documentation)

Dr Michael Zaus, Univ. Oldenburg, Germany, (for the supply of reprints, and his interest for the propagation of "≠\ logic")

Die Programmiersprache APL 13th October, Frankfurt

notes by Adrian Smith

Background

This was a one-day seminar covering all aspects of GUI programming in APL. It was organised (most professionally) by Dittrich & Partner Consulting of Hanover, and was attended by over 40 delegates from all branches of German commerce and industry. The talks covered:

- an introduction to Dyalog/W by Peter Donnelly
- the APL2 product range for OS/2 and AIX by Nancy Wheeler
- Causeway "Ein sicherer Weg über den Gui" by Adrian Smith
- An APL2 data bank by Klaus-Peter Friedrich (Rheinischer Sparkassen)
- APL*PLUS III version 1.2 by Stefan Denker (Dittrich)
- Migration from mainframe to PLUS III by Dieter Düren (CNL)
- Using APL*PLUS II for DOS by Christine Biewer (Cosmos Direkt)

Most of the talks were based around a live presentation, and I hope the audience survived my English better than I survived their German! Fortunately the paper by Dieter Düren was accompanied by a comprehensive handout, so we have had time to translate it for you. Thanks to Gill Smith for the translation.

Thanks to Peter Donnelly for arranging the Causeway session, and to Dittrich and Partner for their hospitality.

Conversion from APL2 (IBM Mainframe) to APL*PLUS III (Manugistics)

by Dieter Düren
(Colonia Nordstern Lebensversicherungsmanagement AG
= Colonia Northstar Life Assurance Management Ltd.)

I. Hard- and Software:

Mainframe: . . . 2 IBM/390 under MVS/TSO

PC: . . . . . 486DX and 486DX2 under DOS 5.0–6.2/Windows 3.1
Novell Network and Stand-alone PC

APL: . . . . . 1. IBM Mainframe APL2 Version 2.1.00
2. IBM APL2-PC Version 1.02 (DOS and Windows)
3. Manugistics APL*PLUS III Version 1.2 (Windows)

II. Overview

IIa. User-Departments

APL will be installed in the following sections:

1. Business Plan
2. Balance Sheet
3. Re-insurance
4. Forward Sales
5. Proposal
6. Customer Service

Number of APL programmers: c. 30 people
Number of End-users: >120 people

IIb. Applications

1. Product development (PC), e.g. programs for calculating premiums, surrender values, surpluses, and the analysis of (product) data.

2. Technical individual calculations [i.e. for individual policyholders] (mf with DB2), e.g. for altering contracts or investigating surrender values.

3. Supporting the Sales department (mf and PC with interfaces to Word and Excel) , e.g. offering programs for special products [bespoke programs for one-off solutions].

4. EDP (Electronic Data Processing) — compensation applications (mf with DB2, GDDM, DCF, AFP, ISPF/DTL-panels), e.g. documentation programs for special products, or altering proposals.

5. Draft and testing of the EDP programs for stock control (mf and PC).

6. Balance sheet and calculating profits (PC, FS-Panel, Word, Excel).

7. Risk Analysis (PC, Word, Excel).

8. Calculating premiums, contribution tables [lit: profit/gain share tables] and example calculations.

9. Managing the system of the re-insurance department (PC, FS-Panel, Word, Excel).

III. Why are we installing APL?

1. APL contains powerful functions or, rather, signs/characters which are derived from mathematical logic.

2. With APL it is possible to portray formulae in a "speaking" form. Because of this, quick development and changes in the interpreter system are possible.

3. APL supports many [different] data structures (scalar, vector, matrix) with general functions, which are applicable for all structures.

4. Quicker building of screen panels through FS-Panel (PC) and ISPF/DTL panels (mf) or internal tools (AP124) is possible.

5. Complete and easy control of print queues and printers (NEC dot-matrix, HP Laserjet) is possible.

6. The interpreter allows easy error-tracking through stop-vectors. Stepping through programs, and controlling or altering data in the workspace during execution is possible, leading to shorter development times.

7. Mainframe and PC APL are almost compatible.

8. ...

IV. Why have we converted?

IVa. Mainframe >> PC

1. Saves CPU costs
2. Response times
3. Independence from mainframe
4. Individual system configurations
5. Quick and higher quality printing
6. Multi-tasking

IVb. APL-IBM >> APL*PLUS III

1. APL*PLUS runs under Windows 3.x, Windows for Workgroups 3.x and Windows NT. According to Manugistics it also already runs under beta versions of "Chicago". APL*PLUS will be developed for a 32-bit operating system. Partly through "downgrading" it has been adapted to run on the 16-bit Windows versions.

2. The APL2/2 versions much advertised by IBM were not available for delivery for months for the "IBM Standard". The demo version we tested crashed as soon as the Editor was called. It was recognisable that IBM would keep the concept of partner-programs. IBM tried to break into the market with dump-pricing of APL2/2 and also OS/2.

3. Manugistics has kept to its timely promises concerning updates before.

4. IBM has kept us waiting more than a year for APL2-PC Version 1.02. The ability to run under Windows was not mentioned in the official documentation. Rumour has it that IBM has reduced the number of APL2 developers.

5. IBM is holding back on the notification/advertising of an APL2 Windows version.

6. The market, our customers, use Windows almost exclusively.

7. The execution speed is c.100% higher with APL*PLUS.

8. The printing of APL characters is possible on every Windows-capable printer.

9. [! — somebody can't count!]

10. APL*PLUS is a Windows application, that is, all Windows features (Clipboard, DDE, etc.) are available, and the co-operation with other Windows applications (Word, Excel, etc.) is guaranteed. Multi-sessions (MDI) are possible.

11. Built-in additional features:
    - Windows dialog box editor (WED)
    - Debugger
    - Online Help
    - Function calls to external Codes (16- and 32-bit DLL, VBX, etc.)

12. Lots of service functions and examples are supplied.

13. The possibility exists of defining one's own user-commands. One can store these outside the workspace.

14. Instead of APs many system functions are supplied, which can also be called from functions.

15. Control structures (IF, WHILE, FOR, etc.) → quicker and more readable code.

16. Data handling results from simple but effective system functions.

17. Two keyboard layouts are available (national and "classic" APL-keyboards).

18. Runtime versions can be distributed free.

19. Limited hardware prerequisites (no co-processor necessary, though recommended).

20. Qualified support through Dittrich & Partner. Dittrich & Partner themselves now only program in APL*PLUS.

21. APL*PLUS will be constantly developed.

22. Manugistics are working on the development of APL*PLUS very closely together with Microsoft.

V. Disadvantages of Conversion

1. APL*PLUS is not fully compatible with IBM-APL2 (for example Format by Example, Partitioned Enclose, Take and Drop with axis are impossible, Evolution Level; see also Appendix 1).

2. All programs with interfaces (Input, Output, Data Handling) must be newly installed or revised.

3. Some programmers have special skills in APL2-PC deeply engrained (e.g. storing data in special APL Data structures) and cannot now change.

4. Lots of manpower is needed to convert old workspaces.

5. "Old hares" [experts in the old techniques] can only be brought round from a text-oriented development environment to the graphically-oriented Windows with great difficulty.

6. A re-training in Windows and APL*PLUS is necessary.

7. APL*PLUS is more expensive than APL-PC from IBM.

8. Without previous knowledge of Windows programming in C and/or C++ one cannot adopt low-level Windows programming in APL*PLUS.

9. The "classic" keyboard layout is not completely identical to the German IBM-keyboard. Three keys are differently arranged.

10. Extra/additional hardware must be bought for the transfer from the mainframe to the PC.

VI. Conversion Procedures

Because of a lot of "old applications" which we must maintain, and because of the introduction of new tariffs/rates from 1/7/94, we could only convert the mathematical core of our new insurance calculations up to now. At the same time we have proceeded as follows:

1. Analysis of the two APL dialects concerning syntax and functionality.

2. Installation of search functions, which should localise the difficult statements, and if found alter them. The complex functionality and the abundance of variables in these statements (e.g. Partitioned Enclose, Format by Example, Activate) led us however to throw out this idea.

3. Should we obtain the complex functionality of these problematic statements?

4. Because the answer to the previous question is no, we have decided to transfer our new maths core from mainframe to APL*PLUS, and to start work on the main functions by trial and error.

5. So it turned out that in most cases of the previously found differences only small/limited problems showed up. In c. 90% of cases a syntax can be found which runs on both platforms, that is we used statements from an intersection of both dialects. In other cases we installed platform-dependent functions (see Appendix 2).

6. New applications, which will only run on the PC, will only be installed with APL*PLUS.

7. We will progressively convert almost all our "old" applications, either when the need arises or when we have time. We assume that this conversion will be finished in about 3 years.

8. In our new applications we are optimising the code for installation of runtime-modules (e.g. control structures), in order to achieve a faster execution speed. Through analysis we use the supplied MFFNS functions of the workspace. In the maths core for insurance calculations, however, we only remove (all) comments and insert Diamonds. This last measure increases the execution time by c. 5%. All measures together give an improvement of c.10%.

Our Conclusion

APL*PLUS III will run on the operating system with the largest market acceptance. It offers a development environment that is state of the art. It shortens the development time of powerful applications with standard interfaces, and can be delivered free to customers on demand [using runtime] in the form of Windows applications. The printed output can be sent to standard tools (Word, Excel), and qualitatively higher value results [i.e. better/higher quality results]. In spite of all the disadvantages we believe that the advantages of APL*PLUS outweigh them, and that the future will belong to APL*PLUS.

Appendix 1: Problems of the Evolution Level

APL*PLUS III uses the switch Evolution Level (=0, 1 or 2) so that source code that was written under previous versions should still run. In the course of APL*PLUS development the implementation has changed some APL characters.

If one sets the Evolution Level to 0(`)evlevel 0`), all functions in which this problem appears crash with the error message (*EVOLUTION ERROR*).

The Evolution Level will not be stored with the respective workspace. On starting APL*PLUS III Default Level 2 is installed.

As model examples of the above problem:

```
      ↑       ⍝ First ...
      ↑⍳10    ⍝ at )evlevel 0
EVOLUTION ERROR
      ↑⍳10    ⍝ at )evlevel 1
1 2 3 4 5 6 7 8 9 10
      ↑⍳10    ⍝ at )evlevel 2
1
```

```
      ⊂ ⍝ Partitioned Enclose
      1 1 0 1⊂'1234'  ⍝ ev 0
EVOLUTION ERROR
      1 1 0 1⊂'1234'  ⍝ ev 1
1 23 4
      1 1 0 1⊂'1234'  ⍝ ev 2
NONCE ERROR
```

Appendix 2: Some Platform-dependent Functions

```
erg←la MIG_FO ra
⍝ Düren 06.07.94
⍝ Abhängig vom jeweiligen APL (APL2-Host, APL2-PC und APL*PLUS III) wird
⍝ bei IBM "IBM-Format by Example" verwendet. Für APL*PLUS wird "Manugistics-
⍝ Format by Example" verwendet.
⍝
→(⎕AV[24]≠'ω')/IBM
⍎((1=⍴,la)∧∧/la='S')/'erg←((⍕⍴,ra),''I1'')⎕fmt ra'
⍎((4=⍴,la)∧∧/la='S')/'erg←((⍕⍴,ra),''I4'')⎕fmt ra'
→0
IBM:erg←la⍕S

erg←MIG_PDH ra
⍝ Düren 06.07.94
⍝ Abhängig vom jeweiligen APL (APL2-Host, APL2-PC und APL*PLUS III) wird
⍝ bei IBM die "paarweise Differenz von hinten" gebildet. Für APL*PLUS wird eine
⍝ nachgebaute Version verwendet.
⍝
→(⎕AV[24]≠'ω')/IBM
erg←(1↓ra)-¯1↓ra
→0
IBM:erg←¯2-/ra

erg←la MIG_PE ra;d1;d2;i
⍝ Gierling 07.09.94
⍝ Abhängig vom jeweiligen APL (APL2-Host, APL2-PC und APL*PLUS III) wird
⍝ bei IBM "Partioned Enclosed" angewendet. Für APL*PLUS wird eine nachgebaute
⍝ eingeschränkte Version von "Partioned Enclosed" verwendet.
⍝
→(⎕AV[24]≠'ω')/IBM
→(1=d1←+/1(+/ra='(')(+/ra='-')(+/ra='o'))/0,0⍴erg←,⊂ra
i←1,0⍴d2←0,(d1←d1/⍳⍴d1←(ra='(')+(ra='-')+ra='o'),1+⍴ra,0⍴erg←d1⍴' '
lab:→((⍴d2)<i←i+1)/0
erg[i-1]←⊂ra[d2[i-1]+⍳d2[i]-d2[i-1]+1]
→lab
IBM:erg←la⊂ra

erg←MIG_PG ra
⍝ Düren 05.07.94
⍝ Abhängig vom jeweiligen APL (APL2-Host, APL2-PC und APL*PLUS III) wird
⍝ bei IBM die "paarweise Gleichheit" angewendet. Für APL*PLUS wird eine
⍝ nachgebaute Version verwendet.
⍝
→(⎕AV[24]≠'ω')/IBM
erg←(1↓ra)=¯1↓ra
→0
IBM:erg←2=/ra

erg←la MIG_RE ra;ind
⍝ Schehl 06.07.94
⍝ Abhängig vom jeweiligen APL (APL2-Host, APL2-PC und APL*PLUS III) wird bei
⍝ IBM "Reduziere Each" angewendet. Für APL*PLUS wird eine nachgebaute Version
⍝ verwendet. 1 0 1/⍤(,2) 0 1
⍝
→(⎕AV[24]≠'ω')/IBM
ra[ind]←⍤⍤ra[ind],0⍴ind←(1=⍤⍤⍴⍤⍤ra)/⍳⍴,ra
IBM:erg←la/⍤⍤ra

erg←la MIG_RN ra;ind
⍝ Schehl 06.07.94
⍝ Abhängig vom jeweiligen APL (APL2-Host, APL2-PC und APL*PLUS III) wird bei
⍝ IBM "Reduziere" angewendet. Für APL*PLUS wird eine nachgebaute Version
⍝ verwendet. 1 0 1/,2
⍝
→(⎕AV[24]≠'ω')/IBM
ra[ind]←⍤⍤ra[ind],0⍴ind←(1=⍤⍤⍴⍤⍤ra)/⍳⍴,ra
⍎(1=⍴,ra)/'ra←⊃ra'
IBM:erg←la/ra
```

Pitkospuut GUIsuon Yli

Causeway in Helsinki – notes by Adrian Smith

Background

October 94 was an eventful month! Having attended the Frankfurt meeting on the 14th, I flew straight out to Helsinki for a most enjoyable weekend, before settling in for a hard day's seminar on the Monday. This was at the invitation of FinnAPL, and we had the use of a splendidly equipped lecture hall at the Finnish timber agency METSA. The day was designed as a follow-up to Swansea, with the objective of introducing a varied population of APLers to the strange new world of Windows programming. We covered the basics of event-based design, with an hour or so of hand-waving around the well-known Windows game of Minesweeper. In my view, an ability to (a) play and (b) design and code this game should be a pre-requisite for any serious Windows programmer. The rest of the day was strictly about APL, and how we can bring our existing APL skills and ideas to bear in this slightly frightening new programming paradigm.

What is Causeway?

Causeway is an architecture for portable GUI development with APL. It is:

- a published and documented standard, freely available
- supported by the British APL Association and encouraged by many other APL clubs around the world
- open to anyone. APLers are positively encouraged to build implementations for the interpreter of their choice
- currently implemented (documented and supported) for Dyalog APL, and available for early testing in APL*PLUS III. These implementations are copyright Adrian Smith and Duncan Pearson respectively.

All the workspaces are available with fully commented source code for you to copy and extend. Causeway offers you:

- faster development
- less complexity
- fewer functions
- more reliable applications

Small systems are realised faster; big systems hurt less – HOW?

How is Complexity Mastered?

The key difference you need to grasp is that you base your design around the Windows objects you see on the screen. These objects already know how to display and update APL variables, but they do a lot more than that:

- objects can be told to watch specific variables, and will update themselves automatically when any watched variable changes.
- objects can react to user-actions through a simple table

  event + condition >> action

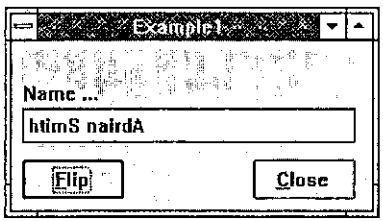
  where the action is any APL expression. Of course it may change some APL variables, in which case other objects will respond as required.
- the main 'container' objects (forms) may have local variables, which are visible to their child forms, and so on down the stack – just like APL functions!

It is hard to believe how much difference this approach can make to development times, so here is a small challenge for you: design and build a Centigrade to Fahrenheit convertor, such that when either number is changed the other immediately flips to match. Ideally, the user should be able to either type in the data, or spin the values from an initial setting of 0=32 with spin boxes. The centigrade figures should spin in 1-degree increments; the Fahrenheit should spin in 2s. *How long do you think this should take you?*

The replies (I tried this in Frankfurt as well as Helsinki) were well spread, and apart from one facetious suggestion of 5 minutes, were all in the half-hour or above range. Most people who thought hard about this (as I hope you will) estimated at least half a day.

Starting with Something Simple

Let's begin with the example from Vector 10.4 – the one which just puts your name on the screen and lets you flip it with a push-button.



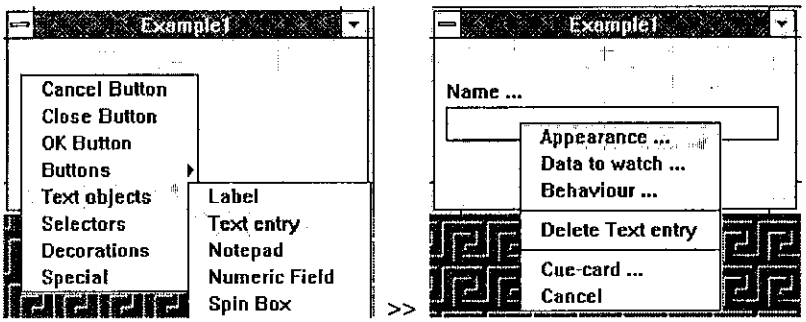
To build this, you start with a quite ordinary APL variable:

```
name←'Adrian Smith'
```

and then build your form with:

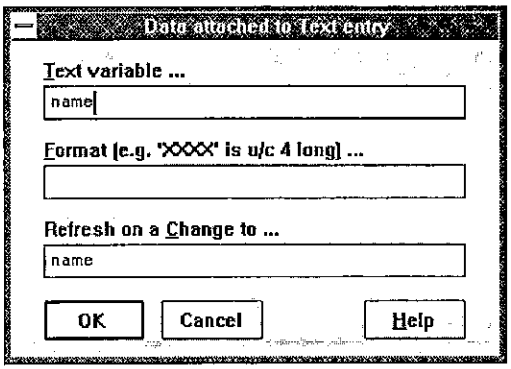
```
Dbx 'example1'
```

The first stage is to add that text edit field using the right-mouse menu:



... which is dragged out to a nice size and captioned with the 'appearance' dialogue (left button on the object). Now we can add a close button and an action button (called 'Flip') and we are ready to start the serious work!

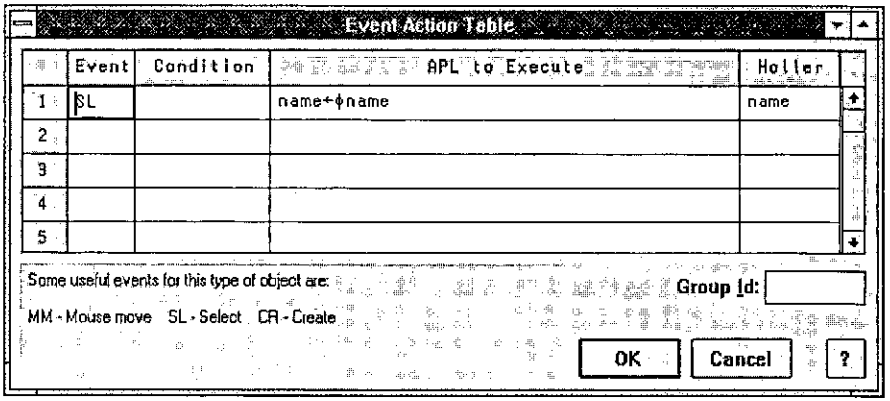
The first stage is to tell the edit field to watch the variable *name*, and to refresh the display whenever that variable changes.



This is the dialogue box you get when you ask for 'data to watch' on any of your objects. All objects have built-in documentation, so the prompts you see are picked up directly from the object definition.

If you define objects of your own, the designer is quite happy to work with them.

Now we should add an action to the push button to flip *name* whenever it is pressed. This will need to execute some APL code, and also 'holler' *name*, so that any other interested objects know that something happened to it.



In the first column, you list any events that you want the object to notice. The second column is an optional condition (any APL expression returning a boolean), and the third column is the code you want to run when that event occurs and the condition (if any) is satisfied. The final column is a list of the variable names which the expression may have modified.

Now you can press the <Run> button on the designer, and your form will work. When you exit to the workspace, *name* will have the last value you saw on the screen.

Stop and think for a moment ... this (admittedly trivial) application requires:

- no functions
- no variables
- no logic

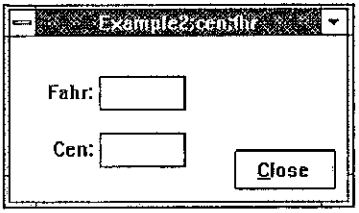
All you coded was a statement of the problem:

```
name←⍎name
```

It will be utterly reliable, because it is too simple to fail.

Moving On

It is interesting to think again about the Centigrade to Fahrenheit problem. Clearly a couple of local variables will be required, and these should be initialised when the form is created (i.e. before it becomes visible on screen). Let's start with a form and a couple of spin-boxes and see how it goes:

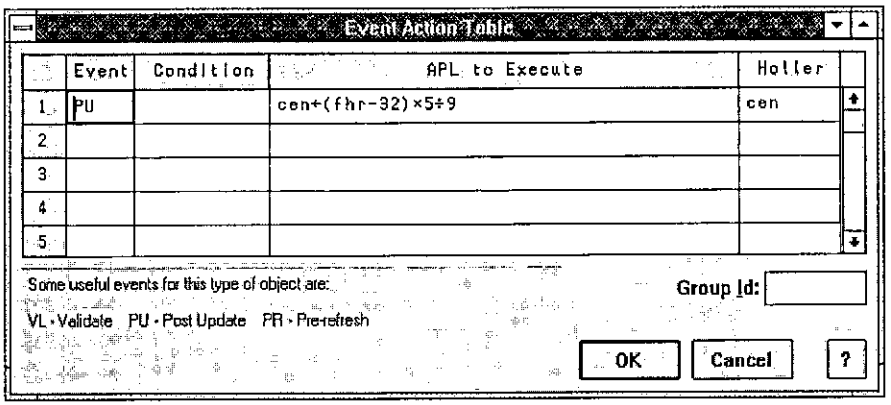


Note that locals are added to the form's titlebar, and can be initialised by setting an action like:

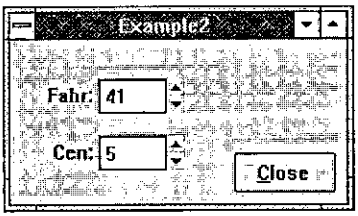
```
cen←0 ⋄ fhr←32
```

which you set on the 'CR' or creation of the form itself.

Now for the tricky bit! Each spin box is set to watch the appropriate variable, and the formatting data is used to set the increment. What we must also do it to catch a 'Post Update' on both objects and holler the name of the *other variable* (which got changed in our APL expression):



Now, when you run the completed form:



Spin either box, and the other follows along. The 'Post-update' trigger is fired just after Causeway has assigned the new value into the variable your object was watching: your code runs, calculates a new value for the corresponding variable, and hollers the name. That is all you need to do!

In Summary ...

No variables, no functions, just two APL statements. Best time to date 2 minutes 29 seconds. *You* stated the problem, *Causeway* handled the GUI.

Now you can be an APL programmer again – just like the old days!

NAMESPACES SPECIAL FEATURE

Namespaces are the most interesting new thing to happen to APL since nested arrays, and in this Vector we have tried to give you a cross-section of opinions on what they are good for, and where the pitfalls and limitations may lie.

This is still a very young technology, and it is up to you to influence its development. Please follow the advice in Duncan's Editorial and experiment with these ideas as much as you can (maybe only on paper if you don't have Dyalog 7 or the latest J release); then let us know what you think and Vector can help form a consensus on what is 'just right', what is harder to use than it should be, and what just got left out entirely.

Peter Donnelly works through a simple (but amusing) tutorial which will help you to grasp the basic ideas that objects can have hidden data. I believe that Vector can claim a publishing scoop here — we are the first APL magazine to reveal the co-ordinates of the *Dyadic Duck*.

Eric Lescasse has contributed a much more technical article which shows you how you can exploit namespaces effectively in building GUI systems.

Finally, Kimmo Kekäläinen offers some suggestions on using the session namespace `⎕SE` to manage your utility set, and some warnings about the dangers of going too quickly into this new world. The feature ends with a return visit to Adrian Smith's 'Coast-to-Coast' game to see how much of his APL code namespaces have replaced.

The Use of Namespaces for Encapsulation: a Practical Introduction

by Peter Donnelly, Dyadic Systems Limited

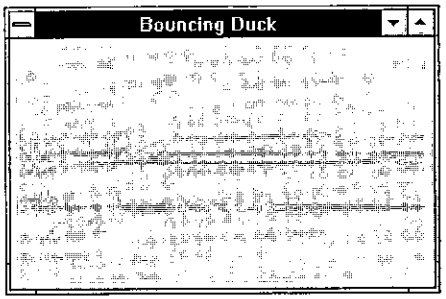
Introduction

In Dyalog APL/W Version 7, Dyadic introduced the concept of *namespaces*. A namespace is a container that may be used to store functions, variables and other objects and provides a separate execution environment which is isolated from the outer workspace and from other namespaces. In Version 7, GUI objects are themselves namespaces and may therefore *contain* any functions and variables they need for their operation. Dyalog APL/W thereby supports ***encapsulation***, an important feature of object-oriented programming. This article is effectively the script of a demonstration that explains how useful this concept is in practice. If you have a copy of Dyalog APL/W Version 7, you will be able to reproduce the entire demonstration by typing in the APL code.

The Demonstration

First we will create a Form called *Dk*. Its Caption is *Bouncing Duck*; it has a Pixel co-ordinate system and is positioned at (300 300) with a size of (175 300):

```
'Dk' ⎕WC 'Form' 'Bouncing Duck' (300 300)(175 300)'Pixel'
```



Now we can step *into* the Form. The system command *)CS* means *Change Space* and is used to switch from one namespace to another. Having changed, *)CS* reports the full pathname of the new current namespace:

```
      )CS Dk
#.Dk
```

Now that we are *within* the Form we can create some child objects. First we need a Static window in which to draw the duck. Note that the Attach property defines how the child object reacts to its parent being resized.

In this case, we want the Static to shrink/expand so that its edges remain a fixed distance from the sides of the Form.

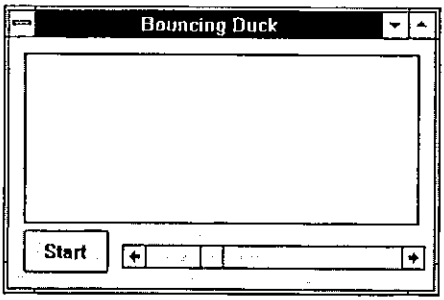
```
'Box' ⎕WC 'Static' (10 10)(120 280)
'Box' ⎕WS 'Attach' ('Top' 'Left' 'Bottom' 'Right')
```

Next we will add a stop/start button called *Stop*. Its Caption (initially) will be "Start", and its Select event will fire a callback function called *START*. This time, the Attach property makes the Button fixed in size and remain a constant distance from the bottom left corner of the Form.

```
'Stop' ⎕WC 'Button' 'Start' (135 10)(30 60)
                    ('Event' 'Select' 'START')
'Stop' ⎕WS 'Attach' ('Bottom' 'Left' 'Bottom' 'Left')
```

Now we will add a scrollbar called *Speed* to input the speed of the duck. Note that, by specifying a height of 0, you get a standard height scrollbar. The HScroll property specifies that it is a horizontal scrollbar. The Range property defines the scale. The Step property defines the amounts (small change and large change) by which it scrolls. The Thumb property specifies the position of the "thumb". The Attach property fixes the height of the scrollbar, but lets it expand and contract horizontally with the Form.

```
'Speed' ⎕WC 'Scroll' (145 80) (0 215) ('HScroll' ¯1)
'Speed' ⎕WS ('Range' 60)('Step' 2 10)('Thumb' 15)
'Speed' ⎕WS 'Attach' ('Bottom' 'Left' 'Bottom' 'Right')
```



The last object we need is a Timer. The job of a Timer is to fire an event at regular intervals. We can animate the duck by attaching a callback function (which draws the duck) to the Timer. Our Timer will fire every 50 milliseconds, but initially it will be inactive.

```
'Timer' ⎕WC 'Timer' 50 ('Active' 0)
```

Whenever the Timer fires it generates a Timer event. We will attach this to a callback function called *DRAW* that resides (and runs) *within* the Duck object, which in a moment we will create as a child of the Static Box.

The function is therefore referenced by its pathname (from here) which is *Box.Duck.DRAW*

```
      'Timer' ⎕WS 'Event' 'Timer' 'Box.Duck.DRAW'
```

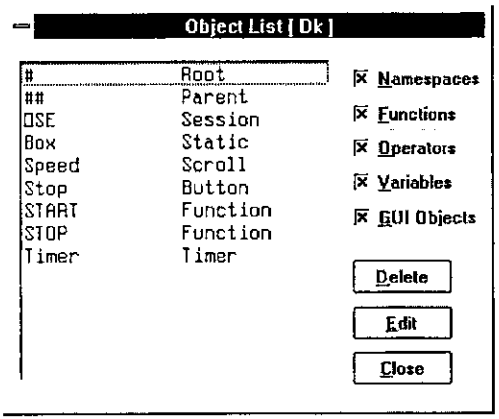
The next thing to do is to write the two functions needed to start and stop the animated display. Let's first write the *START* function:

```
     ∇ START
[1]    'Timer'⎕WS'Active' 1    ⍝ Activate the Timer
[2]    'Stop'⎕WS('Caption' '&Stop')('Event' 'Select' 'STOP')
     ∇
```

The first line of *START* activates the Timer by setting its Active property to 1. The second line changes the Caption of the Button to "Stop" and changes the callback function to *STOP*. (Incidentally, this nicely illustrates how you can control an application dynamically by changing the callback function associated with an event "on the fly".) The *STOP* function is simply the reverse.

```
    ∇ STOP
[1]   'Timer'⎕WS'Active' 0    ⍝ De-activate the Timer
[2]   'Stop'⎕WS('Caption' '&Start')('Event' 'Select' 'START')
    ∇
```

STOP de-activates the Timer by setting its Active property to 0. Then it changes the Caption of the Button to "Start" and changes the callback function back to *START*.



So what do we have now? Let's use the Object List dialog to see what objects and functions we have defined within the *Dk* Form.

Next we need to define the co-ordinates of our Duck. First we will step *into* the Box object in which we want to draw the duck:

```
      )CS Box
#.Dk.Box
```

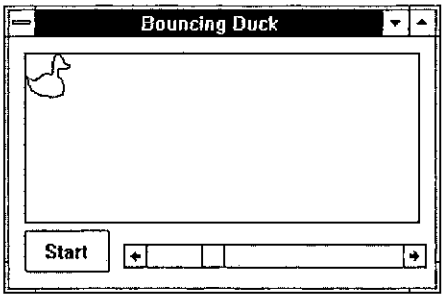
Then set up the co-ordinates in a variable called *DUCK*.

```
DUCK←44 2⍴0
DUCK[;1]←15 17 18 18 17 15 14 13 14 13  9  6  2  0  0  2  6  7
          7  9 10 10 10 11 12 15 17 18 22 25 28 29 30 30 30 29
         29 29 28 27 26 24 20 15
DUCK[;2]← 0  2  4  6  8 10 12 14 17 18 18 18 19 21 23 25 27 29
         30 30 28 26 24 22 22 22 24 25 26 26 25 23 21 19 17 14
         12 10  8  6  4  3  1  0
```

(Naturally, if you have the co-ordinates already set up in a variable in another workspace, you simply copy the variable in. `)COPY` brings objects into the *current namespace* as you would expect.)

Using these co-ordinates, we can now create a Poly object called `Duck`. (Note that in Dyalog APL, graphical objects are not transient things but are **true objects** that generate events and can be manipulated like Forms, Buttons and so forth.) The FStyle and FillCol properties define a solid yellow fill.

```
'Duck' ⎕WC 'Poly' DUCK ('FStyle' 0)('FillCol' 255 255 0)
```



The next step further illustrates the facility in Dyalog APL to ***encapsulate*** code and data ***within*** the object to which they rightfully belong. First, we will copy the variable containing the duck's co-ordinates *into* the Duck object:

```
      Duck.DUCK ← DUCK
```

Then erase the variable from here (`Dk.Box`):

```
      )ERASE DUCK
      ⍴⎕NL 2 3 4    ⍝ Nothing here now
0
```

Now we will step *into* the Duck namespace:

```
      )CS Duck
#.Dk.Box.Duck
      )VARS
DUCK
```

To animate the duck, we need to write the *DRAW* function that is attached to our Timer.

To simplify the code we will first define some *static* variables. A static variable is one that is global to a namespace. It is therefore visible to functions that run in that namespace, but is not visible from outside that namespace. In this case we will use static variables to remember the position (*POSN*) and direction of motion (*DIR*) of the duck between successive calls to *DRAW*.

```
      POSN←0 0
      DIR←1 1
```

Rather than computing it each time, we will also create a variable *SIZE* defining the size of the object. We need this to calculate when it hits a wall.

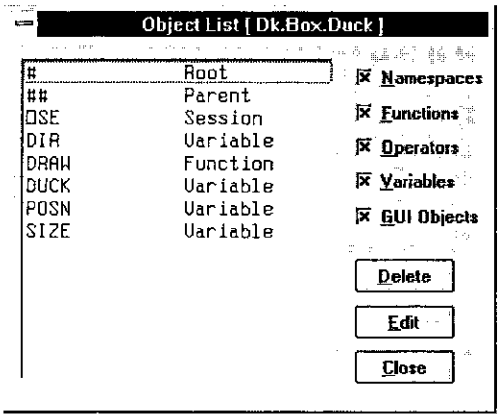
```
      SIZE←(⌈/DUCK)-⌊/DUCK
```

Finally, the *DRAW* function itself:

```
    ∇ DRAW;WINSIZE;SPEED
[1]   WINSIZE←'##'⎕WG'SIZE'                    ⍝ Size of parent window
[2]   SPEED←(×DIR)×'##.##.Speed'⎕WG'Thumb'     ⍝ Speed from scrollbar
[3]   POSN←POSN+SPEED                          ⍝ Calculate new position
[4]   DIR←×DIR×1 ¯1[1+(POSN≤0)∨WINSIZE<POSN+SIZE] ⍝ Does it bounce ?
[5]   POSN←0⌈POSN⌊WINSIZE-SIZE                 ⍝ Update position
[6]   ⎕WS'POINTS'(DUCK+(⍴DUCK)⍴POSN)           ⍝ and redraw
    ∇
```

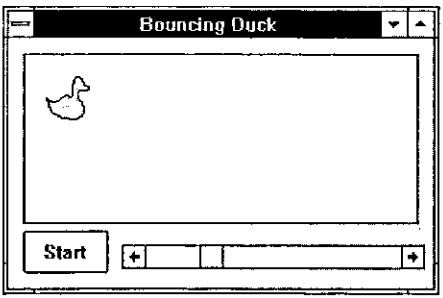
Note that the variables and the function we have created are ***encapsulated*** within the Duck object.

```
      )FNS
DRAW
      )VARS
DIR     DUCK     POSN     SIZE
```



We can test our system by running the *DRAW* function directly from here. You can see how it works using the Tracer.

```
      DRAW  ⍝ Trace it !
```



Now let's switch back to the main workspace ...

```
      )CS
#
```

... and start the system by clicking *Start*. While this is running, the session remains active. We can (for example) directly change the properties of the Duck object; for example:

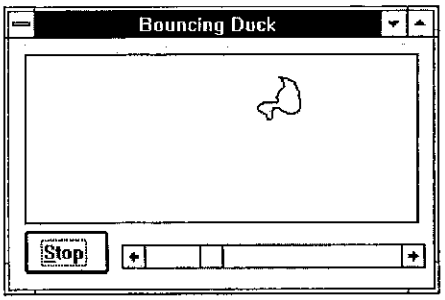
```
      'Dk.Box.Duck' ⎕WS 'FillCol' 255 0 0    ⍝ Red
      'Dk.Box.Duck' ⎕WS 'FillCol' 255 255 0 ⍝ Back to yellow
```

More interestingly, we can step *into* the Duck object and do it from there:

```
      )CS Dk.Box.Duck
#.Dk.Box.Duck
      POSN←10 10
      DUCK←⍎DUCK
```

Encapsulation and Inheritance

So what have we achieved? Essentially, we have produced an Object (called *Dk*) which contains within it all the sub-objects, code and data needed to perform its allotted task; in a nutshell, ***encapsulation***. Having made one object, we can clone it to make others. This introduces ***inheritance***, another important principle of object-oriented programming.




```
      )CS #.Dk.Box
#.Dk.Box
      'Clone' ⎕WC ⎕OR 'Duck'
```

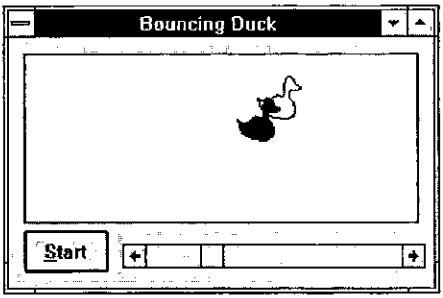
The new object `Clone` is a complete copy of the original namespace, including the functions and variables it contains. (In practice, these are merely pointers, so they do not consume undue amounts of workspace.) Notice that the cloned duck is displayed on top of the original one because it inherits all its attributes, including its size and position.

We can move *Dk* by running the *DRAW* function:

```
      Duck.DRAW
```

We can also identify the clone by making it red:

```
      'Clone' ⎕WS 'FillCol' (255 0 0)
```



To animate the second object we need to create another Timer object:

```
      )CS ##
#.Dk
      'T2' ⎕WC 'Timer' 100 ('Event' 'Timer' 'Box.Clone.DRAW')
                           ('Active' 0)
```

In addition, we need to edit the *START* and *STOP* functions to activate and de-activate the new Timer *T2*:

```
      ∇START[1]'T2' 'Timer' ⎕WS⍠'Active' 1∇
      ∇STOP[1]'T2' 'Timer' ⎕WS⍠'Active' 0∇
```

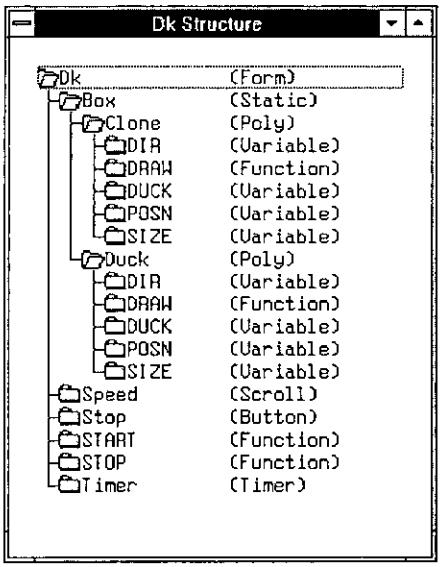
The *Clone* object inherits its *DRAW* function from the original *Dk*. We can however change it to rotate the duck at random intervals:

```
      )CS #.Dk.Box.Clone
#.Dk.Box.Clone

      ∇DRAW[7] →(5≠?5)/0 ⋄ DUCK←⌽DUCK ∇
```

Now start it up:

```
      )CS
#
```

Click *Start* button.



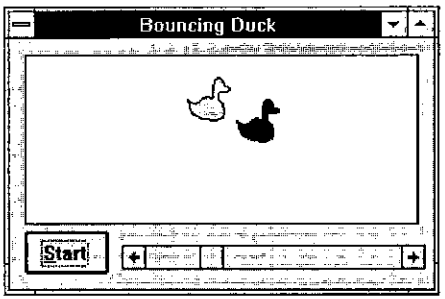
This picture illustrates the structure of the Form (and namespace) *Dk*. It was produced using the MSOUTLIN.VBX (OUTLINE) custom control which is distributed with Visual Basic and which can be accessed directly from Dyalog APL/W Version 7.

Finally, we can save the object we have created:

```
      )OBJECTS
Dk
      )SAVE DUCK
```

The object can then be copied into any active workspace where it will come up ready for use exactly as it was when it was saved:

```
    )COPY DUCK Dk
```



Namespaces

by Eric Lescasse (Uniware)

Introduction

I have always been very surprised to notice how slow APL users are, in general, to start using new features of the language. We have seen that only a rather small percentage of the user base is taking advantage of such simple, powerful and useful tools as the APL*PLUS User Command Processor, error handling facilities (including wonderful utilities like `HANDLERFOR` or `ELXHANDLER`) and nested arrays. On the other hand, one encouraging note is how well the Control Structures have been accepted and adopted by the APL*PLUS community.

But, I have asked myself several times what are the conditions for a new feature to be adopted by APL users **quickly**. I think they are multiple:

- there needs to be a lot of noise and publicity made around the new feature
- people have to understand the new feature and what benefits it can bring to them
- the new feature has to be simple to use (i.e. control structures) and really useful
- users have to be educated in simple terms about the new feature

I think, among these 4 conditions, the second two, and especially the last, are by far the most important and I feel that it is through the lack of them that many APL users are still using only a small part of their favourite software.

One brand new concept recently brought to APL is the one of *namespaces*. This article is aimed to help people discover namespaces (a new Dyalog APL feature), and show them a few simple techniques involving namespaces that can make a huge difference in terms of development ease.

What are Namespaces?

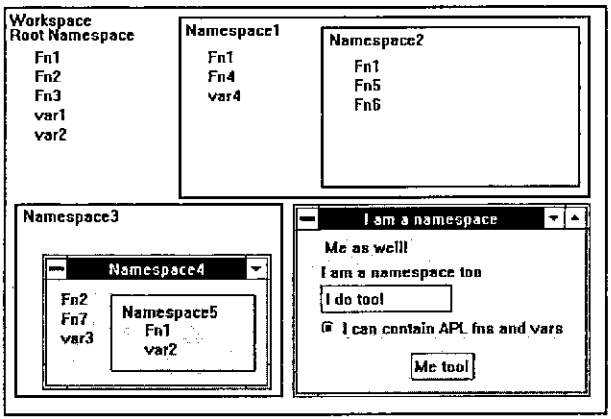
To simplify, namespaces are just sub-workspaces:

> One workspace can contain one or more namespaces, as well as other objects: functions, variables and GUI (Graphical User Interface) objects like forms.
>
> A workspace always has a root namespace (represented by #).
>
> A namespace can itself contain other namespaces, as well as other objects: functions, variables and GUI objects.

> In fact, a GUI object, like a form, is itself a namespace, and every other GUI object (buttons, list boxes, combos, scroll bars, you name them) are all namespaces.

One consequence, which we will be using a lot, is that any GUI object, being a namespace, can itself contain functions and variables. These functions and variables are said to be "encapsulated" within the GUI object. Another consequence is that the workspace and its own namespaces constitute a hierarchy.

Here is a simple diagram better explaining what namespaces are:



As one can see, several objects can have the same name within one workspace, provided that they belong to different namespaces.

Notation

The dot serves as a separator to denote the hierarchy leading to one object. Thus, function `Fn1` within `Namespace5` is represented by the following full name:

```
Namespace3.Namespace4.Namespace5.Fn1
```

while function `Fn1` in `Namespace1` is represented by the following full name:

```
Namespace1.Fn1
```

and `Fn1` within the workspace is represented by the following full name:

```
Fn1
```

All this, assuming that we are positioned within the root namespace of our workspace. Yes, you guessed it: you can decide to position yourself within any of your workspace namespaces, in which case names are relative to the place where you stand.

Assume you have put yourself within `Namespace4`: to access function `Fn1` in `Namespace5` from there, you only need to type:

```
Namespace5.Fn1
```

If you positioned yourself within `Namespace2`, you can type:

`Fn1`	to execute the `Namespace2` `Fn1` function
`##.Fn1`	to execute the `Namespace1` `Fn1` function
`#.Fn1`	to execute the workspace `Fn1` function
`#.Namespace1.Fn1`	to execute the `Namespace1` `Fn1` function

`##` represents the parent namespace.
`#` represents the workspace itself.

This notation is very similar to the DOS notation used to access files within directories, so you will quickly feel very comfortable with it.

All this is very simple, isn't it.

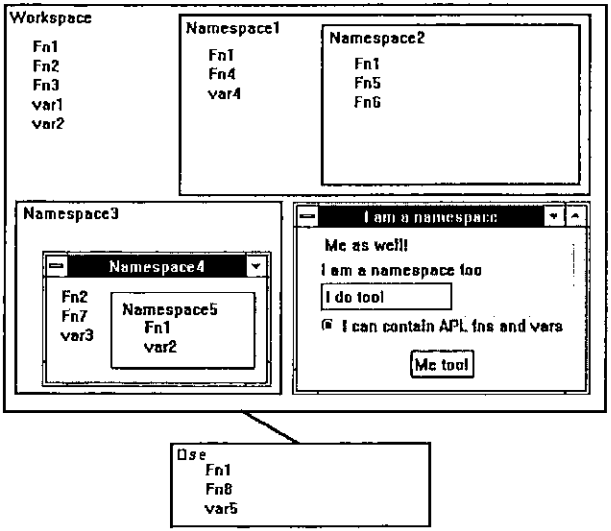
The APL Session Namespace (`⎕se`)

Namespaces are a brilliant idea. But the implementors have had a second brilliant idea which marries well with namespaces.

The APL session itself, which is your development environment, with its own menus, toolbar, statusbar, is itself a Dyalog APL GUI object, called `⎕se`, hence a namespace! This not only means that you can configure it at will (i.e. you can change its menus to your own native language, change the toolbar and the statusbar, etc...), but you can also store objects within it, namely, functions, variables and other namespaces!

`⎕se` works as an external namespace, relative to your workspace. This means that if you load another workspace you can still directly access objects within `⎕se`; this means that if you do `)clear` you can still directly access objects within `⎕se`.

That leads to an immediate idea: store your main and most often used utilities, forms, ... within `⎕se`. They will suddenly become available to you at all times, while you work in Dyalog APL. The previous diagram becomes:



To execute the ⎕se Fn1 function, from your workspace or from any namespace in your workspace, just enter:

```
⎕se.Fn1
```

If you positioned yourself inside ⎕se, you can enter:

```
#.Fn1                    to execute your workspace Fn1 function
#.Namespace1.Fn1         to execute your Namespace1 Fn1 function
```

Positioning Yourself in a Namespace

The)CS command is used to position yourself in any namespace.

The)NS command, used "niladically", tells you in which namespace you currently are. Here are a few examples:

```
      )ns
#                        we start from the root namespace in your workspace

      )cs Namespace1     change from root to Namespace1
#.Namespace1             we are now in Namespace1

      )ns                proof
#.Namespace1
```

```
      )cs Namespace2          change from Namespace1 to Namespace2
#.Namespace1.Namespace2       we are now in Namespace2

      )cs ##                  change to parent namespace
#.Namespace1

      )cs Namespace2          back into Namespace2 again
#.Namespace1.Namespace2

      )cs #.Namespace3.Namespace4    change from Namespace2
                                     to Namespace4
#.Namespace3.Namespace4

      )cs                     niladic )cs brings back to root namespace
#
```

As you see, navigating between namespaces is fairly easy.

Creating New Namespaces

You use the)*NS* command to create namespaces. Examples:

```
      )ns
#                             we start from the root namespace

      )ns Namespace6          create Namespace6
#.Namespace6                  echo

      )ns
#                             pay attention: we are still in the root namespace

      )cs Namespace6          shift from root namespace to new Namespace6
#.Namespace6

      )ns Namespace7          create sub-namespace Namespace7
#.Namespace6.Namespace7

      )ns #.Namespace8        from N'space6, create Namespace8 as child of root
#.Namespace8

      )ns
#.Namespace6
```

```
      )cs
#
```

```
      )ns Namespace9.Namespace10      create 2 new namespaces at once
#.Namespace9.Namespace10
```

Nothing difficult in all this either.

There is another way of creating a new namespace, and, as you guessed, it is to create any new GUI object, using ⎕*wc*. Examples:

```
'F' ⎕wc 'Form'          create namespace F (and also create form F, of course!)
'F.OK' ⎕wc 'Button'     create namespace OK as a child of namespace F
```

Moving Objects from Namespace to Namespace

The dyadic ⎕*ns* system function is used to:

- create new namespaces (if left argument is not an existing namespace)
- move objects from one namespace to another (if right argument is not empty)
- report current namespace (if both arguments are empty)

Its syntax is:

```
R ← D ⎕ns S
```

where:

R	is the full name of the *D* namespace
D	is the destination namespace (created by ⎕*ns* if non existent)
S	is one or more objects to be copied into *D*

Examples:

```
   'Namespace11' ⎕ns ''              create Namespace11
#.Namespace11

   'Namespace11' ⎕ns '#.Namespace1.Fn1' '#.Namespace3.Namespace4.Fn2'
#.Namespace11                        copy 2 functions into Namespace11

   'Namespace12' ⎕ns '#.Fn1' '#.Fn2' create Namespace12 and copy into it
#.Namespace12
   '' ⎕ns ''                         report current namespace name
#
```

```
    )cs Namespace12
#.Namespace12

    )fns
Fn1  Fn2
```

Note that a simple way to copy variables from one namespace to another can be achieved by the following simple method:

```
dest.var ← source.var
```

where `dest` is the destination namespace and `source` the source namespace.

Example:

```
#.var3 ← #.Namespace3.Namespace4.var3
```

Rules for Evaluation

The last important topic to understand, before we move to the major subject of this article, concerns the way the interpreter evaluates expressions when they involve namespaces. Assume we are positioned in the workspace root namespace and want to execute the following expression:

```
R ← Namespace1.Fn1 var1
```

where `var1` is a variable in the root namespace. Dyalog APL does the following:

- evaluates variable `var1` in the root namespace to produce argument for function
- switches to namespace `Namespace1`
- executes function `Fn1` within `Namespace1`, using argument `var1` from root namespace
- switches back to root namespace
- assign variable *R* in root namespace

One very important notion to understand is that, while `Fn1` is executing, in the previous example, it is executing **WITHIN** `Namespace1`. That means, that, if it needed to use variable `var2` from the root namespace, it should refer to it as `#.var2` and **NOT** just as `var2`.

This is essential to understanding how to work with namespaces.

What Else Can You Do with Namespaces?

Most system functions and the ⍎ primitive are namespace aware. This means you can do such things as:

```
      )ns
#.Namespace1
      #.Namespace3.Namespace4.⎕nl 2
var3
      '#' ⍎ 'Fn1'
      ⎕se.⎕ed 'Deb'
      Namespace2.var1←⍳10
```

Benefits from Using Namespaces

After having read these few pages introducing namespaces, you have understood most namespaces concepts, but may be still asking yourself what are your benefits of using such things?

In fact they are numerous. Namespaces can help you:

1. **Store your utilities in the ⎕se namespace and have them handy at any time.** This is probably one of the first things you will want to do. It brings similar advantages as APL*PLUS User Command Processor, with better performance. I could no longer work in APL without either one of these tools.

2. **Avoid name conflicts.** Example: if you kept your workspace root namespace empty and stored all your functions and variables in namespaces, you would never worry about name conflicts when copying objects in your workspace.

3. **Clean up your workspace and group objects logically.** And do not clutter any namespace with hundreds of functions and variables.

4. **Avoid local functions.** They can be replaced by identical functions called from namespaces.

5. **Have several objects with the same name within one workspace.** Namespace will let you do such things as emulate the following SQL syntax:

   ```
   select employee.name,dept.name,employee.salary
   from Employee where employee.id eq dept.id
   ```

   where *name* and *id* would be variables residing in the *employee* and *dept* namespaces! This would have been impossible without namespaces or without using quotes.

6. *Encapsulate* **all callback functions in GUI objects.** This really is the major advantage of namespaces, in my opinion. And I will conclude this article by showing a namespace technique which greatly simplifies the programming of GUI objects and of Windows applications with Dyalog APL/W.

7. **Exchange information between GUI objects without using global variables or** *'data'* **property.** It very often occurs when programming GUI objects that a callback routine needs a piece of information created by another callback routine. Unfortunately, the nature of event driven programming makes all callback functions independent of each other and moreover global objects in the workspace. Therefore the only way to pass information from one to another is to use global variables. But as you all know, this is not good APL programming practice and should be avoided. Both APL*PLUS and Dyalog APL provide a *'data'* property for almost all of their objects, within which you can store any amount of information. But there is only one data property per object and, even if you store nested arrays in it, it is not as convenient to use as just storing variables in the object namespace.

There are certainly several other advantages of using namespaces which I have not yet discovered or used, but these ones are already more than enough to make me wish that namespaces one day become standard in any APL. As well as control structures!

A Useful and Simple Namespace Technique to Program GUI Objects

GUI programming

The previous pages were necessary for Dyalog APL/W or namespace novices to understand the following part of this article. We have seen that each GUI object has its own namespace. The whole job of GUI programming is to write small APL routines to react to events occurring on GUI objects.

The work is quite simple:

- create and design your form, installing objects in it and giving them properties
- identify all user events that can occur on these objects
- write one APL callback function for each (object, event) couple

For example, if one wants to react to a user click on the *OK* button of form *F*, one needs to associate an APL function to the *'Select'* event on the *'F.OK'* button, with the following expression:

```
'F.OK' ⎕WS 'Event' 'Select' 'F_OK_Click'
```

where *F_OK_Click* is the name of the APL callback function.

When the user clicks on the *OK* button, the APL interpreter instantaneously executes *F_OK_Click* because it knows, since our ⎕*ws* expression, that we have associated this name to the *'Select'* (alias click) event on the button.

Problems with GUI programming

The problem is that the GUI programmer quickly discovers that his workspace soon gets cluttered with hundreds of callback functions. This is because one dialog box can easily contain 20 objects and each of those can easily get several different events. Remember that you have to write one APL function for each (object, event) couple.

A second problem is that your form and all of its children objects will not run unless all of its callback functions and (global!) variables are there with it in the same workspace environment.

Imagine one day wanting to copy this form object into a new workspace and then discovering that you have also to copy a hundred callback functions to choose among a thousand functions residing in the source workspace. What a headache!

A solution to the problems of GUI programming

The answer, of course, comes from namespaces. Here are the rules:

- It seems natural to store a callback function in the object (namespace) to which it refers
- It seems natural to name this callback function before the event it handles

For example, an APL callback function handling the Select event on a button, will be named *Select* and will be stored in the button namespace.

We would then really use a lot of the power of namespaces: we will have several functions in our workspace bearing the same name (*Select* for example): only namespaces allow that!

We will also encapsulate all callbacks within our form and its child objects, making our form self-functioning. We can then copy it as a stand-alone object in another workspace and start using it immediately: it will run perfectly, because the hundred callback functions will have been carried with it when copied into the new workspace.

That's great: it means we can write self-functioning forms!

We have spent years writing utility functions, and creating our own library of powerful utilities to develop lightning fast with APL: it means we can now start writing utility forms or better, what I will call **PARTS**, i.e. GUI objects that we can easily plug into any new application we write.

Imagine: almost all Windows applications have a `File` menu and the `File` menu structure is fairly standard: even the accelerator keys it uses tend to be standard across applications. Well, we can use namespaces and write a `File` menu **PART** which will encapsulate parameterized callbacks.

Then we can plug it in any new Windows application we write, just as is, or clone it with `⎕OR` and change it by exception. Half an hour (or may be an hour) saved each time.

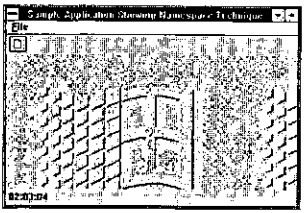
A namespace technique that will save you a lot of time and effort

With Dyalog APL/W, you generally start developing your forms with the `WDESIGN` workspace which is a nice resource editor. However, once you have designed your form and set most of its properties and children's properties, you generally want to create a `_MAKE` function representing your form and start working with this `_MAKE` function, forgetting about `WDESIGN`.

If you install callbacks within your form objects, they will be lost every time you run your `_MAKE` function, since the `_MAKE` destroys your form when recreating it with `⎕WC`.

Soon comes the idea of installing the callback functions within the `_MAKE` function so that they are re-installed properly within their relevant objects' namespace every time `_MAKE` is rerun. *The namespace technique I have developed is doing exactly that.* We have worked with it for a couple of months, developed a lot of forms using it, and it has proved to be a real time and effort saver, for programming GUI objects.

How does it work in practice? It is best described by an example.



Here is a sample application: It is a simple MDI application which shows several objects:

- a form
- a menu bar with a File menu
- a toolbar with one icon
- a status bar with one field displaying the current time
- an MDI client with a bitmap image

This application handles the following events:

Object	Event	Action to perform
F	*Close*	Kill the timer
F	*KeyPress*	If Esc, terminate the application
F.TM	*Timer*	Display the new time
F.MB.FILE.NEW	*Select*	Start child window application (*#.Transaction*)
F.MB.FILE.QUIT	*Select*	Terminate application
F.TB.B1	*Select*	Same as *Select* on *F.MB.FILE.NEW*

The listing below shows the stand-alone APL function that can recreate the whole application. The top part of the function has been more or less created by *WDESIGN* and contains the instructions that can recreate the form and its child objects. The bottom part of the function contains all the callbacks relative to this form. Each of them starts with a line following a ∇ symbol.

The key to our technique is the use of the *storefns* routine, called from ⎕*se*, on line 30. This utility analyses the code of its calling function (*Main* here), recreates the callback functions and installs them in the relevant objects. It will be explained in more detail later on.

```
     ∇ Main
[1]
[2]   ⍝ Create main form
[3]    'F'⎕WC'Form' 'Sample Application Showing Namespace Technique'
[4]    'F'⎕WS('bcol' 255 255 255)('Coord' 'Pixel')
[5]    'F'⎕WS('Accelerator'(27 0))('3D' 'Default')
[6]
[7]   ⍝ Create toolbar
[8]    'NEW'⎕WC'Bitmap' 'c:\wdyalog\ele\bmp\new'
[9]    'F.TB'⎕WC'ToolBar'
[10]   'F.TB.B1'⎕WC'Button' ''(2 4)(22 24)('Picture' 'NEW')
[11]
[12]  ⍝ Create menu bar
[13]   'F.MB'⎕WC'MenuBar'
[14]   'F.MB.FILE'⎕WC'Menu' '&File'
[15]   'F.MB.FILE.NEW'⎕WC'MenuItem' '&New'
[16]   'F.MB.FILE.QUIT'⎕WC'MenuItem' '&Quit'
[17]
[18]  ⍝ Create timer
[19]   'F.TM'⎕WC'Timer'('interval' 1000)
[20]
[21]  ⍝ Create status bar
[22]   'F.SB'⎕WC'StatusBar'
[23]   'F.SB.F1'⎕WC'StatusField'('size'0 60)
[24]
[25]  ⍝ Create MDI Client and menu
```

```
[26]   'BMP1'⎕WC'Bitmap' 'C:\WINDOWS\WINLOGO.BMP'
[27]   'F.MDI'⎕WC'MDIClient'('3D' 'Default')('Picture' 'BMP1' 1)
[28]   'F.MB'⎕WS'MDIMenu' 'FEN'
[29]
[30]   ⎕SE.storefns          ⍝ store callbacks in form objects
[31]   ⎕DQ'F'
[32]   →0
[33]
[34]  ⍝ +++ Callbacks section +++
[35]
[36]   ∇
[37]   Close                                        ⍝ F
[38]   ⎕EX'TM'
[39]
[40]   ∇
[41]   KeyPress                                     ⍝ F
[42]   ⎕EX'TM'
[43]   #.⎕EX'F'
[44]
[45]   ∇
[46]   A←FormatCurrentTime;subroutine               ⍝ F.TM
[47]   A←,'G<99:99:99>'⎕FMT 100↓⎕TS[⎕IO+3 4 5]
[48]
[49]   ∇
[50]   Timer                                        ⍝ F.TM
[51]   '#.F.SB.F1'⎕WS'text'FormatCurrentTime
[52]
[53]   ∇
[54]   Select                                       ⍝ F.MB.FILE.NEW
[55]   #.Transaction
[56]
[57]   ∇
[58]   Select                                       ⍝ F.MB.FILE.QUIT
[59]   #.⎕EX'F'
[60]
[61]   ∇
[62]   Select                                       ⍝ F.TB.B1
[63]   #.F.MB.FILE.NEW.Select
     ∇
```

You can notice that:

- all callbacks described in the above table are defined within the *Main* function
- they are separated by a ∇ symbol
- all callbacks are named before the events they represent
- on line 30 utility *storefns* is called from ⎕*SE*
- *FormatCurrentTime* is a subroutine: it is not an event.
- the word *'subroutine'* must be localized in *FormatCurrentTime* to distinguish it from a callback
- callback *Select* on button *F.TB.B1* calls another callback *F.MB.FILE.NEW.Select*
- callback *Timer* uses a subroutine (*FormatCurrentTime*)
- it has not been necessary to use such expressions as (*'Event' 'Select' 'F.TB.B1.Select'*)

Let's now analyse in more detail, what the *storefns* is doing.

Analysis of *storefns* utility

This function looks at the code of the function that calls it, i.e. the *Main* function here. It searches lines starting with the ∇ symbol and extracts pieces of code separated by ∇s and fixes them as functions in the root namespace.

It then analyses the comment on line 0 of these functions (another nice feature of Dyalog APL/W that we are exploiting here) which is supposed to contain the name of the object in which the callback is to be stored.

It then activates the events by issuing the *'object' ⎕WS 'Event' 'event' 'object.event'* instruction for us: this means we do not need to add all these instructions in the top part of our *Main* function! Note that it does not activate events for functions that have the word *subroutine* localised.

It then copies the callback functions in the relevant objects and finishes by erasing the callback functions from the root namespace. In case you want to use this technique, here is the code of *storefns*:

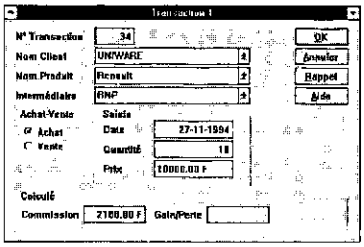
```
      ⎕se.⎕vr'storefns'
      ∇ storefns;A;B;C;D;E;G;I;L;⎕ML;⎕IO
[1]    ⍝ Store local functions in their corresponding objects namespaces
[2]    ⍝ Can only be executed in main workspace namespace.
[3]    ⍝ Copyright (c) 1994 Eric Lescasse 21oct94
[4]    ⎕IO←×⎕ML←2
[5]    A←#.⎕CR 2⊃⎕SI                    ⍝ name of calling function
[6]    A←(∊1 0)+¨(A[;2]='∇')⊂[1]A       ⍝ event handlers code
[7]    I←1                              ⍝ loop index
[8]    →L←((⍴A)⍴⍳),0                    ⍝ loop labels
[9]   a:B←I⊃A                           ⍝ function ⎕cr
[10]   C←B[1;]                          ⍝ function line 0
[11]   D←((C⍳'⍝')↑C)~' '                ⍝ object name
[12]   →(1='⍟'∊D)↓b                     ⍝ is it a dynamic name?
[13]   D←'#'⍎D~'⍟'                      ⍝ evaluate dynamic name
[14]  b:E←#.⎕FX B                       ⍝ create function
[15]   G←('#.',D)⎕NS'#.',E              ⍝ copy function in object namespace
[16]   →(1∊';subroutine'⍷C)⍴c           ⍝ do not activate event if subroutine
[17]   ('#.',D)⎕WS'Event'E('#.',D,'.',E) ⍝ activate event for this object
[18]  c:#.⎕EX E                         ⍝ erase function
[19]   →L[I←I+1]                        ⍝ loop back
      ∇
```

Raising the Difficulty

Our application is an MDI application and as such can create child forms. One of the problems of MDI applications is that you have to keep track of child forms' names since the user can generally create multiple copies of them (MDI would not mean Multiple Document Interface otherwise, would it?). Therefore it is your

application that needs to dynamically generate these child form names as the user creates new ones. How does this fit with the *storefns* utility? We cannot pre-allocate dynamic object names on the line 0 comment of callback functions. Well, this is solved by prefixing the object names with the execute symbol, as our application *Transaction* function shows.

But first let's look at the Transaction child document:



The function that creates the above child document is the following (where we have not reproduced all lines in order to save space in this article):

```
      ∇ Transaction;childname
[1]
[2]     childname←'F.MDI.TR',▼D          ⍝ dynamic name
[3]
[4]     childname ⎕WC'SubForm'
[5]     childname ⎕WS('BCOL'(192 192 192))('CAPTION'('Transaction ',▼D))
[6]     ... ⍝ change more properties
[7]
[8]     (childname,'.BOK')⎕WC'BUTTON'('CAPTION' '&OK')('POSN' ...
[9]     (childname,'.BAnn')⎕WC'BUTTON'('CAPTION' '&Annuler')('POSN' ...
[10]    ... ⍝ create more objects
[11]
[12]    ⎕NQ childname'MDIActivate'
[13]
[14]    ⎕SE.storefns
[15]    →0
[16]
[17]    ∇
[18]    Select                                  ⍝ ⍕childname,'.BAnn'
[19]    ⎕EX'##'⎕NS''
[20]
[21]    ∇
[22]    R←CheckField A;B;I;subroutine           ⍝ ⍕childname
[23]    ... ⍝ is only a callback subroutine
[24]
[25]    ∇
[26]    Select;A;B;C;D;E;F;G;H;I;bad             ⍝ ⍕childname,'.BOK'
[27]    A←##.CheckField'CCli'
[28]    B←##.CheckField'CPro'
[29]    ...
      ∇
```

We just need to add an ⍎ symbol before the dynamic expression that represents the name of our child document: lines 12 and 13 of utility `storefns` handle object names starting with the ⍎ symbol (such as: `⍎childname,'.BOK'`).

Benefits of the Namespace Technique Explained Above

The benefits are many:

- you can write self-functioning forms with complete encapsulation
- your whole form can be recreated by just one function (`Main` or `Transaction` here)
- you do not risk losing any callback encapsulated within objects
- you do not need issuing the `'object' ⎕ws 'Event' 'event' 'callback'` sentences
- you can visualise several or all callbacks at once
- you can most easily copy and paste APL code between callbacks (often needed in GUI programming)
- you do not clutter your workspace with numerous callback functions

The whole MDI application shown above (simplified from a real case) is contained in the 2 following functions:

```
    )fns
Main Transaction
```

Is Encapsulation Possible with APL*PLUS III?

For those of you who are using APL*PLUS III, it IS possible to encapsulate all your callback routines within your main application form, although in a different manner.

Here is a possible technique.

- Create an additional button called *code* on your form
- Give it a *'where'* property of `¯100 ¯100` so that it is located outside the form
- Store all your callbacks and utilities in the *'data'* property of this button. Something like:

```
'form.code' ⎕wi 'data' (⎕vr¨'callback1' ... 'lastcallback')
```

- Then, define the *Open* callback of your form as:

```
'form' ⎕wi 'onOpen' 'handlers←⎕def¨''form.code''⎕wi''data'' ⋄ form_Open'
```

- And define the *Close* handler of your form so that it erases all handlers:

```
'form' ⎕wi 'onClose' 'form_Close ⋄ ⎕erase¨handlers'
```

Conclusion

Namespaces is a very interesting extension to APL. It is the path to a real object oriented APL which will exist one day. And for now, it is a rather simple to use and very powerful feature. Some further refinements are needed, some nice extensions of the namespace concepts are possible as well, but even as they have been done in the first place, they are more than useful: they can make GUI programming with APL much nicer and much easier and they really allow us to write PARTS and, if used well, bring us re-usability and encapsulation, and are close to bringing us polymorphism as well.

Namespaces: a Way to a Well Organized World or Just Another Means to Multiply your Chaos

by Kimmo Kekäläinen
email:Kimmo.Kekalainen@metsa.fi

My waiting was finally rewarded by Dyalog version 7 last summer. About namespaces I had heard John Scholes' introduction in Swansea. Although 7 was full of new fancies — toolbar, hints & tips, VBX, timer, MDI, metafile and so on — they all were just new features — welcome, useful indeed, but mainly to be classified as add-ons to GUI-functionality, even the Grid. Namespaces were something different. When exploring 7, I saved it till last. As I could have concluded from John's demo, this new concept seemed to provide thus far unreached possibilities to organize your working environment in a new, more controllable and productive manner.

Extensions to the Interpreter

Who had not a subset of little utils like *OVER, DEB, ROWFIND, DLTB*? You need them time after time, in every workspace. To do anything with APL, you soon miss some of them, you copy them. Why not have them around like primitives or *⎕FUNCTIONS*? Now namespaces provide you with this possibility, at least in a logically analogous manner. You don't happen to have *⎕ROWFIND*. However, if you had, *⎕ROWFIND* or *⎕SE.ROWFIND*, what's the difference to a calling program?

But you can go much further. If you are an application developer using APL, you definitely have a subset of tools to do common tasks that are needed, whatever application area you are involved in. You probably use some dialog boxes to ask questions, give WAIT- and OK-messages, print, show reports to the user, maybe graphics, file handling etc. The problem with these in every task is not only to copy those, but also the varying range of sub-functions they are calling. And you also develop them, whether to correct bugs or do improvements. But where on earth was the last version.

Then, after a programming period you take *)FNS* to see what you have and are faced with the problem of extracting the problem-specific functions from the common namejunk of *OVER*s and *QUERYSTRING*s. Then you think again that

shouldn't they have put these into the interpreter, of course deliberately ignoring the fact that if they really would have, your EXE would finally end up exceeding the size of your RAM. Well, *Namespaces* allow you to do this yourself, without overloading the exe-file, but logically at the interpreter-level. Here is the way to make your own enhancements to the environment. Having these in mind I started.

Step into Spaces

First I put my *ROWFIND*s directly under the ⎕*SE*. That's a proper place, at least for idiom-like things. They are small, mostly oneliners, old, safe and robust, familiar like primitives. Good to have them around, always. The same counts for my set of general GUI functions named according to task like *QUERYSTRING*, *CHOICEFROM*, *POP_UP_MENU*, *OK_MSG*, *WAIT_MSG* and so on.

When you proceed with hacking your toolset, you probably face the problem that the complexity of your common task-related tools grows. They call sub-functions, both idiom-like and task-related. If you want them to be always available, the answer is to create your own namespace and save it directly under ⎕*SE*. Basically, this is a very straightforward clear concept. However, if you have a GUI-tool to show a report on a scrollable form on screen, you probably want to provide the possibility within that tool to print it, guide it to the clipboard or Excel, maybe give your user a chance to change fonts or colours on screen and so on. But to provide parallel things like these you may need to call functions from other namespaces under ⎕*SE*, or in namespaces under them. Still, this showing a report on screen is a very common need. Most of my application programming somehow relates to reporting. It must be handled in a unique manner, with one general tool. So I did it ...

... and went on. In a few days I had about 15 different sized namespaces under my ⎕*SE*. There were my goodies, available in a clear workspace as I always had wanted. My goal was gained. Hadn't I earned I feeling of relief and satisfaction — perhaps a cold beer! *[Have two – Ed]*

Still this made me wonder. Most of my ⎕*SE* was OK, definitely, idiom-like stuff and so on. What made me nervous was the growing number of references between functions in different namespaces. How ever am I going to maintain this in the future? The reason in the start was sound and clear — they were already there. Another way would have been to copy and commit a sin of multiplication; wasn't that to be something of the past age, before namespaces? Dyalog allows you to go as deep as you wish in namespace structure. If you really do, good luck — *and prepare a map*. I began to feel a bit, if not lost, at least confused. What

if I want to use some of these tools independently of current ⎕SE? Am I now somehow stuck with my profile? I had happily got rid of the namejunk in my application workspace, at the price of namejunk in my ⎕SE.

A consistent naming convention is a thing I often preach to others as a free and powerful way to give readability to code and to provide self-carrying information to an application structure. This was something I forgot to plan in my anxious start. Some namespace-names under ⎕SE were UPPERCASE, some were lowercase, some started with Uppercase and so on. Had I adopted a disciplined standard it would have made functions calling these easier to read and extract in application code. OK, go and change the names of the namespaces. Yes, but then go and change all the references to functions in them that lie there waiting in those 15 namespaces or under them. Then there was this trouble of finding those AP- or DLL-functions that you forgot to kill or localize. There they stayed lost in some unfound namespace and efficiently prevented you from saving your latest work.

While worrying about these I made a little DIR to find out that my original ⎕SE-file had grown from 100 Kilos to half a megabyte (part of that goes to a bug in the Dyalog saving structure, but still ...). What I did? I created a workspace copy of ⎕SE- functions and of every single ⎕SE-namespace under it. And stopped.

Conclusions

This process was worthwhile. I made a good inventory of what I have. My attitude towards namespaces is totally positive. They are not only a promise, I'm convinced that they really provide you with a way of handling your utils better. Mistakes were mine. I think I learned a lot. Benefits don't come automatically. So, when I next start organizing my little Dyalog World, I will carefully think and judge my existing toolset in the following terms:

- What are the general routines you really are going to need as functions directly under ⎕SE? What really are the general routines you are going to need as namespaces directly under ⎕SE?
- Are you going to allow a namespace-located function to call a function living directly under ⎕SE, and more important, do you allow it to call to functions in another namespace?
- How many namespaces under a single ⎕SE will you be supporting from the point of view of memory load and maintainability?

- How many session profile files do you think will be needed to provide a controllable and productive environment, starting from the fact that if you are an application programmer, the minimum is two — one to develop applications, one to run them (you don't need *WSDOC* to do that)?

- What's the proper place to save and maintain the "source" code for utility functions that end up in *⎕SE.namespace*? Will the traditional WS still the be the most flexible alternative for that, after all? It would allow the independence of the ns-concept (totally if ns-references are not allowed) in the case of possible use in environments where namespaces are not supported. Still they could be easily called and hooked into *⎕SE* by copying in at the start and easily stripped off when not needed. I'll probably never be able to expunge a third of my interpreter as I in certain situations wish, but should at least be able to do so for my session enhancements.

  Currently I still have more questions than answers, sorry!

When you keep developing applications with APL for several years, you will get accustomed to doing things in a certain way, adopting habits, preferring certain techniques. They give a kind of "stamp" to your work. This is good in the sense that it brings consistency to your work, bad in the way that you may get stuck with them and miss possibilities that others have discovered, which might help you do things better.

Every now and then, it is good to stop and do some evaluation; look around. If you haven't done this lately, namespaces is a good place for that, in fact a must. **They are a stop point**, in case you don't want to miss the point and lose the benefits. They can help you to organise your application development environment to be a productive and disciplined world, or just stay as an alternative means to multiply your chaos.

Coast-to-Coast Revisited

by Adrian Smith

Please refer to Vector 10.1, page 97 for the background to this note. One of the ideas that I introduced was of managing the Dyalog 'data' property using a pair of functions `set_data` and `get_data` to store named variables in any GUI object. I commented at the time that I felt this was a "huge step forward in design" and fitted in well with the APL style we already knew. I started with the assumption that I ought to be able to remove all this code, as namespaces were designed to do precisely the job I had coded around.

If you look first at `init_icons` (page 101) or `init_game` (page 105) you can see that *as long as you know what your object is called,* namespaces clean things up nicely:

```
[8]  'ttmk'⎕WC'BITMAP' ''bmp cm
[9]  ttmk.rtn ttmk.fn ttmk.map←2 'make_term 3' map

[24] BD.ToPlay←'WN'   ⍝ Either player may start!
```

The fun begins when you have the same callback on many objects, so the object name comes in as an argument. Look at `rotate_tile` on page 104:

```
[4] rtn fn←bmp get_data 'rtn' 'fn'   ⍝ Old version

[4] rtn fn←⊃bmp,'.rtn ',bmp,'.fn'    ⍝ New version
```

This is nasty enough, but putting the data back again gets quite horrible:

```
[8] bmp set_data ('rtn' rtn)('map' map)  ⍝ Old version

[8] ⊃bmp,'.rtn←rtn ⋄ ',bmp,'.map←map'    ⍝ New version
```

I know that Dyadic are working on `⎕CS`, to let us shift namespace under program control, so I suppose I will soon be able to write:

```
[8] ⎕CS bmp ⋄ rtn←##.rtn ⋄ map←##.map ⋄ ⎕CS '##'  ⍝ Switch ↑↓
```

... which is better, but leaves me in danger of forgetting to switch back at the end. I think what I really want is something that looks like:

```
[8] bmp ⎕CS { rtn←##.rtn ⋄ map←##.map }  ⍝ Execute in namespace
```

... but I don't hold out much hope of seeing it!

TECHNICAL SECTION

This section of VECTOR is aimed principally at those of our readers who already know APL. It will contain items to interest people with differing degrees of fluency in APL.

Contents

Puzzle Corner: The Age of the Vicar

from Alan Sykes

The Problem

A vicar says to his curate:

> "I have three parishioners whose ages multiply together to equal 2450 and whose ages sum to twice your age — what are the ages of the parishioners?"

The curate thinks for a while and then tells the vicar that he does not have enough information.

> "Quite right" replies the vicar "but if I tell you an extra piece of information you will have enough."

The extra piece of information is that the vicar is older than any of his parishioners.

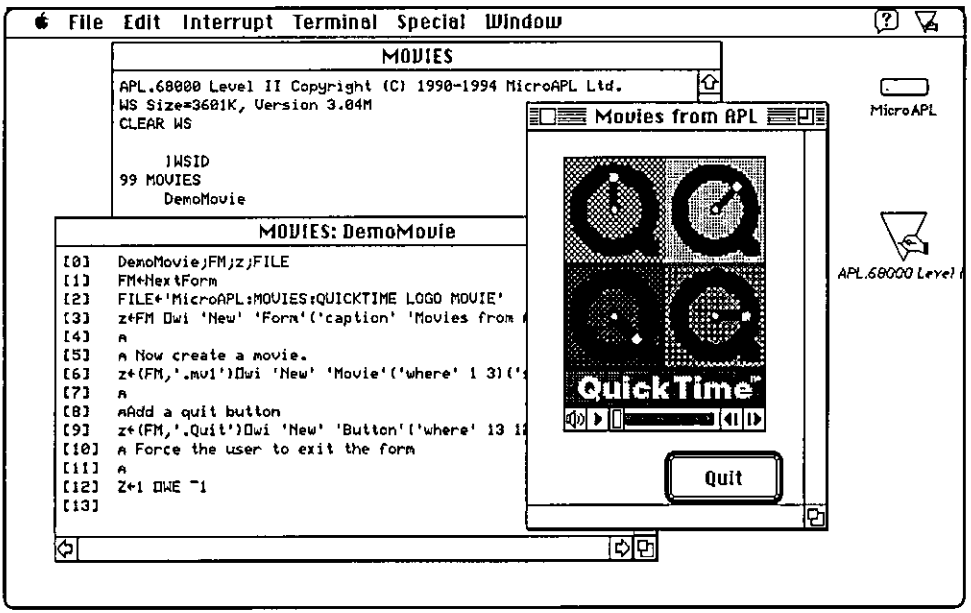
Your task is to find the age of the vicar!

The Solution

... will be in Vector 11.4 along with a full explanation!

APL for the Apple Macintosh

Since 1985 MicroAPL has pioneered the use of APL in graphical environments. Our latest version of APL.68000 Level II for the Apple Macintosh is now available, offering dramatically enhanced GUI programming facilities.



APL.68000 Level II for the Macintosh includes the folowing features:

- Runs on all models of Apple Macintosh
- Native version for the Power Macintosh
- Conforms closely to the APL2 specification
- Uses the standard Mac interface
- Object-based GUI programming via ⎕WI
- Full event handling via APL callbacks
- Free runtime version with application packager

Accelerated for Power Macintosh

MICRO APL

MicroAPL Limited	South Bank Technopark, 90 London Road, London, SE1 6LN, UK
Voice:	0171 922 8866
Fax:	0171 928 1006
Applelink:	microapl
Internet:	microapl@microapl.demon.co.uk

The Axiom Waltz (The Information Wall) or When 1+1 make 0

by Gérard A. Langlet APL-CAM Journal, Vol. 15, No. 4, 16 October 1993, pp 601-609.
Translated by Diane Whitehouse and Gill Smith

Editor's note: I am very grateful to the two translators. Gérard's French is very difficult to translate, as it is packed tight with various kinds of word play, not to mention obscure French proverbs and sayings. Such great brilliance sometimes makes it harder to follow the train of thought; with Gérard there are probably three simultaneous trains of thought anyway. I would be glad to hear whether readers enjoy reading our best efforts at rendering Langlet into English. If so the latest issue of Les Nouvelles d'APL has a further two articles!

Summary or Introduction

One of the most famous axioms in the history of mathematics is undoubtedly Euclid's: *"Through a point situated outside a line, one can only draw a single line parallel to that line."*

The abandonment of this dogma, fixed in the mind by centuries of teaching Euclidean geometry based on the aforesaid axiom, happened many centuries after Euclid, with Riemann and Lobatchevski, and led to a tremendously rich new mathematics (from which relativity ensued). Yet at the same time the whole world is convinced that *two and two make four...*

On a little reflection one realises that it is now possible to break down the **Information Wall**, like the Berlin Wall, without a great effort. This is largely thanks to APL with its wonderful properties, such as a pure mathematical notation, and the possibilities this language offers for experimentation in new ideas using any kind of modern microcomputer and then expressing these ideas concisely. In mathematical terms, this comes down to overhauling fundamental axioms, to exploring new routes — routes neglected not only by pure mathematical theory but which are, above all, missing *ipso facto* from models in physics and biology. It is unfortunate that these have, up till now, relied mainly on considerations of mass and energy, while the opportunity of research into **Information** remains wide open; yet it contains the seed of fruitful discoveries.

Physicists or biologists always gather Information (and very little energy) in their experiments. The computer, now a familiar tool to both, can be stretched to an

unlimited extent to handle whatever Information one gives it. The DNA which programs us all is more Information than matter. So, let us destroy the Wall and try to revise the axioms. In principle, when one travels up the course of a river to find its source (whether that river is the Nile, the Seine, the Amazon, or the Mississippi), the widest branch at each confluence is not always the longest: one must explore every branch of every fork one after the other to get to know it all (which can't be done all at once, unless you can use a satellite).

Information's Natural Processes

We know very little about the processes that lead a man to think, nor of those that govern the development of a human being starting from a chemical program (a metre and a half long if one unravels the wonderful double helix of the DNA sequence — the *helical* spring of our being wound up in the nucleus of our cells and invisible to the naked eye, with a "listing" of 23 pairs of sub-programs called chromosomes). When a system is self-organizing, the appearance of observable order would go against the laws of thermodynamics, for entropy seems to decrease in such a process; now entropy is, by definition, tied to energy: it is itself defined as its "quality". On the other hand, if one considers the information in a self-organized system, one can postulate that this information must be preserved without any deterioration in quality, for example from one generation to the next, or during **metamorphoses**: the *caterpillar* contains all the information of the chrysalis which itself holds all the information of the *butterfly*.)

The **Principle of Conservation of Information** can replace, if not be identified with, that of conservation of momentum in mechanics, a principle that is well known to billiards players. But information is only expressed in bits or in pixels. One cannot either average it or smooth it without degrading its content. One can never increase the quantity of information contained in a system by either interpolation or extrapolation. Only reversible processes conserve all the information contained in a given system: these exchanges should produce a constant volume of information to remain optimal. If a system grows in size and adds to its information, it is because it has captured information from somewhere outside itself, to the detriment of another system that it has destroyed (naughty!); or, it has copied, aped, or cloned the information from another system without necessarily destroying it (dodgy but kind).

In mathematical terms, beginning with linear algebra for instance, we are led to seek plausible models for a simple and above all correct formulation (so as to make the least number of mistakes, we will put forward the smallest possible number of axioms.

Matrix Mathematics

An inverse matrix (with no numeric mistakes) is an ideal operator for transforming information. Indeed — though in theory only — this matrix and its inverse offer us the possibility of transforming a problem's data into results, but also of recovering the data from the results. In practice, it is impossible, except in particular very rare, if not trivial, cases, given any matrix M, to calculate the inverse matrix M^{-1} such that the inverse matrix of the latter $(M^{-1})^{-1}$ is strictly identical to M. Readers with a knowledge of APL can try to disprove this more easily than others, for in APL any comparison of real magnitudes takes place while taking into account a relative *epsilon* called the **comparison tolerance**.

However, to model a **reversible** process, we need a self-inverse matrix, with M by definition identical to M^{-1}. Hence, it seems useless to refine the numeric algorithms of matrix inversion. It would seem much wiser to use pure reasoning to investigate the required self-inverse matrices. To do so, we shall start small and then explore the trivial example of a one-row, one-column matrix necessarily containing 1 (in APL `1 1⍴1`), and then two-by-two matrices. Afterwards, let us extend our reasoning to a greater number of rows, going, if possible, right up to infinity to see whether by chance we might have forgotten some fundamental options, just as by failing to explore the pathways corresponding to branches of reasoning and perhaps to the toppling of parity in the positing of the initial axioms, the little black or white pebbles of Perrault's Little Tom Thumb, lost in the combinatorial forests of a Game of GO, from "go to" to GOTO, the great Japanese theoretician of decision-making.

Self-inversion of Matrices of Rank 2

For every matrix of rank 2, we can write a system of four equations (1) (2) (3) (4). This will allow us to search for the possible values for the four terms a b c d if the matrix is self-inverse. (The period symbol "." on its own shows the generalized matrix product expressed by the inner product in APL in the form `+.×` for the numerical arguments.)

```
ab . ab -> a²+bc  ab+bd <--> 1 0 thus: a² + bc = 1(1) ab + bd = 0 (2)
cd   cd    ac+cd  bc+d²      0 1       ac+cd = 0  (3)   bc+d² = 1 (4)

Equations (1) and (4) imply: a² - d²  = 0                          (5)
then a² = d²  therefore either a = d or a = -d

A determinant equal to 1 is expressed by: ad-bc = 1                (6)
Equation (2) is resolved by either: b = 0 or with a = -d
Equation (3) is resolved by either: c = 0 or with a = -d
The solution a = d is only possible if b = 0 and c = 0.
Thus, according to (6) ad equals 1, thus a and d are both equal to 1 or -1.
```

Two matrices are therefore **self-inverse**. They are:

```
1 0  and  -1  0
0 1        0 -1
```

```
Thus a = -d, then equation (6) becomes: -a2 - bc = 1                    (7)
```

```
Adding (7) to (1) would force the equation to be resolved as:
```

$$0 = 2 \qquad (8)$$

This resolution is especially sensitive in the context of axioms that are particularly well established. This kind of result constitutes an **Information Wall** that is generally impossible to climb. Unless ...

A Change in Algebra or a Very Natural Algebra?

When we announce that *"two and two make four"*, we presuppose that *and* means *plus* and that *"one and one make"* is already *"two."* In reality, if one human being and another human being make two humans, a man and a woman form a couple, and make a child. Counting purely numerically is therefore an abstraction of reality; or, more precisely, it is an abstraction of our judgement of reality (which occurs with the aid of our senses and our "understander", the brain). If Huygens had already announced, in his *Treatise on Light* (1671) — and in French — *"We perceive ONLY the differences,"* he had already understood the puzzle's fundamental axiom. However, Huygens had neither the APL nor a suitable computer to make any kind of progress except with the help of continuous functions.

All our biological receptors are discrete, and no measurements of anything can be made other than through sampling (and this assertion was already true before the time of Poincaré and Shannon) and we can never analyse nor model the universe itself: we can ONLY try to interpret, understand, and possibly calculate, our perceptions of the so-called universe.

The information we perceive can always be expressed by 0 and 1. We have found no other effective means of handling information other than as a series of 0s and 1s. We can rightly enquire, "Why?"

Both conceptually and naturally, all information can be reduced to 0s and 1s. Why therefore do we try, on a day-by-day basis, to shape that information into other forms which are much more difficult to digest and to manage by computers which, by definition, only know how to handle 0s and 1s? The simple

inversion of a more or less large matrix executes thousands of millions, even billions, of useless and perfectly short-circuitable, conversions between various series of bits with a floating decimal point called code, and the numbers we are accustomed — through education — to use for counting and reasoning.

A new-born computer (without software) is, a priori, only able to transform 0s into 1s and 1s into 0s. However, it is already more intelligent than one might believe. The only arithmetic it can use is a natural arithmetic, the result of the physical processes which allow it to function: the quantified leap of a recognized state (electronically or magnetically symbolised either by + or by −) to another recognized state (electronically or magnetically symbolised either by — or by + respectively). Before we teach it something by stuffing it full of software, all it knows of floating-point and/or universal arithmetic is the **changing of the sign**. And the *rule of signs* applies:

```
+ and + is +, 0 + 0 is 0
+ and - is -, 0 + 1 is 1
- and + is -, 1 + 0 is 1
- and - is +, 1 + 1 is 0
```

The word "and" has become "plus", but in MODULO 2. It is then to our advantage to replace the functional symbol + with the APL sign ≠. Thus, we can understand to what extent, just how, and why Huygens was right.

Modulo 2 algebra was not yet known in the Enlightenment; it was studied by Galois at the beginning of the nineteenth century, and later by Boole (around 1860) who, thanks to some famous polynomials, codified its isomorphism with logical algebra. Even today, mathematicians still use the ⊕ symbol for + modulo something (among others, 2), as if the definition of exclusion or logical difference were derived from addition, and not the other way round. Unlike the symbol ≠ (which symbolises a true primitive when handling information), addition — which we still class as an "elementary operation" — is not one. It is impossible to make it so in a physical system in a simple way, and it is highly unlikely that we will one day be able to teach a molecule the necessary algorithms. On the other hand, a simple chain of molecules containing alternating single and double chemical bonds is already an organism which can carry out an operation like ≠\ by itself, like a rope hooked to one of its ends.

To understand the effect of ≠\ and its consequences, it is better to learn APL and start experimenting... Thus, we notice that ≠\ induces a genuine, waving, periodic process, chaotic in the extreme, and very similar to electrostatic, magnetic or gravitational effects. When we consider 0 as an empty space, and 1 as either a material or magnetic mass or as a charge, effectively the 1 entities are

repelled only a short distance and are attracted over the long distance across the empty space (0 0 0 ... 0 0 0) without there being any need to understand the mechanism or to create equations of any sort. And ≠\ also represents the ideal model for an optimal decision-making chain, which in other programming languages is expressed laboriously with interwoven loops like:

IF a THEN BEGIN b: = NOT b; IF b THEN BEGIN c: = NOT c; IF c THEN BEGIN... ... END; END; END;

over several pages ... But, if `a` is a watchful neuron, and the sequence `b,c,d...x,y,z` a chain of sleeping neurons (all equal to 0), where `z` is the release mechanism for other processes, `≠\a,b,c...` is quite sufficient to model the moment that `z` wakes up as soon as `a` jumps to 1. It is only if there are *inhibitions* (equal to 1) between `b` and `y` that the chain will stop spreading the wake-up process. `≠\ V` is sufficient in APL if `V` already contains the twenty-six bits necessary for the model. We can extend this to several million bits using one of today's microcomputers.

The computer's new-born arithmetic is a result of the natural physical properties of its constituents, which involve either opposing charges and "magnetic masses" that attract or identical charges that repel, as all physicists know perfectly well. The paradox 0 = 2 from equation (8) is resolved effortlessly. Two electrons with a negative charge (by convention, - or 1), which one unfortunately wants to be in the same place, are going to leap elsewhere, leaving behind a void which will be seen and recorded as an absence of charge (in reality, 0) but always noted down as +, a positive charge (which it is not really) purely by convention. In an empty space (0), we can put nothing (0), or put something (1). We can take nothing away (0) from a full space (1), or we can empty it, setting it back to 0, by taking out the something (1) that was inside. But we certainly cannot fill the space up again, because it would not let itself be pushed around like that.

This waltz of the monopoles and electrons follows an identical logic to that of the information waltz it manages: physical properties and natural algebra are then isomorphic at the level of parity, the logic of the two numbers 0 and 1, the only elements of the set Z/2Z.

Forgotten Matrices

Actually the algebra of the set Z/2Z is the only known algebra in which 1+1 MODULO 2 indeed equals 0. With the help of *programmable APL*, we can express and calculate this more easily in binary algebra as `0≡1≠1`.

So, if one returns to the impossible equation 0 = 2, the reasoning we have followed so far has also constructed a mathematical proof that there is NO other choice. Of course, the equations (1) (2) (3) (4) with which we started can also be posed entirely in modulo 2 algebra in which sum and product exist, the sum identifying with the difference.

So, a = d and a = -d are a single, same solution: a and d must equal 1 if bc is zero. According to equation (6), b or c must therefore be 0, and both can be zero. But the solution ad = 0 with a = d also allows the finding that bc can equal 1. In Z/2Z, this leads to b = 1 and c = 1. Thus, in Z/2Z, there are four solutions to the problem of self-inverse matrices of rank 2, of which three are not trivial:

```
(I)    (anti-I)    (Gh)    (Gv)
1 0      0 1       1 0     1 1
0 1      1 0       1 1     0 1
```

In terms of transformational matrix operators that are neither trivial nor almost trivial (as is the case of anti-I), only Gh and Gv are left. They are the only possible involution operators in Z/2Z of an algebra that is not signed and is necessarily linear, to describe any transformation that is supposed to be reversible.

In Z/2Z there are **four** square roots to I, the matrix-unity of rank 2. But other matrices can be reversed, because condition `(6) ad-bc = 0` can be rewritten as `ad ≠ bc` in binary algebra, where multiplication becomes the logical function of logical AND ∧:

`b ∧ c = 1` leads to `b = 1` and `c = 1`.

In Z/2Z, the product `bc` becomes the function MINIMUM (b,c). In APL it is `b ⌊ c`.

So, we must have `ad = 0` since, if a is equal to 0, d is equal to 1, or vice-versa. As a result, the only other two possible reversible matrices are:

```
(G)          (Gd)
1 1    and   0 1
1 0          1 1
```

These two matrices are not part of the set of four matrices listed above. They cannot be self-inverse; they must, however, be the inverse of each other. Besides, it is necessary that the set of six matrices which combine them forms a multiplying group for the matrix product in Z/2Z.

The product of these two matrices one multiplied by the other, through symmetry, must be commutative and if the two matrices are symmetric the result must be that one of the matrices is itself symmetric, so (I) or (anti-I). The

simple scalar product MODULO 2 of 1 1 by 0 1 has 1 as a result, which eliminates (anti-I). Consequently, the matrices (G) and (Gd) have (I) as a product. So, if the product of the two inverse matrices is the matrix-unity (I), the matrices will each be the cubic root of (I). As transformational operators, they will have the properties of j and of j^2, the complex cubic roots combined with unity.

Without particularly looking for any other examples, these properties appear in all their simplicity, like Botticelli's Venus, where the enigma 0 = 2 provides the key to the electrical and magnetic fields which cause Venus to rise so discreetly from the waves.

Our reasoning showed, before we sought to extend it to matrices of any rank R (R varying from 2 to infinity), that there are no other matrices than those listed here for rank 2. In an entire algebra in modulo 2, which is isomorphous to logical algebra, we know how to define a coherent set of "spinors", that **generate a three-dimensional space.** In fact, we can define a three-sided object with orthogonal axes and three orthogonal planes. This implies that we can at least identify the rotating operator of a third of a turn around the diagonal of this trihedron. It needs a rotation matrix equivalent to j (and so, necessarily, the inverse matrix equivalent to j^2 to allow the inverse).

In complex classical algebra, for rank 2, this is physically impossible; it cannot be achieved without error or approximation in a device trying to put these calculations physically into effect. In no other programming language than APL can we define j (alias G) more simply — and in a rigorously exact manner, in four bits — and then use it.

The symbol G identified with j means "Geniton".

A Geniton generates symmetry in a topological space of parities which are either 0 or 1. The choice of identifying G with j and Gd with j^2, rather than the opposite, is a convention. It is like the trigonometric direction in a complex plane, or the *minus* sign designating the charge on an electron, or 0 referring to False in Logic, or the sign for heat being emitted (in Chemistry, this is a positive sign, but negative in Physics where the system is considered as losing energy). We can see that a system retains its total information during a reversible phase-change; for example when water freezes, it keeps all its information since it returns to its former structure when it melts — as many times as we wish.

Extension to Higher Ranks or Orders

Are there any matrices, in the same algebra, that have the same properties as G and Gd (the inverse of their square), Gh and Gv (self-inverse), and of which the matrices of rank 2 are the sub-matrices, for every value of row R?

The symmetry in relation to the second diagonal, in G2, can replace the geniton in rank 2: this is at the same time, the operation of matrix inversion and that of the elevation to the square. (It is the equivalent of the symmetry in relation to the horizontal axis of the complex plane, called conjugation, which changes the sign of the imaginary part of a complex number.) By auto-similarity, let us replace every 1 in G2 by G2 and every 0 by Z2, the null matrix of rank 2. We obtain G4:

```
1 1 1 1                                                 0 0 0 1
1 0 1 0     of which the symmetrical expression         0 0 1 1
1 1 0 0     in relation to the second diagonal is       0 1 0 1
1 0 0 0                                                 1 1 1 1
```

The matrix product can be found from the products of the sub-blocks of rank 2 (I2 is the unified matrix of rank 2):

```
G2 G2 . Z2 G2
G2 Z2   G2 G2
```

with the result

```
 I2 Z2
 Z2 I2
```

of which the unified matrix is I4.

Through repetition, this reasoning can be extended to infinity. It shows that, by replacing each 1 in G4 directly by G4 and each 0 directly by Z4, G4, then G8, then G16, and so on, (or even directly, G16 from G4), are always the symmetrical inverses about the second diagonal d. Their product, that is, the elevation to the square of each matrix thus obtained, always has as a result — at all its orders — this same inverse.

So, this property will also be preserved if we remove the last line AND the first column of all the matrices G4, G8, G16, etc. The example below of G3, sub-matrix of G4, shows this:

```
1 1 1
0 1 0
1 0 0
```

The symmetry, in relation not to the second diagonal but to the centre, is:

```
0 0 1
0 1 0
1 1 1
```

If we continue suppressing both the last line and the first column, in every matrix G of row R equal to the power of 2 (it will eventually be very large), the characteristic must be maintained. We will have both a construction and a rigorous representation, from 2 to infinity, of the rotation operators j and j^2 in a complex plane. There will be no need to define, as is usual as a precondition, any number other than 0 and 1. Nor will it, above all, be necessary to define *i*, the imaginary root of -1, which first required the definition of negative numbers. (A well-organised head being worth more than a well-filled head, Montaigne.)

These verifications and demonstrations are child's play for those who know APL and its marvellous primitives which allow one to study all these constructions, and even enjoy it (and why not?).

The union of the sets G and Gd, forms the "bi-compound" set. The set is so called because it is double, and contains all the compound axial rotation operators in both senses of rotation. It can also, if we use a computer, be expressed through binary algebra (thus, in bits) without there being any need to use whole, real, or above all complex, arithmetic.

Similarly, we can also easily show, by starting from the sub-blocks G2, G2h, G2v, and Z2, then G4 and so on, and successively suppressing lines and columns, that the horizontal rotations Gh, and/or, via symmetry, the vertical rotations of the "bi-compounds" G or Gd, for each row R from 2 to infinity, will be self-inverse. As a result, they represent the only matrix operators able to carry out direct orthogonal transformation on information sequences (which are, by definition, coded in bits and able to sample every modulation and form all imaginable computer programs) without error or approximation. They will also model both physical and biological changes while conserving the information of the systems — which are, by definition, reversible.

If, as Jacques Brel sang, the [axiom] waltz has taken some time, the blows from its battering ram will get the better of the (Information) wall, little knock by little knock, which can also be modelled by ≠\.

At Work and Play with J

by Eugene McDonnell

Parallel Jacobi

Warning: this column contains material which may either put you to sleep or turn you against applied mathematics altogether. To take some of the sting away I have added a problem which may give you some pleasure in trying to solve. If you completely distrust your ability to read descriptions of programs, no matter how well-written, I advise you to go at once to the section headed "Problem" and avoid the preliminary exposition, or the material following, valuable as it is.

Background

Recently I had need of a program to perform eigenanalyses of square symmetric matrices, and went to Vector 9 3 for January 1993, which had Donald McIntyre's article "Jacobi's Method for Eigenvalues: an Illustration of J". I refer you to that article for McIntyre's lucid explanation of what the method is. In the course of transcribing his 11-line jacobi program, along with its sixteen subprograms and its seven utility verbs, I thought I saw the possibility of speeding it up significantly by taking advantage of some of the parallelism inherent in the problem. I have communicated with McIntyre concerning this, and he tells me that he has used this method for many years, beginning with a Fortran program which he obtained from someone many years ago, transcribing it into APL and recently, as his article shows, into J.

If you look at his program, you will see that at the heart of it are the lines

```
r=. ((cos,-sin),sin,cos) (ia R)} I
Q=. q ip |:r [ R=. r ip R ip |:r
```

The first line amends an identity matrix conforming to the argument matrix by replacing two of its diagonal elements and the two corresponding off-diagonal elements with a 2-by-2 rotation matrix. The elements amended are chosen by finding the off-diagonal element of maximum magnitude, say at row-column indices p,q, and inserting the 2-by-2 matrix items at locations (p,p), (p,q), (q,p) and (q,q). This amended identity matrix r is then used with two matrix products involving R, the original argument, and Q, originally an identity matrix. Those involving R have the effect of zeroing out elements (p,q) and (q,p) of R, while leaving the eigenvalues of R unaltered. When this operation has been performed a sufficient number of times, one finds that all of the off-diagonal elements are

essentially zero, and that the diagonal elements are the eigenvalues of the argument matrix . Those involving Q produce the eigenvectors of the argument matrix.

The valuable book *"Matrix Computations"* by Golub and Van Loan describes this method (section 8.5), but because the search for (p,q) is O(n^2), goes on to suggest that it might be more efficient to select p and q in a more rigid way. For the case of a 4-by-4 argument, they suggest that p and q be selected in the following order:

```
p   q
0   1
0   2
0   3
1   2
1   3
2   3
```

and go back to the beginning, repeating until a sufficiently good solution appears. Golub and Van Loan go on to point out that the rows of the (p,q) table can be arranged in a disjoint, or non-conflicting fashion:

```
   a           b           c
0   1       0   2       0   3
2   3       1   3       1   2
```

and that, in a parallel machine, separate processors can be assigned to perform the individual matrix product operations. For example, in the 4-by-4 case, two processors are needed, so that in step A one processor could do the (0,1) case and the other processor could do the (2,3) case; in step B one processor could do the (0,2) case and the other processor could do the (1,3) case; and similarly for step C. They point out that this method works only for even-order matrices, but that the odd case can be handled by bordering the argument matrix on the right and at the bottom with zeros, and then dropping these excess columns at the end. Thus the rotation matrices needed would look like this:

```
          step A              |        step B          |             step C
        c01   s01    0     0  | c02    0    s02     0  | c03    0     0   s03
       -s01   c01    0     0  |  0     0     0      0  |  0     0     0    0
proc1   0     0      0     0  |-s02    0    c02     0  |  0     0     0    0
        0     0      0     0  |  0     0     0      0  |-s03    0     0   c03
        0     0      0     0  |  0     0     0      0  |  0     0     0    0
        0     0      0     0  |  0    c13    0    s13  |  0    c12   s12   0
proc2   0     0     c23   s23 |  0     0     0      0  |  0   -s12   c12   0
        0     0    -s23   c23 |  0   -s13    0    c13  |  0     0     0    0
```

My contribution enters here. I realized that one doesn't need a parallel machine to obtain the benefits of this parallel Jacobi method. One can combine the rotation matrices, since they are disjunct, as follows:

```
   step A                          step B                          step C
 c01  s01    0    0      c02    0   s02    0      c03    0    0   s03
-s01  c01    0    0        0  c13     0  s13        0  c12  s12     0
   0    0  c23  s23     -s02    0   c02    0        0 -s12  c12     0
   0    0 -s23  c23        0 -s13     0  c13     -s03    0    0   c03
```

This technique reduces the number of matrix products required for a matrix of size n by a factor of n%2. Thus the larger the matrix, the greater the savings. A 10-by-10 problem can be reduced by a factor of 5; a 100-by-100 problem by a factor of 50, and so forth.

The Problem

Now we come to the playful part. As you can see, the row-column pairs to be included at each step must somehow be derived. In the case of a 4-by-4 matrix, we see that step A uses the pairs (0 1) and (2 3); step B uses (0 2) and (1 3); and step C uses (0 3) and (1 2). The problem is to determine a permutation z that produces the desired result. For example, for n=4 any of the following permutations will do:

```
0 2 3 1
0 3 1 2
1 2 0 3
1 3 2 0
2 0 1 3
2 1 3 0
3 0 2 1
3 1 0 2
```

If we set z=.0 3 1 2, we can experiment as follows:

```
 ]a=.(z&{)^:(i. <:#z) i. #z NB. all of the possible permutations
0 1 2 3
0 3 1 2
0 2 3 1

 ]b=.((2!#z),2)$,a NB. exhibit all the pairs of items
0 1
2 3
0 3
1 2
0 2
3 1
```

```
   ]c=.(>/"1)b  NB. mask shows where lead item is greater than trail
0 0 0 0 0 1

   ]d=.c |."_1 b NB. pairs with leading smaller item
0 1
2 3
0 3
1 2
0 2
1 3

   ]e=./:~d NB. pairs in ascending order
0 1
0 2
0 3
1 2
1 3
2 3
```

Problem 1: Define a verb which takes as argument a positive even integer n and yields a permutation which, repeatedly applied to a conforming identity permutation, produces, in successive pairs of items, all possible choices of 2 items from n, with no duplications.

Problem 2: How many of the !n permutations of even order n are solutions to problem 1?

Solutions to this problem may be sent by email to eem@ipsaint.ipsa.reuter.com or by ordinary mail to Eugene McDonnell / 1509 Portola Ave. / Palo Alto, CA 94306 / U. S. A.

Principal verbs

The verbs described below were written for J8. If you are using an earlier version of J you may wish to get your system upgraded. Here are the verbs making up my solution to the parallel Jacobi problem. The two verbs CEA and CEAI produce identical results, but CEA is written using the rhetorical control structures which have been added to J recently (see my last article) and CEAI uses the algebraic control structures which have been in J from the beginning.

Each main verb `CEA` and `CEAI` (Complete EigenAnalysis) takes as argument a square symmetric matrix `A` and returns two conforming matrices, the first with the eigenvalues along the diagonal, and zeros elsewhere, and the second whose columns are the eigenvectors for the corresponding eigenvalues. They each test the parity of the number of rows of `A`. If this is even they laminate to `A` a conforming identity matrix, using the utility verb `IM`, and then apply the subverb

PJ to this initial argument. If it is odd, the action is to border A on the right and the bottom with a column and row of zeros, using the utility verb bz, and then to apply CEA (or CEAI) to this, and at the end removing the bottom row and rightmost column of each matrix of the result with the utility verb ub.

```
CEA =. 3 : 'if. (2|#y.) do. ub"2 CEA bz y. else. PJ y.,:IM y. end.'
CEAI=.(PJ@(,:IM))`(ub"2@(CEAI@bz))@.(2:|#)
```

The subverb PJ (parallel Jacobi) takes as argument an array of two square matrices. It prepares four global variables for use by hsjr: a quantity eps as the product of a globally defined tolerance tol and the Frobenius norm of the first matrix, yielded by the utility verb NF; a quantity s, the number of rows in the first square matrix; a list k, the integers from 0 to s-1; and a list p, a permutation which will be used to alter the arrangement of the atoms of k, using the utility verb mxp. It then employs the verb hsjr (half of s Jacobi rotations) to the limit. At the limit, it yields the desired complete eigenanalysis of the original argument.

```
PJ=. 3 : 0
eps=:tol*NF {. y.
s=:# {. y.
k =: i. s
p=:mxp s
hsjr ^:_ y.
)
```

The subverb hsjr (half of s Jacobi rotations) takes as argument an array of two square matrices. It begins by making a rotation matrix rm, using the verb RM. This rotation matrix is used with the first matrix of the argument to develop PJ0, the next stage of the eigenvalue matrix, one which has a smaller off-diagonal norm than the previous one, and setting to zero any of its elements which are less than or equal to the quantity eps, using the utility verb clean. Next, it uses the same rotation matrix rm with the last matrix of the argument, to develop PJ1, the next stage in the eigenvector matrix. The two matrices are laminated to give the result array.

```
hsjr=.3 : 0
rm=.(k=:p{k) RM {.y.
PJ0=.(({|:rm)+/ .*({.y.)+/ .*rm) clean eps
PJ1=.({:y.)+/ .*rm
PJ0,:PJ1
)
```

The subverb RM (rotation matrix) builds a parallel Jacobi rotation matrix.

It takes as left argument a particular permutation of the integers from 0 through sP1. It fashions this into a two-column table t, then reverses those rows of t in which the first atom is greater than the second atom. An array cs of 2-by-2 cosine-sine matrices, one for each row of t, is formed, using the verb csm. These will be used to amend a matrix of zeros in locations specified by a conforming array of 2-by-2 boxes ix, whose atoms are each a 2-atom list derived from the corresponding row of t, formed using the utility verb CP (Cartesian product). For example, if a row of t is 2 3, the 2-by-2 boxes corresponding to it will be:

```
+---+---+
|2 2|2 3|
+---+---+
|3 2|3 3|
+---+---+
```

Finally, a matrix of zeros is formed, conforming to the right argument y., and the positions in this corresponding to positions given by the matrices of ix will be amended with the corresponding matrices of cs, yielding the desired parallel Jacobi rotation matrix.

```
  RM=.3 : 0
:
t=.((-:s),2)$x.
t=.(>/"1 t)|."0 1 t
cs=.y. csm"2 1 t
ix=.CP t
cs ix}0:"0 y.
)
```

The subverb csm (cosine-sine matrix) takes as left argument a square matrix and as right argument a 2-element list of indices for that matrix, the first element giving a row number and the second element giving a column number, with the row number less than the column number. If the entry in the matrix at that row-column position is zero, the result will be a 2-by-2 identity matrix. If it is nonzero the result will be a 2-by-2 Jacobi rotation matrix, using the verb makecs.

```
csm=.makecs`(=@(i.@2:))@.(0:=<@]{[)
```

The subverb makecs (make cosine-sine table) takes as left argument a square matrix and as right argument a 2-element list of indices for that matrix, the first element giving a row number and the second element giving a column number, with the row number less than the column number. It yields a 2-by-2 Jacobi rotation matrix.

```
makecs=. 3 : 0
 :
```

```
tau=.(((<2#). y.){x.)-(<2#{. y.){x.)%+:(<y.){x.
t=.(*tau)%(|tau)+4 o. tau
c=.%4 o. t
s=.t*c
(c,s),:(-s),c
)
```

The subverb `mxp` (make index permutation) takes a positive even integer as argument and yields a list which is a permutation of the integers from 0 through one less than the argument. The permutation is such that when applied repeatedly to a conforming list, none of the successive pairs in the lists are equal.

```
mxp=.[: C. 0: ; <: , (,~ >:@|.)@>:@+:@i.@<:v
```

Utility verbs

The utility verb `CP` takes a list as argument and returns the Cartesian product of the items of the list.

```
CP=.    {@;"1~
```

The utility verb `IM` takes as argument a matrix and yields an identity matrix having the same number of rows.

```
IM=.    [: = [: i. #
```

The utility verb `NF` takes a matrix argument and yields its Frobenius norm as result.

```
NF=.    [: %: [: +/ [: , *:
```

The utility verb `clean` takes a numeric array as left argument and a positive atom as right argument. It yields a conforming array as result, wherein each element of the left argument with magnitude less than the right argument is replaced by zero.

```
clean=. [ * ] < [: | [
```

The utility verb `bz` takes a matrix argument and yields a similar matrix bordered on the right and below by a new column and row of zeros.

```
bz=.    >:@$ {. ]
```

The utility verb `ub` takes a matrix argument and yields a similar matrix with the rightmost column and bottom row removed.

```
ub=.    _1 _1&}.
```

Test Information

Alter the following value as desired to control accuracy and speed:

```
tol=.1e_6 NB. value should be in the range 1e_2 to 1e_17

 NB. Test matrices

   ]A=.1 1 1 1,1 2 3 4,1 3 6 10,:1 4 10 20
1 1  1  1
1 2  3  4
1 3  6 10
1 4 10 20

   ]m=.1.5 _1 _0.5,_1 2 _1,:_0.5 _1 1.5
   1.5 _1 _0.5
   _1  2   _1
_0.5 _1  1.5

    ]r=.1 1 0.5,1 1 0.25,:0.5 0.25 2
   1    1  0.5
   1    1 0.25
 0.5 0.25    2

NB. test results, using tol as specified above (executed on a
Macintosh)

   CEA A
 0.453835         0         0         0
        0  0.038016         0         0
        0         0   2.20345         0
        0         0         0   26.3047
 0.787275 _0.308686  0.530366 0.0601868
_0.163234  0.723091  0.640331  0.201173
_0.532107  _0.59455  0.391833  0.458082
 0.265358  0.168411 _0.393897  0.863752

   CEA m
         2         0        0
         0         3        0
         0         0        0
  0.707107 _0.408248  0.57735
_9.8829e_10  0.816497  0.57735
  _0.707107 _0.408248  0.57735
   CEA r
_0.0166473         0         0
         0   1.48012         0
         0         0   2.53653
  0.721208   0.44428 0.531483
 _0.686348   0.56211 0.461473
 _0.093729 _0.697601 0.710329
```

Bodyguard of Lies

by Peter Merritt

Well, several months and one new motherboard later, welcome to part two of my encryption piece for simple (minded) APL-ers. I must start by expressing my thanks to Ray Cannon for constructing and publishing my apology for lateness and my equally sorry solution to the original problem! Slipping out of my hair shirt for a while, I was a little disappointed at the lack of reaction in the letters page, but, as a regular contributor to historical magazines, I've come to expect it. Anyway, if anyone else has read this far, it's time for the solution and some background notes on the ideas behind it.

As was mentioned last time, the message started life as a rank-2, 4-row by 30-column simple character matrix. The encryption process itself was in two stages, each involving a form of substitution (normally the easiest to crack, but I've added a twist). Firstly, I randomly generated a table of 256 2-character codes (using only upper- and lower-case letters, and the numbers 0-9). Then, using the order of occurrence in ⎕*AV* to provide indices, I generated a numeric matrix of these indices with the same dimensions as the original text object. So far, so bland.

Now the problem was how to both disguise the original data AND transmit enough of the decryption key for the receiver — which is where my original APL doodling came in. Using a randomly generated number for each element of two vectors with the shape of the first and last dimensions, the ROTATE symbol was then applied, thus 'jumbling' the elements — in effect substituting one element for another. So now I had a collection of character tables and numeric vectors which needed to be 'packaged' in some regular form, ready for transmission, storage, or publication in the national press (depending on content, of course). As this was to be the 'simple' form for Vector competition purposes, the package was assembled as follows:

> PART (1) — the table of 2-character codes (or as much as necessary; in the example, only characters, numbers and ONE punctuation symbol were used, or 63 chrs in all);
>
> PART (2) — the three numbers which describe the object's original dimensions, but expressed as codes from the above table (using the numbers as positional information — 6 chrs);

PART (3) — the 34 numbers which were the rotation figures (using their positions in the code table again; a further 68 chrs);

PART (4) — the 240 characters derived from the 120 code-position numbers which were produced by the original, simple look-up (or in other words, the data — you knew we'd get to it eventually, if only you hung on long enough.....).

The eventual character vector was then split into 5-character sets, so beloved of espionage systems in the 30's to early 50's (the gaps easing transmission/recognition), the remaining odd set being 'padded' with randomly generated garbage (as opposed to the sophisticated garbage which preceded it). Now, to transmit the key-table does form a heavy overhead for small messages, but this becomes insignificant as the amount of text increases (as the table is the same size whatever the circumstances). The most obvious feature of the final vector is the unique pairs at the start, as opposed to the later repeating patterns — it is this break which is the best clue to solving this puzzle, the rest being extended game-playing. Interestingly enough, several of the testers who have tried the competition at first rejected some of their results because they were not expecting a mix of numbers and text — they ASSUMED that all results should conform or be significant EITHER as characters OR as text.

This was just the beginning, however — we can get much more devious than this, which is where the title of this piece comes in [Winston Churchill's instruction for the protection of the Overlord invasion plans was to ".....shield the truth with a bodyguard of lies....."]. Amongst the other techniques which could be tried, again using the simple application of the rotation operator, are:

- to rotate each of the 5-character sets (thus destroying the obvious unique key at the start);
- to rotate the order of the sets by a given number (either positive or negative);
- to add, somewhere near the start, a large-ish number (perhaps date-based?), together with a reasonably large prime number — this has NO relation to our encryption technique but is used by so many others that automated decryption can go off merrily down the garden path for hours. Still, it keeps the computers busy.....

Of course, once you start to use multiple (optional?) methods, then a further signal needs to sent to the other end indicating which methods apply and, equally important, the ORDER in which they are applied. One suggestion I'd like to make concerns the use of binary-equivalents as disguised selection vectors (that is to say, if methods 1, 2, 4 & 5 apply, this is 1 1 0 1 1 as a selection vector,

but can be sent as a single number — or the code table substitute — of 27). This is also a good method for getting rid of the large-number dross mentioned earlier.

Oh, on a final note those with access to speech-to-text phonetics software might like to consider the advantages of using these files as a basis — again very useful in an age of automated decryption where the machine has a copy of the complete Oxford English (or perhaps Oxford Serbian?) Dictionary built-in, but which doesn't have "Heh-LLohw" {= Hello} in its look-up table, and so would reject any method which obtained this text as 'wrong' — perhaps semi-logical, lateral humans aren't redundant quite yet after all....

Making Menus with Causeway

by Adrian Smith

Introduction

At the heart of any significant Windows application is the menu-bar for your top-level form. For the user, this menu-bar is the gateway to all that APL code you spent days or weeks developing - so the design and structuring of the menu options deserves more thought than it often gets. However the menu can serve another equally useful purpose – as on-line documentation which will point some future APLer at the function names in the workspace, and that will go some way towards describing what they do.

This article describes how menus are constructed in the Causeway utility set, and includes examples from a number of recent APL systems, as well as a couple of classics (from the SAP package – see Appendix-2) which may help you to avoid the worst excesses of the over-enthusiastic Windows programmer. An old stand-alone version of the menu-builder utility function (in Dyalog 6.3 code – see Appendix-1) is included for those who would like to try these ideas, but do not want to take on the whole Causeway workspace.

Getting Started

In its simplest terms, a menu is simply a caption (for the user to see) and either an action (to be executed) or a pointer to another menu. It could look like:

```
Hello:2+2
World:⍳12
```

This might be a vector of vectors, or a simple character matrix. The text to the left of the colon is what the user sees; the text after the colon is executed by APL when the user selects that option:

```
Gui_menu 'Hello:2+2' 'World:⍳12'
```

Dyalog-7 users can load the Causeway workspace and try this for themselves. The effect is to create a pop-up menu (where the mouse is at the time) and echo either `4` or `1 2 3 4 5 6 7 8 9 10 11 12` to the APL session when an option is selected. This is fine for a simple pop-up (obviously you would normally have a function call here), but in a real system the first thing you need is normally a

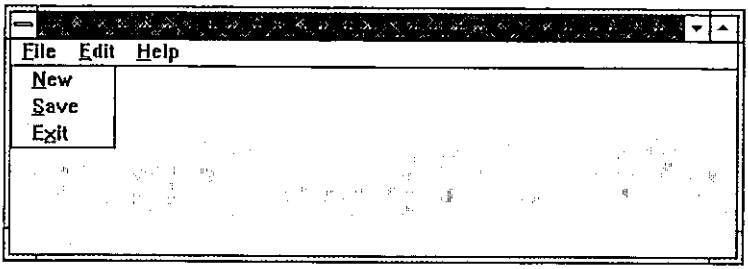
menu-bar for your main form. The convention that most Windows programs adopt is to hang a sub-menu underneath each and every entry on the menu-bar, so this simple style needs to be extended slightly to cope with (recursively) nested menus. I based my ideas on the Motif standard (Unix users can look at `.mwmrc` which is the Motif root menu definition) to give a definition like this:

```
[root]
 &File>file
 &Edit>edit
 &Help>help

[file]
 &New:2+2
 &Save:3+3
 E&xit:'Farewell cruel world'

[edit]
 ... and so on
```

I hope you can see the pattern! Anything after the colon gets executed (as before) and anything after a > chains to another menu. The indenting and spare lines are just for clarity, but the square brackets are essential, and the names must match exactly. Of course you can nest the sub-menus as deep as you like, but do read the Microsoft style guide before you give all your users a bad attack of the screaming heebie-jeebies with 4-level cascading menus. If you cannot hang everything you need on a single layer of pull-downs (with judicious use of right-mouse pop-ups for context-sensitive functionality) your application is too damn complicated and you should go back to the drawing board until you have simplified it.

Let's make a small form and see the effect of hanging a menu definition on it:

```
        Gui_init ''
        'ff' wc 'Form'
        'ff' Gui_menu mm
Farewell cruel world
```



As you can see, I chose the 'Exit' option and the corresponding message was echoed to the APL session. A more realistic example (note that any line starting with a hyphen is treated as a separator) might be:

```
[root]
 &File>file
 &Edit>edit
 &Dictionary>dict
 &Arrange>arrange
 &Options>options
 &Help>help

[file]
 &New: NEWFILE
 &Open: OPEN ''
 &Save: SAVE 0
 Save &As: SAVE 1
 &Merge:MERGE ''
 Print Pre&view...: print_view
 &Print ...: print_sel
 Print Setup ...: psetup
 --- SEPARATOR ---
 E&xit: EXIT

[edit]
 &Undo: undo
 &Goto page ...: jump ''
 Go &Home: home
 &New page ...: new ''
 &Copy page ...: copy
 --- SEPARATOR ---
 Select &All: select_all
 --- SEPARATOR ---
 &Rename page / change descr ...: rnm
 &Remove page from pad ...: zap
 --- SEPARATOR ---
 &Maintain Function/Process info ...: fnproc

[dict]
 Collate &Transaction list ...: collect_tran
   ... etc

[arrange]
 &Link selected objects: linkup
   ... etc

[options]
   ... etc

[help]
 &Contents ...: Guide
 &About ...: ABOUT
```

From the user's point of view, this definition is entirely adequate, but what can the APL coder (in this case me) hope to get out of it? This workspace was completed in late 1993, so I have by now forgotten most of the function names - what better way to find my way into the code than to put the menu definition on screen:

```
)ed Δmenu
```

... and double-click my way to the underlying code? In a sense, the main menu definition of a Windows workspace performs the same role as the ⎕*LX* in an old mainframe application - it is the starting point from which the maintenance programmer finds his or her way to the APL code. Given this fact (which I only began to realise some while after I finished writing this particular system), what can we add to the definition to help matters? An obvious possibility would be some judicious comments:

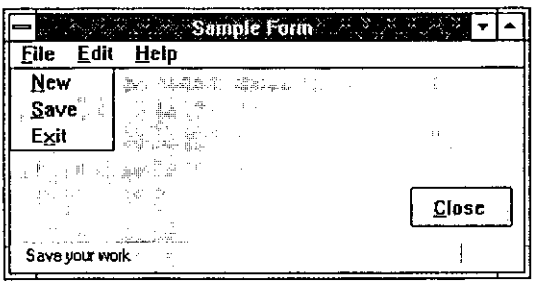
```
[file]
 &New: NEWFILE          ⍝ Start a new drawing
 &Open: OPEN ''         ⍝ Open an existing drawing
 &Save: SAVE 0          ⍝ Save your work
 Save &As: SAVE 1       ⍝  ... with a new name
 &Merge:MERGE ''        ⍝ Merge with another drawing
... and so on
```

... but perhaps it would be handy for the user to see those comments as well! Here we must move to Dyalog-7, so be warned that the *Gui_menu* code quoted in the appendix does not support this extra feature. This time, I am going to make my form with the Causeway designer, and specify the menu definition as the 'data' for the main form:

```
      Dbx 'xx'
     Disp xx
```

FM	*Sample Form*	588 144	144 364	*mm*				
ST		119 0	24 364					
CL	*&Close*	84 284	28 72					

```
Gui_call xx
```



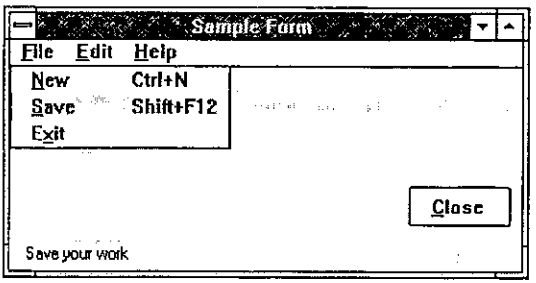
```
      +mm
[root]
 &File>file
 &Edit>edit
 &Help>help

[file]
 &New; Start a new file :2+2
 &Save; Save your work  :3+3
 E&xit; Sign off        :'Farewell cruel world'

[edit]
 ... and so on
```

Here I have added a 'Status Bar' object to my form - Causeway automatically set this as the 'Hintobj' for any children of that form – and picked out the part of the caption following the first semi-colon as a hint. This way, the user sees the comments as he or she runs the mouse up and down the menus, and the APL developer sees the comments too! Again, the alignment is ignored by `Gui_menu`, but it helps the programmer a lot.

Adding Hot-keys

Again, this is specific to Dyalog-7, and I rather wonder if I am beginning to overload the definition. However, here is how I did it:

```
[file]
 &New=Ctrl+N; Start a new file     :2+2
 &Save=Shift+F12; Save your work   :3+3
 E&xit; Sign off                   :'Farewell cruel world'
```



Anything in the caption after an = sign is stripped off, parsed and set as the accelerator key for that option. Now if I were to hit Ctrl+N I would see 4 echoed into the APL session. I also turn the = into a tab character, which makes the menu look much neater to the user.

Taken from Life

Those of you who came to Swansea in July will remember the help-file builder I used to illustrate some features of a Causeway system. Here is its main menu:

```
      ↑∆menu
[root]
 &File>file
 &Edit>edit
 &Options>options
 &Help>help

[file]
 &New;Starts a new file                :NEW                  :∆file
 &Open;Opens an existing file          :OPEN                 :∆file
 &Save=Ctrl+S;Saves your work          :SAVE 0
 Save&As;Takes a copy with a new name  :SAVE 1               :∆file
 &Export .TXT ...=Ctrl+E;Makes a plain ASCII file...:Export tsel
 -----
 &Build .RTF ...=Ctrl+R;Makes a suitable file ...   :Build
 &Hex error ...=Ctrl+H;Quick search for any ...     :browse_rtf ∆file,'.RTF'
 &Testfly .HLP ...=Ctrl+T; Tries out the Conten ... :Testfly
 -----
 E&xit:Gui_post 'SC'

[edit]
 &Find=Ctrl+F; Locates a text string anywhere in the topic list :find_txt

[options]
 &Set Copyright;Adds copyright details to contents page:
∆copyright+∆copyright Win_input 'Please enter your name and the date ...'
 &Headers ...;Sets up type-styles and other op...   :headers
 &Chapters ...;Sets up chapter headings and sequence:    Chaps:∆topics
 &APLFont ...; Toggles the use of an APL font in the editor :setfont
 &Icon ...; Picks an Icon ...: ∆iconfile+∆iconfile Win_input 'Icon file ...'

[help]
 &About ...:ABOUT
 &Help Contents...=F1:WHLP 'helpstuf.hlp'
```

Note that I have truncated some of the hints to save the text wrapping across two lines. Just in case you worry about the time taken by APL to riffle through all this and get the form on screen:

```
      Gui_menu ∆menu
```

... has a pop-up on screen in about 0.8 seconds (DX2/50 processor), and of course you only do this once, when the application is started. I think this is a small price to pay for a menu structure which is easy to read, and could potentially be pulled in from a simple ASCII file at startup time. Maybe your users would like to redefine some of your structuring, eliminate some of the less interesting options, translate the hints into Finnish?? All they need is a text editor and this article!

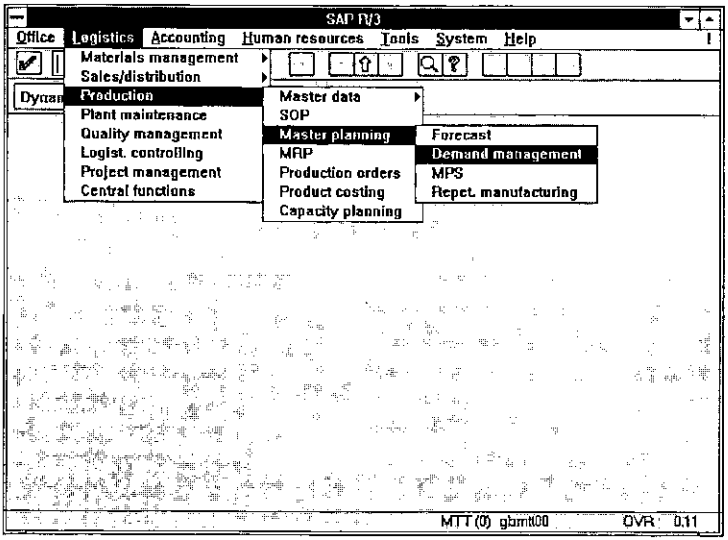
Appendix-1: the Stand-alone Code

```
 (_pnt)Gui_menu _arg;_dq;_mt;_grp;_cap;_nm;_ps;_itm;_inm;_iex;_ict;_sct;⎕IO
⍝ Build menu structure defined in <_mt> at section [grp]
⍝ This either hangs from a menubar, or is rooted. Rooted
⍝ menus are popped up at the cursor and locally DQed.
 ⍎(0=⎕NC'_pnt')/'_pnt←''''' ⋄ ⎕IO←1
 ⍎(3>|≡_arg)/'_arg←⊂_arg' ⋄ _arg←3↑_arg,2⍴⊂'' ⋄ _dq←0
 _mt _grp _cap←_arg ⋄ _ict←_sct←1
⍝ Check for top-level menus, which may be owned by a FORM.
⍝ If so, make a MENUBAR to hang them on!
 →(0=⍴_pnt)↑Root ⋄ →('FORM'≡_pnt ⎕WG'TYPE')↑MBar      ⍝      <<<< WATCH IT <<<<
Child:_pnt←_pnt,'.menu',_grp ⋄ _pnt ⎕WC'MENU'_cap ⋄ →Sect
MBar:_pnt←_pnt,'.menubar' ⋄ _pnt ⎕WC'MENUBAR' ⋄ →Sect
Root:_pnt←'rootmenu1' ⋄ _pnt ⎕WC'MENU' ⋄ _dq←1 ⋄ →Sect
⍝ Now chop out the right section of the structure ...
Sect:_ps←'[' ⊃ _mt ⋄ →(⍴_grp)↑Top
 _ps←_mt⍳⊂'[',_grp,']' ⋄ →(_ps>⍴_mt)↑0
Top:_grp←_ps↓_mt ⋄ _grp←(¯1+(⊃¨_grp)⍳'[')↑_grp
 ⍝ Check each entry, and make item or another menu ...
Next:→(⍴_grp)↓Done ⋄ _itm←⊃_grp ⋄ _itm←(+/∧\' '=_itm)↓_itm
 →(⍴_itm)↓Skip ⋄ →('-'=⊃_itm)↑Sep
⍝ Split off name and ⍎ part at ':'
 _ps←⍎'/_itm⍳'':''' ⋄ _inm←_ps↑_itm ⋄ _iex←_ps↓_itm
 →(':'=¯1↑_inm)↑Menu ⋄ _inm←¯1↓_inm
⍝ ================ MENUITEM ====================
 _nm←_pnt,'.item',▼_ict ⋄ _ict+←1
 _nm ⎕WC'MENUITEM'_inm ⋄ →(' '∧.=_iex)↑Skip
⍝ Set items to execute action on select ...
 _nm ⎕WS'event' 30 'Gui_exec' _iex ⋄ →Skip
⍝ ================ SEPARATOR ===================
Sep:(_pnt,'.sep',▼_sct)⎕WC'SEPARATOR' ⋄ _sct+←1 ⋄ →Skip
⍝ ================ SUBMENU =====================
⍝ Next menu down gets our caption as its title (Yuk)
Menu:_pnt Gui_menu _mt(_iex←'[]')(¯1↓_inm)
Skip:_grp←1↓_grp ⋄ →Next
Done:→_dq↓0 ⋄ ⎕DQ _pnt



_ex Gui_exec _msg
⍎ _ex     ⍝ also not required in Dyalog 7
```

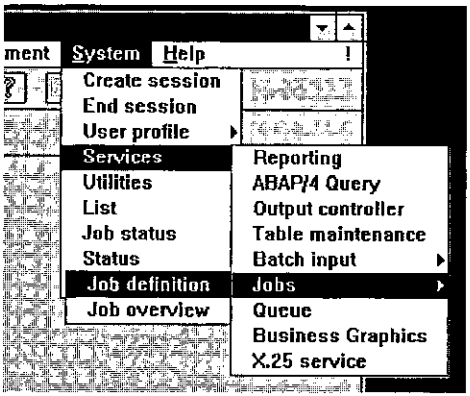
This will be fine in Dyalog 6.3 – but Dyalog 7 users will need to change *FORM* to *Form* (thanks lads) to make it work. The nasty names are to avoid conflicts with executed code (relevant to pop-ups only, as *Gui_menu* won't be on the stack when your application runs otherwise).

Appendix-2: How not to do it!

Here are a couple of screen snaps from the SAP system, which illustrate very nicely the pitfalls of over-enthusiastic menu design:



Seen like this, it doesn't look all that bad, but think of the poor user who wants to be in *Demand Management* and can't remember where it was. She starts off looking under *Materials Management* – about 6 × 6 × 4 options to explore, before homing in on *Production*, but a quick scan of the sub-menu shows nothing interesting ... and so on.



Even for the expert, navigating accurately down and across and down and across requires a lot of concentration and is both visually and physically tiring.

Sometimes, the sub-menus get so far across the screen that they start popping up to the left, and confusion reigns supreme.

Don't do it!

J INSCRIPTION
0 :

by Richard Oates

Release 1 version 7 of J introduced an unobtrusive keyboard process for explicit definition. It is started with a zero left argument. I call it *inscription* and do it with the adverb train `IN:=. 0 :` . Inscription can be scripted. It is much more convenient than the prior definition technique. It has validated script and made the saved workspace obsolete. A *script* is a file where every line is a J sentence. To script a file is to read it as a keyboard, like Unix standard input.

I describe a J utility which edits DOS script. I like DOS more than Windows but less than Nextstep. Forty tacit verbs are defined in four explicit verbs. One tacit equivalent is fixed for each explicit verb. The utility runs from the explicit verbs or the tacit equivalents. It uses the adverse and agenda control conjunctions.

Introduction

My profile scripts eight adverb trains and three verbs. A conjunction and either of its arguments is a *train* which makes an adverb waiting for the other argument. In J Release 2 some arguments of the foreign `1:` conjunction are changed.

```
h:=. 0!: NB. Host       IN:=. 0 :       NB. Inscribe
f:=. 1!: NB. File       v:=. "_         NB. Verb from noun
n:=. 4!: NB. Name
m:=. 5!: NB. Map        C=. (13{a.)v:   NB. Carriage return
s:=. 8!: NB. Screen     L=. (10{a.)v:   NB. Line feed
d:=. 9!: NB. Dial       e:=. (1.0 0)v:  NB. Empty
```

The J editor `9 s:` acts on line text but not on verbs. A *line text* is a vector where each line ends with a line feed. Before version 7 an argument of explicit definition could be a sentence in quotes, a list of boxed sentences, or the open thereof. Boxed sentences are clumsy. In an early version of the language I wrote a utility that used `9 s:` to edit a verb in the workspace. I then saved workspaces and ignored script. It worked like APL del so I called it `Nd` for "Not del".

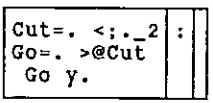
Inscription suspends execution while the user enters J sentences one after another. Execution resumes on entry of a bare parenthesis. The new verb is in the

workspace. The process is serial but the sequence can appear in script that can be edited freely before it is scripted into J. If a complete locale is defined in script, the whole can be scripted at the start of the session. A *locale* is a workspace or part of one. See "Version 7" below. Saved workspaces are no longer needed and support will be withdrawn. When version 7 appeared I changed `Nd`. It now applies `9 s:` to text from a DOS script.

Boxed Sentences vs Inscription

Brute force at the keyboard or in script is required to manage a list of boxed sentences.

```
   a=. 'Cut=. <;._2'
   b=. 'Go=. >@Cut'
   c=. ' Go y.'
   (Table=. (a;b;c) : '')   NB. Outer paren's display the verb
```




```
   Table'cup box '
cup
box
```

Inscription suspends the indented prompt and sentence execution. It has a double begin/end structure. A bare colon separates the monad from the dyad. A bare right parenthesis terminates the process. Explicit and tacit verbs and adverbs and conjunctions can be inscribed. Explicit verbs are selected by a zero right argument to definition: `0 : 0` . This inscription defines the same verb.

```
   Table=. 0 IN:
Cut=. <;._2
Go=. >@Cut
 Go y.
:
)
```

`Table` can be "edited" by bringing each sentence from the output to the input area with Ctrl/Enter. If the verb is longer than this or will be needed on another day it is defined in script.

In addition, line text can be inscribed with `4 IN:` . If each line is a J sentence the line text can be used in a subsequent definition, `Jtext : Y` . I do not discuss this. Most tables are easier to inscribe than to write in script with primitives like

append or shape. I inscribe them in line text and cut and open each text on the end-of-line indicator to produce a J noun, as described below in the verb Ndn.

End-of-Line Indicators

Carriage return and line feed appear at the end of every line in DOS script. Line feed appears at the end of every line in line text. These indicators shatter boxed maps.

```
   (65{a.)v:          (13{a.)v:          (L=. (10{a.)v:)
┌─┬─┬─┐            ┌─┬─┬─┐            ┌─┬─┬─┐
│A│"│_│            │"│_│ │            │ │ │ │
└─┴─┴─┘            └─┴─┴─┘            │"│_│ │
                                       └─┴─┴─┘
```

I convert them to verbs with the constant conjunction in `v:` . *Constant* `noun"noun` is not as well known as *rank* `verb"noun` . Constant makes a verb that ignores its arguments and returns the noun on its left as its result. The verb `2:` works the same way.

```
   's'2: 3       65{a.       's'((65{a.)v:)3       a.i.
's'(L=.(10{a.)v:)3
 2             A          A                      10
```

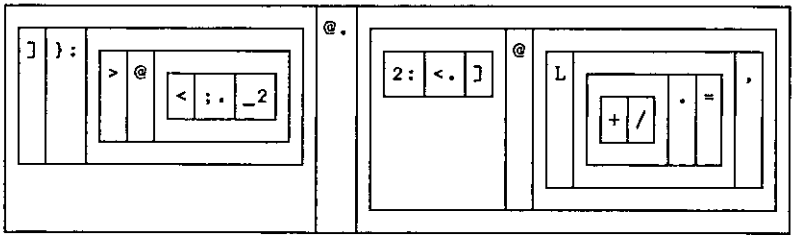
In `Nd` the hook `CL` sticks carriage return and line feed on the end of every line in a table.

```
  y.      a.i.y.      a.i.CL y.      (CL=. ,"1 (C , L))      CL y
e       101         101 13 10                                ,"1
v       118         118 13 10                              y   \
e       101         101 13 10                                    ,
r       114         114 13 10                                  /   \
y       121         121 13 10                                C     L
                                                             y     y
```

Cup Utility

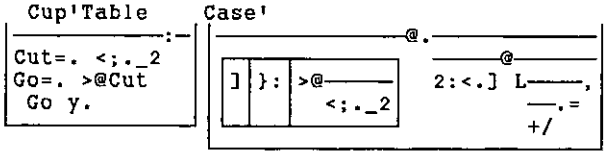
The `Nd` utility is mapped with Cup. I introduce Cup with `Table` and a tacit verb. In `Case` the tie ` conjunction forms a gerund from verb arguments. A *gerund* is a list of boxed noun atomic representations. The right argument of the agenda `@.` conjunction selects one of the three cases for execution.

```
   ( Case=. ]`}:`(>@(<;._2))@.((2: <. ])@(L +/ .= ,)) )
```




```
   $Case 99 98 97 10 99 10{a.
2 3
```

The Cup utility maps verbs with lines instead of boxes. A gerund is not a verb so the boxes are retained. Cupped maps are less precise than boxed maps but snug ambiguous display is not foreign to J or apl.



The Nd Program

There are four explicit verbs but no branches or labels. Explicit reference is confined to the last sentence in each definition. The tacit verbs appear in bottom-up order but can appear in any order. I expect to find local tacit verbs useful even within a named locale. The tacit equivalents for monadic `Nd`, `Ndp`, `Nds` and `Ndn` are fixed in Exhibit B.

Monadic	Nd	Edits any script
	Ndp	Defaults full DOS name for Nd
	Nds	Scripts the verb or the line text noun
	Ndn	Makes a J noun from line text
Dyadic	Nd	Copies a script that inscribes one object

Nd Verb

Four steps make Nd work like APL del:

1 f: File Read inputs a C/L vector

9 s: J Edit takes and makes a line text

2 f: File Write outputs a C/L vector

3 h: Silent Script inputs a verb or a line text noun

Monadic Nd reads the file named in its argument, purges carriage returns, hands line text to the J editor, restores carriage returns, rewrites the file, and scripts it. "Silent" kills the echo of the "keyboard" on the screen. If the file, say Voice , does not exist the adverse :: conjunction in Read places Voice=: 0 IN: at the top of a fresh screen. If Voice is to be a noun, not a verb, change 0 to 4. If you misspell the name and get a fresh screen when you were expecting a definition, erase the top line to kill an unwanted script. Nd does not fail easily:

```
   Nd';Voice'
Name? ;Voice
```

Dyadic Nd copies a file to a new DOS name. It changes the name on the top line to match the DOS name. For example, 'Fax'Nd'Voice' changes Voice=: 0 IN: to Fax=: 0 IN: . Monadic Nd assumes the name at the top of a script matches the DOS name.

```
 Cup'Nd'
```

```
NB. Nd'name'       Not Del  NB.'new'Nd'old'    Copy

CL=. ,"1 (C , L)            Write=. ] 2 f: <@;@[
Tag=. ;@(CL&.>@(<;._2))
EOL=. _2&}.@Tag@(] , L)     Jname=. }.@;@{:@}:@{.
Out=. EOL 2 f: <@;@[        From=. ] +./\ .= '='v:
Write=. Out`]@.(0: = #@])   With=. [ , From # ]
                            Old=. 1 f:@<@;@{:
View=. 9 s:                 New=. Jname With Old

Old=. 1 f:@<@; -. C         Copy=. {. Write New
Jname=. }.@;@{:@}:          Run=. Nds@{. [ Copy
New=. Jname , '=: 0 IN:'v:  Go=. Run@(Ndp"0@])
Read=. Old ::New            No=. ('Name?'v: ; ])v:
                             Go ::(>@No) x.;y.
Edit=. ] Write View@Read
Go=. (Nds [ Edit)@Ndp
No=. 'Name? 'v: , ":
 Go ::No y.
```

Ndp Verb

Ndp extends the argument of Nd with three defaults which complete the DOS file name. Each part of the name is boxed. Ndp is immune to the length of the first default. Its argument can override the second and third. Ndp"0 appears in dyadic Nd. Ndp is the only verb that needs to be adapted to a different operating system.

```
  Ndp 'Table'
```

\J7	\I	\Table	.Js

```
  N=. 'W\Table.Bak';'W\Much\Deeper\Table.'
  Ndp"0 N
```

\J7	\W	\Table	.Bak		
\J7	\W	\Much	\Deeper	\Table	.

```
  ;@Ndp"0 N                                NB. DOS names
\J7\W\Table.Bak
\J7\W\Much\Deeper\Table.
  }.@;@{:@}:@Ndp"0 N                       NB. J names
Table
Table
  $Ndp';Voice'
0
  Cup'Ndp'
```

```
NB. Ndp'n' Ndp'd\n.e'   DOS path

Class=. -@1: < 0 n:@<@}.@;@{:@}:
Cut=. e.&'\.' <;.1 ]
Hit=. +./@([ =/ ' \ .'v:)
Cull=. ;@(-.@Hit # ])
Dos=. (#~ Class)@Cut@Cull

Root=. '\J7\'v:
Dir=. 'I\'v:
Ext=. '.Js'v:
Default=. Root ; Dir ; ] ; Ext

 (Dos Default)@(>@])y.
```

Nds, Line Text and Ndn

Nds scripts the verb or the line text noun. It executes Ndn when the inscription defines a noun. Noun cannot be a tacit verb because local names in Nds would mask global names for the name class 0 n: verb.

Ndn cuts and opens a line text to make a J table. Ndn is also applied in my profile to each noun after all have been scripted, and it appears in the script of any noun that is not an open table, as seen in Df . Reform cannot be a tacit verb because a tacit copula =: does not act on nouns.

```
   Cup'Nds                          Ndn'
┌──────────────────────────────:─┐ ┌──────────────────────────────────:─┐
│NB. Nds Ndp'n'   Script        │ │NB. Ndn'n'  Noun from line text    │
│Script=. 3 h:@<@;@[            │ │Reform=. '' : '(<x.)=: <y.'        │
│Noun=. '' : '2=0 n:<y.'        │ │Shape=. (2: <. ])@(L +/ .= ,)      │
│Form=. e:`(Ndn@])@.Noun        │ │Up=. ]`):`(>@(<;._2))@.Shape@".    │
│Jname=. ).@;@{:@}:             │ │ e:@(] Reform Up)y.                │
│ (Script Form Jname)y.         │ └───────────────────────────────────┘
└───────────────────────────────┘
```

```
   2 h:<;Ndp'Df' NB.Script
   Df=: 4 IN:
19940128
\J7\W\Manu.Js
)
   Ndn'Df'
   Df=: (".@{. ; {:)Df
   Df
```

19940128	\J7\W\Manu.Js

```
   1 f:<;Ndp'Df' NB.Read file
Df=: 4 IN:
19940128
\J7\W\Manu.Js
)
Ndn'Df'
Df=: (".@{. ; {:)Df
```

```
   Go=.>@Cut=. <;._2   NB. Tacit copula
   Cup'Go Cut'
┌>@─────┐ ┌<;._2┐
│  <;._2│ └─────┘
└───────┘
```

Version 7

In addition to validating script, Version 7 introduces an error stack, suspended execution, and named locales. Suspension permits sentence execution in the local environment, and resumption. Named *locales* are alternate symbol tables. I have not used them yet, but I did put all utility scripts in a utility directory and the scripts for each application in a directory for that application. I expect each directory will become a named locale. My profile scripts a directory verb, runs it to get the names in the directory, scripts Ndn, scripts the other objects, and moves to the next directory. Taken together, these changes make 7 the first version of J that can be used outside the classroom.

I would like some additions. A foreign conjunction that edits script with the J editor and scripts the verb. An inscription which makes an open character noun of rank 2 or less; I do not inscribe numeric tables. Deletion of trailing blanks from each line of an inscription. A foreign conjunction that scripts a whole directory

into a named locale; I had no conflicts with caseblind DOS names when I converted the objects in each version 6 workspace to script but this quirk of DOS needs to be outwitted. Fix `f.` as a conjunction instead of an adverb; sentences like the ones in Exhibit B would be simplified by an additional verb that could suspend name replacement.

Conclusion

Tacit definition simplifies documentation. After the arguments of a tacit verb are described it's just J all the way. Further is better. Further enlarges the space where unexplained data cannot lurk. The name of a tacit verb is more potent than a comment — it appears more than once. A comment to the right of the definition can provide an additional hint, but verbs like `CL` cannot be fully described without turning the program into a haystack. Tacit programs, like J explicit and APL programs, are best read actively at an open keyboard.

A workspace must be cleaned before you save `2!:2` or `)SAVE` it. A locale made from script is never saved. A directory does not get as dirty as a workspace. To clean it sort on the timestamp and check the scripts at the bottom. Other system support is available like selective backup and string search and replacement through all script in a directory. In APL2 I wrote programs for jobs like these.

When del appears in an APL session the log goes to lunch and all hell breaks out. After lunch you can display the function but you don't. It's just "paper" you can't use. Inscription fills the black hole of del. Session and definition — any definition — are one.

Unlike APL del, explicit definition permits independent specification of the monad and the dyad. Is `0 : 0` the simplest possible way to define a verb?

Exhibit A: Booting

Nd edits explicit definition and is produced from explicit definition, so how do you start? First you need the eight adverbs. A "given" name (a name which ends in a colon) cannot be reassigned in version 7 until it is erased so put the adverbs and e: in your profile with DOS. Also C and L.

Start with Ndp because it is needed by Nd. Key the nine tacit definitions in Ndp above from Class to Default. If you get an error copy the definition from the output area to the input with Ctrl/Enter and correct. After all are accepted key the final sentence substituting 'Table' for y.. If you do not get the right four-box result Ctrl/Enter single sentences to correct. When 'Table' works try other arguments. When all work:

```
s=. <'\J7\I\Ndp.Js'  NB. 1 Name script
s 2 h:<''            NB. 2 Begin script out
Ndp=: 0 IN:          NB. 3 Begin inscription
                     NB. 4 Ctrl/Enter the 10 sentences
                     NB.    using( y. )not( 'Table' )
:                    NB. 5 End monad
)                    NB. 6 End inscription
2 h:<''              NB. 7 End script out
3 h: s               NB. 8 Silent script
```

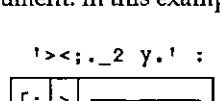
Display Ndp and try Ndp'Table'. Script appends to itself. If Ndp does not work 0 h:'erase ',>s and Ctrl/Enter steps 2-8 correcting as necessary. When Ndp works repeat with Nd substituting 3 h:@<@; for Nds in Go. Try Nd'Table'.

When Nd works use it to define Nds. When Nds works on verbs substitute Nds for 3 h:@<@; in Nd by keying Nd'Nd'. If you have bad luck you can fix up Nd with DOS edit. As an alternative, copy Nd with DOS first and immediately change the J name on the top line to match the new DOS name.

Exhibit B: Tacit Definition

J without tacit definition is simpler than APL. J with tacit definition is richer. It is best learned from the adverb :20 which proposes a tacit definition for its argument. In this example 3 d: 2 5 sets the maps to boxed 2 and to linear 5.

```
   '><;._2 y.' : 20
```

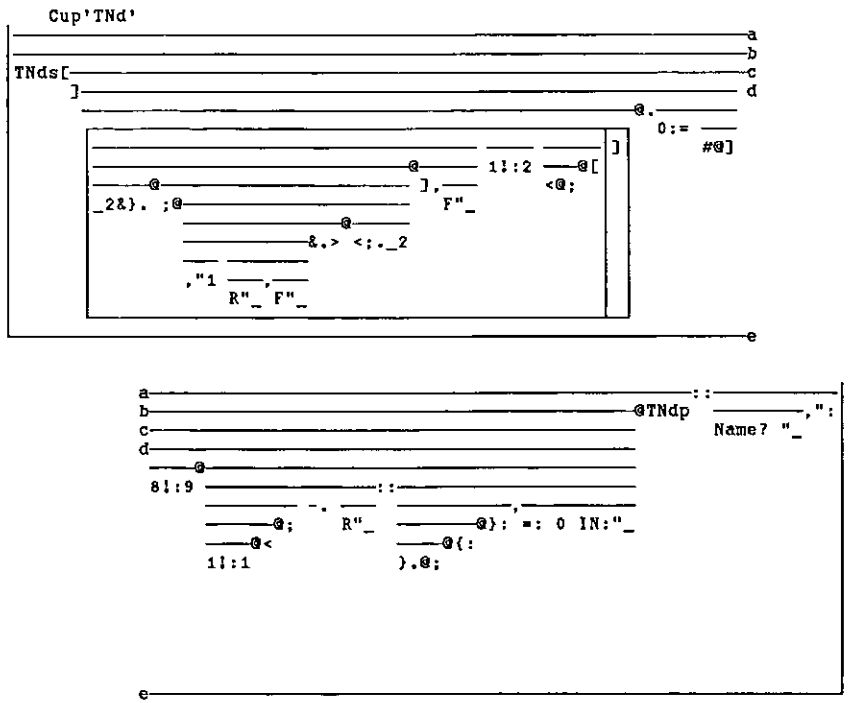


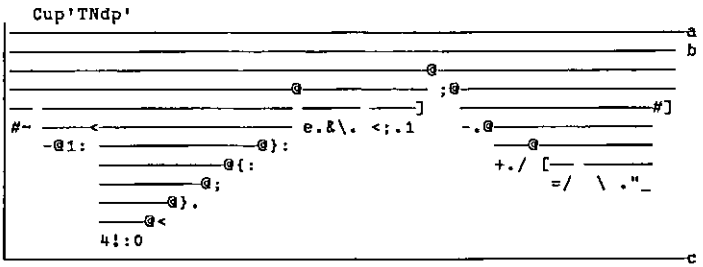

```
[: > <;._2
```

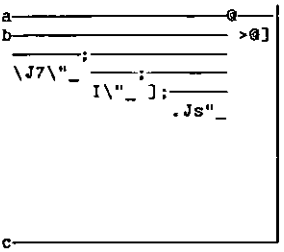
Tacit definition is the most remarkable animal on the Iverson farm, and the most rewarding. It may facilitate formal manipulation of the program but it is not just soft chips. The fix adverb `f.` substitutes definitions for names. The map it juxtaposes is the most readable text of any program. Tacit equivalents for monadic `Nd`, `Ndp`, `Nds` and `Ndn` are defined by inserting a line in each and running `Nd` on some noun (not verb). Each insert becomes the penultimate line in the definition. `TNd` is about forty percent faster than monadic `Nd`.

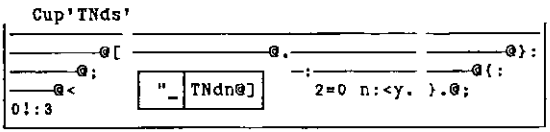
```
TNd=: (TNds [ Edit f.)@TNdp ::(No f.)
TNdp=: (Dos Default)@(>@]) f.
TNds=: Script f. e:f.`(TNdn@])@.(Noun f.) Jname f.
TNdn=: e:@(] Reform Up)f.
```

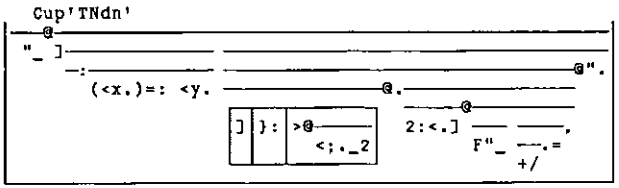
Define `C=. 'R'v:` and `L=. 'F'v:` and rerun `Nd` before mapping the verbs.




Cup'TNdp'
e.&\. <;.1
4!:0


\J7\"_
I\"_];
.Js"_


Cup'TNds'
"_ TNdn@]
2=0 n:<y. }.@;
0!:3


Cup'TNdn'
(<x.)=: <y.
] }: >@
<;._2
2:<.]
F"_ -.=
+/

Nice One, Microsoft!

from Gérard Langlet via Adrian Smith (Vector Production)

Following the note on fonts in the last Vector, I received a splendid *cri de cœur* from Gérard, not un-naturally upset about the lack of the *oe* diphthong from APL2741. He also mentioned a problem in using the Windows clipboard to transfer APL code from Winword back into APL (either Dyalog or PLUS III) for final testing before publication.

Suspecting (as one does) the APL interpreter in one's life, I set out to investigate. Sure enough, function listings from Winword arrived in Dyalog looking very strange indeed — then I spotted that the execute had come through as a hyphen. If you look carefully at page 106 of Vector 11.2 you will see that execute is opposite en-dash — a vital clue! What the morons of Redmond have (allegedly — they might read this) done is to substitute what they thought you might have meant for what you actually put!

So — for " you get " and for " you get ", and yet (oh joy) for ' you get '. For • you get o (that's right, they just knew you meant little letter o when you typed bullet) and so on. Try this from Winword to Write (or Notepad or Works) and you will see what I mean.

The work-around is to save your document in Write format and clipboard from there, or just use a sensible word-processor like MS Works which does not exhibit this 'helpful' behaviour. At least we can rule out Winword as a candidate for *Vector OnLine.*

Index to Advertisers

All queries regarding advertising in VECTOR should be made to Gill Smith, at 01439-788385; CompuServe: 100331,644.

Submitting Material to Vector

The Vector working group meets towards the end of the month in which Vector appears; we review material for issue n+1 and discuss themes for issues n+2 onwards. Please send the text of submitted articles (with diskette as appropriate) to the Editor:

Anthony Camacho,
11 Auburn Road, Redland,
BRISTOL, BS6 6LS
Tel: 0117-9730036
Email: acamacho@cix.compulink.co.uk

Authors wishing to use Windows Write should contact Vector Production for a copy of the Vector APL TrueType font and Vector APL typebox.

Camera-ready artwork (e.g. advertisements) and diskettes of 'standard' material (e.g. sustaining members' news) should be sent to Vector Production, Brook House, Gilling East, YORK YO6 4JJ.

Tel: 01439-788385 (any time)
Compuserve: 100331,644.

British APL Association: Membership Form

Membership is open to anyone interested in APL. The membership year normally runs from 1st May to 30th April, but new members may join from 1st August, November or February if preferred. The British APL Association is a special interest group of the British Computer Society, Reg. Charity No. 292,786

Name: _____
Address: _____
_____
Postcode / Country: _____
Telephone Number: _____
Email Address: _____

Membership category (please tick box): . . . . . .

UK private membership	£12	❑
Overseas private membership	£14	❑
Airmail supplement (not needed for Europe)	£4	❑
UK Corporate membership	£100	❑
Corporate membership overseas	£135	❑
Sustaining membership	£430	❑
Non-voting UK member (student/OAP/unemployed only)	£6	❑

PAYMENT — in Sterling only

Payment should be enclosed with membership applications in the form of a UK Sterling cheque to "The British APL Association", or you may quote your Mastercard or Visa number.

I authorise you to debit my Visa/Mastercard account

Number: ⌊⌊⌊⌊ ⌊⌊⌊⌊ ⌊⌊⌊⌊ ⌊⌊⌊⌊ Expiry date: ⌊⌊ | ⌊⌊

for the membership category indicated above,

❑ annually, at the prevailing rate, until further notice
❑ one year's subscription only

(please tick the required option above)

Data Protection Act:
The information supplied may be stored on computer and processed in accordance with the registration of the British Computer Society.

Signature: _____ Send the completed form to:

British APL Association, c/o Rowena Small, 8 Cardigan Road, LONDON, E3 5HU

The British APL Association

The British APL Association is a Specialist Group of the British Computer Society. It is administered by a Committee of officers who are elected by a postal ballot of Association members prior to the Annual General Meeting. Working groups are also established in areas such as activity planning and journal production. Offers of assistance and involvement with any Association matters are welcomed and should be addressed in the first instance to the Secretary.

1994/95 Committee

Chairman:	Dr Alan Mayer 01792-205678x4274 a.d.mayer@swansea.ac.uk	European Business Management School, Swansea University, Singleton Park, SWANSEA SA2 8PP
Secretary:	Sylvia Camacho 0117-9730036	11 Auburn Road, Redland, BRISTOL, BS6 6LS
Treasurer:	Nicholas Small 0181-980 7870	8 Cardigan Road, LONDON E3 5HU
Journal Editor:	Anthony Camacho 0117-9730036 acamacho@cix.compulink.co.uk	11 Auburn Road, Redland, BRISTOL, BS6 6LS
Activities:	Duncan Pearson 01483-33329 100265.1564@compuserve.com	7 Lynne Court Chesham Road GUILDFORD, Surrey GU1 3LR
Education:	Dr Ian Clark 01388-527190 100021.3073@compuserve.com	9 Hill End, Frosterley Bishop Auckland Co. Durham DL13 2SX
Technical:	Jonathan Barman 01488-648575 100116.1030@compuserve.com	Hill Top House, East Garston, NEWBURY, Berks RG16 7HD
Projects:	George MacLeod 01442-878065 100412.1305@compuserve.com	Greymantle Associates Ltd., Bartrum House, Ravens Lane, BERKHAMSTED, Herts HP4 2DY
Publicity:	David Eastwood 0171-922 8866 microapl@applelink.apple.com	MicroAPL Ltd., South Bank Technopark, 90 London Road, LONDON SE1 6LN
Recruitment:	Jon Sandles 01904-411635	22a Arthur Street, Lawrence Street, York YO1 3EL
Administration:	Rowena Small 0181-980 7870	8 Cardigan Road, LONDON E3 5HU

Journal Working Group

Editor:	Anthony Camacho	0117-9730036
Production:	Adrian & Gill Smith	Brook House, Gilling East, YORK (01439-788385)
Advertising:	Gill Smith	Brook House, Gilling East, YORK (01439-788385)
Support Team:	Jonathan & Bridget Barman (01488-648575), Ray Cannon (01252-874697), Richard and Adam Weber (01302-539761), Sylvia Camacho, Duncan Pearson, John Searle (0181-858 6811), David Ziemann (0171-267 8032), Jon Sandles (01904-411635)	

Typeset by APL-385 with MS Word 5.0 and GoScript
Printed in England by Short-Run Press Ltd, Exeter

VECTOR

VECTOR is the quarterly Journal of the British APL Association and is distributed to Association members in the UK and overseas. The British APL Association is a Specialist Group of the British Computer Society. APL stands for "A Programming Language" — an interactive computer language noted for its elegance, conciseness and fast development speed. It is supported on most mainframes, workstations and personal computers.

SUSTAINING MEMBERS

The Committee of the British APL Association wish to acknowledge the generous financial support of the following Association Sustaining Members. In many cases these organisations also provide manpower and administrative assistance to the Association at their own cost.

Compass R&D Ltd
10 Frederick Sanger Road
Surrey Research Park
GUILDFORD, Surrey GU2 5YD
Tel:01483-302249
Fax:01483-302279

HMW Trading Systems Ltd
Hamilton House,
1 Temple Avenue,
LONDON EC4Y 0HA
Tel:0171-353-8900
Fax:0171-353-3325
Email:100020.2632@Compuserve.com

Soliton Associates Ltd
Groot Blankenberg 53
1082 AC Amsterdam
Netherlands
Tel:+31 20 646 4475
Fax:+31 20 644 1206
Email:ljh@soliton.com

Manugistics
2115 East Jefferson St
Rockville
MARYLAND 20852 USA
Tel: (301) 984-5412
Fax: (301) 984-5094
Email:aplsales@manu.com (US)
Email:intl@manu.com (International)

Dyadic Systems Ltd
Riverside View, Basing Road,
Old Basing, BASINGSTOKE,
Hants, RG24 0AL
Tel:01256-811125
Fax:01256-811130

MicroAPL Ltd
South Bank Technopark
90 London Road
LONDON SE1 6LN
Tel:0171-922 8866
Fax:0171-928 1006

Dutch APL Association
Postbus 1341
3430BH Nieuwegein
Netherlands
Tel:03474-2337

Insight Systems ApS
Nordre Strandvej 119A
DK-3150 Hellebæk
Denmark
Tel: +45 42 10 70 22
Fax: +45 42 10 75 74
Email: insight@inet.uni-c.dk