VECTOR

APL and the Internet

Plus ...

The Journal of the British APL Association

A Specialist Group of the British Computer Society

ISSN 0955-1433

www.vector.org.uk

Vol.13 No.4 April 1997

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 1996-97

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	10	5
Non-voting Member (Student, OAP, unemployed)	£6	1	1

The membership year normally 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, Mastercard or JCB, 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

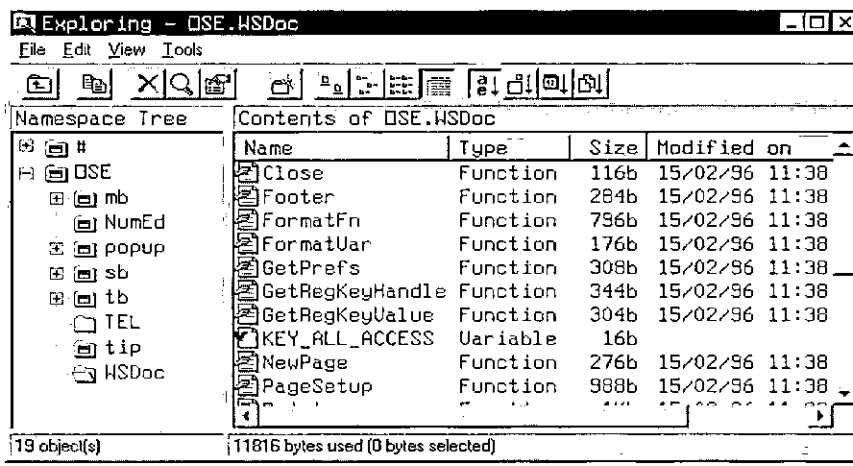
Advertisements in VECTOR should be submitted in typeset camera-ready format (A4 or A5) with a 20mm blank border after reduction. Illustrations should be photographs (b/w or colour prints) or line drawings. Rates (excl VAT) are £250 per full page, £125 for half-page or less (there is a £75 surcharge per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



Dyalog APL Namespaces let you ...

- **Organise** your workspace into self-contained sub-systems
- **Encapsulate** functions and variables *within* GUI objects
- **Isolate** utilities from your application code

Plus The new Version 8 **Explorer** lets you browse namespaces, drag-drop objects from one to another, and a whole lot more.

That's why Dyalog APL/W remains the **professional** choice. For further information, contact Dyadic or your local distributor today.

MICROSOFT® WINDOWS™ COMPATIBLE

Dyadic Systems Limited, Riverside View, Basing Road, Old Basing, Basingstoke, Hants. RG24 7AL, United Kingdom.
Tel:+44 1256 811125 Fax:+44 1256 811130 Email: sales@dyadic.com

Editorial

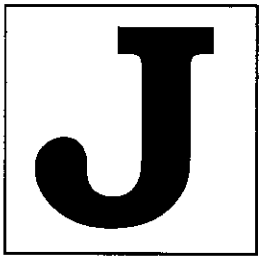
by Duncan Pearson

As this issue's correspondence section shows, the old APL vs. J argument has raised its head again. Don Mattern's letter in the last issue has prompted a number of responses. It is seen by some that Vector is dominated by J and that there is little of interest for APLers. As Norman Thomson's response points out, the quantity of J is less than that of APL and indeed has not changed considerably in the last four or five years. What could account for this unfavourable impression however is the number of regular "columns" that are about J. This is due to the fact that a small number of J users are willing to give freely of their time each quarter to produce regular articles of high quality. Norman Thomson and Eugene McDonnell are the prime examples, having contributed for many years. Some of Norman's articles moreover are of benefit to APLers as he often demonstrates how the application of J ways of thinking can enrich our APL.

What is perhaps not clear is that the presence of these articles is not driving out APL articles that would otherwise appear. Most of the other J articles that appear are unsolicited whereas in order to achieve something of a balance we in the Vector Working Group must go out and ask APL authors to contribute. If we take as an indication of the relative interest in the two languages their submission rates to comp.lang.apl, then it would appear that Vector devotes a disproportionate amount of space to APL. This would not however be a fair measure to use as J is considerably easier to submit to an internet newsgroup. Still it gives an indication of the willingness of J users to talk and write about what they are doing.

In general any article that we believe would be of interest to a broad range of members of the association will be included. If the article is about the use of APL or J to solve an abstruse technical problem then, in order that the article be accessible to the majority of readers, an introduction explaining the background of the problem and its importance should precede the technical content. Articles about the recreational use of APL and J are welcomed. Apart from the simple benefit of enjoying ourselves, quite often the techniques used can be adapted by others to solve serious problems. Many of the vendors of APL systems provide articles which both interest and illuminate us. However in fairness to the association's sustaining members, many of whom are also APL vendors, blatant plugging of products should be kept for paid advertisements. Articles will also be welcome that are about neither APL nor J but that demonstrate techniques or approaches that would be of interest to us.

In summary we will continue to welcome (and solicit) material from both communities and try to achieve a balance that is acceptable to the members of the association. If in your opinion we are failing to do that then write or email to tell us and we will take your opinions into account in planning future issues.



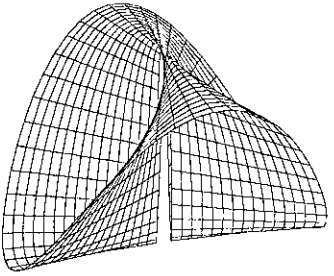
Look for Release 3.04: A complete OpenGL interface for high quality graphics, plus a native version for the Mac PowerPC!

Check out our web site:

Now on the Web page: complete systems for Windows 95/NT, Windows 3.1, Mac, Linux, Sparc and RS/6000. Download the full system for evaluation or to upgrade an earlier version, and use the convenient order form to register your system and order other ISI products.

See you at APL97!

Distributed by:
Strand Software Inc.
19235 Covington Court
Shorewood, Minnesota
USA 55331
Tel (612) 470-7345
Fax (612) 470-9202



www.jsoftware.com

Quick Reference Diary 1997

Date	Venue	Event
August 17-20	University of Toronto	1997 International Conference on APL "Share Knowledge, Share Success"
November 3-5	Orlando, Florida	APL2000 User Conference

Conference News

APL97: an International Conference on APL, August 17-20
Organized by: Toronto APL Special Interest Group (SIG)

The *"Share Knowledge, Share Success"* theme has been chosen to celebrate APL and J. This year's conference will focus on providing "hands-on" opportunities in computer laboratory settings. Experts and fellow-users will share their knowledge to help assure your success!

The 1997 **APL2000 Conference** will be held Monday November 3, 1997 through Wednesday November 5, 1997 at the Sheraton World Resort in Orlando Florida. Presentation and Training schedules will be posted as soon as they are available.

Dates for Future Issues of VECTOR

	Vol.14 No.1	Vol.14 No.2	Vol.14 No.3
Copy date	6th June	5th September	5th December
Ad booking	13th June	12th September	12th December
Ad Copy	20th June	19th September	19th December
Distribution	July 97	October 97	January 98

CORRESPONDENCE

Strategies for Optimising the API Problem

From: Robert Bernecky (bernecky@eecg.toronto.edu) March 6 1997

Reading Roger Stokes' letter about "Nick Cox's Antecedent Precipitation Index" problem, (Vector 13.3 p.8) I realized that the problem is simply solved by using the recurrence relationship function that APLers have known and loved (or hated) for years:

```
    ∇ sequence←addend recurrence multiplier;t
[1]   ⍝ sequence[i]←addend[i]+multiplier[i]×sequence[i-1]
[2]   ⍝ sequence[⎕IO]←addend[⎕IO]
[3]   ⍝ 0∈multiplier ↔ 0.  ⎕IO-independent.
[4]   ⍝ (sauce/recurrence.1 from the sharp apl utility library)
[5]    t←×\multiplier
[6]    sequence←1/t×+\addend÷t
    ∇
```

To get the effect of `0.5 api1 1 2 3 4`, you would use the expression `1 2 3 4 recurrence 0.5 0.5 0.5 0.5`. This works fine (I naively thought) and is substantially faster than any of the functions presented by Stokes, taking 110 milliseconds for a 1000-element argument on my 486/DX4-100 running APL+Win V1.3.00 under Windows 3.1.

Unfortunately, this function breaks down just at 1024 elements, because the `×\` in line 5 starts producing zeros, which in turn causes the divide in line 6 to fail. This sort of failure is typical of APL algorithms that attempt to mimic iterative algorithms by the use of scan. The partitioned scan utility functions are also prone to this sort of data sensitivity.

Daunted, I returned to iterative solutions, observing that `api1` builds its result by repeated catenation. Here is the version of `api1` that I used:

```
r←x api1 y;p;api;i;k
⍝ antecedent precipitation index NJC 15 April 1996
⍝ Rewritten in APL+Win 1997-02-24
⍝ Robert Bernecky
⍝ Repetitive catenate version (slow)
⍝
⍝ dcl double scalar x
⍝ dcl double vector y
k←x
p←y
api←,0.0
:for i :in ⍳⍴y
 api←api,p[i]+k×api[i]
:endfor
r←1↓api
```

This approach to iterative construction of a result is almost always a *bad* idea in APL and J. It results in a non-linear amount of work being done, instead of a linear amount. To see why this is, note that iteration I copies I elements of the old value of `api` to generate the new value. Thus, *N* iterations will copy `(N×(N-1))÷2` array elements to generate an N-element result.

To eliminate this extra work, I rewrote `api1` as `api3`, preallocating result space and using index assign to reduce the amount of array copying:

```
r←x api3 y;p;api;i;k
⍝ antecedent precipitation index NJC 15 April 1996
⍝ Rewritten in APL+Win 1997-02-24
⍝ Robert Bernecky
⍝ This version avoids repetitive catenation,
⍝ by preallocating result and using indexed assign.
⍝
⍝ dcl double scalar x
⍝ dcl double vector y
k←x
p←y
api←(1+⍴y)⍴0.0
:for i :in ⍳⍴y
 api[i+1]←p[i]+k×api[i]
:endfor
r←1↓api

api3 =. 3 : 0
NB. antecedent precipitation index NJC 15 April 1996
NB. k api p
NB. This preallocates result and uses
NB. indexed assign. Robert Bernecky 1997-02-24
:
k =. x.
```

```
p =. y.
api =. (1+#y.)# 0.0
i =. 0
while. i< #y. do.
api =. ((i{p)+k* i{api) (i+1)} api
i=. i+1
end.
}. api
)
```

These functions compute the final result shape, preallocate a result of the appropriate shape, then perform indexed assign or merge to insert the partial results into the final result, thereby performing work proportional to the shape of the result.

The results of this rather simple optimization were enlightening. On an argument size of 10000 elements (shorter arguments ran too fast to measure on my system), I observed mean execution times in seconds as follows:

Benchmark	*APL+Win* 1.3.00	*J*3.0
*api*1	28.95	24.88
*api*3	1.65	38.11

The performance of *api*1 confirms benchmarks done by others, suggesting that J tends to execute slightly faster than APL on many codes. In the APL version of *api*3, we see that performance has improved significantly, due entirely to elimination of the repetitive array copy operations. APL application programmers should be aware that they can often make substantial improvements to programs through such preallocation techniques in any algorithm that builds its result by aggregation.

Even if the final result size is unknown, it is often possible to obtain most of the benefit of preallocation by the following method: first, allocate a large intermediate result, *TMP*. On each iteration, if the new data for this iteration fits within *TMP*, use indexed assign to amend *TMP*. If it does not fit, catenate a generous amount of filler onto *TMP* and try again. At the end, drop off any remaining filler. Thus, we see that applying a bit of craft to the problem results in a dramatic performance improvement.

Turning to the J version of api3, I was surprised to see that performance is, in fact, worse than it was for api1. It is evident that the J merge operation did not detect the ability to work in place and avoid the copy operation. During execution, it copied the entire merged array over and over again. This resulted in N*N array elements being copied, explaining the degraded performance.

I reported this bug to Roger Hui, the implementor of J. He was also puzzled, as we both knew that this case of merge should work in place, which would make its performance roughly equivalent to that of the APL version of `api3`. When I get a fix, I will report updated figures for this benchmark.

So, we now have a real-world application that can not be solved, in general, using non-obvious, non-iterative algorithms, such as `recurrence`, due to loss of numerical precision. Incorrect results aside, the non-obvious nature of these algorithms makes it hard for naive programmers to design and use them. It is time for APLers to admit that iteration has its place in a programmer's toolkit.

In order to use iteration effectively, the APL programmer must have the confidence that execution times for iterative code are similar to those written in C or FORTRAN. This can only be done with adoption of compiler methodology, such as that provided by APEX (Bernecky: apl93, apl95, MSc Thesis in press). The APEX compiler is able to take iterative APL code and translate to C code suitable for execution on multiprocessor or uniprocessor systems. I used APEX to compile `api1` and `api3`, and obtained the following results on 10000-element arguments, using Linux (Slackware'96) and gcc on the same hardware platform used above:

Benchmark	*APL+Win* 1.3.00	*J3.0*	*APEX*	*APEX Speedup*
api1	28.95	24.88	0.35	82.7
api3	1.65	38.11	0.04	41.2

Note that the `api1` code compiled with APEX is running about 80 times faster than the interpreted code. However, the `api3` timings are so low that they are being affected by timer precision on the PC. When I reran the `api3` APEX benchmark with a 800000-element argument, it took 1.68 seconds, about the same as the interpreter with a 10000-element argument. Thus, it also obtains an 80-fold speedup over the interpreter.

Since the performance of `api3` compiled with APEX is within 20% of the same program written in FORTRAN and compiled with g77 (compiler options were: g77 -O2 -m486 -dalign), it is clear that APL programmers no longer need to fear iteration.

Robert Bernecky
Snake Island Research Inc

APL or J Association?

From: Norman Thomson 20 February 1997

Don Mattern's letter on the above subject in Vector 13.3 contains enough factual error, prejudice and misjudgement to demand a reply.

First the bulk of Vector is not on J. A rough survey of the technical contents pages over the last few years shows the following percentages of APL/J:

1992 : 69/31, 1993 : 76/24, 1994 : 77/23,
1995 : 59/41, 1996 : 67/33, 1997 (one issue):58/42.

Secondly, the APL character set may look like Greek but in fact they have only four symbols in common. Arguably the need for a special character set has greatly hindered APL's progress in the programming world. Although it is true that the iconic properties of some of the APL characters is lost by using ASCII characters, whether programs are intuitive or not is a matter for the stylistic skills of the writer combined with the effort expended by the reader in mastering the specific meanings which J imposes on these symbols in context. For those who are not prepared to make this effort to say that J looks like "comic book invective" is no more valid a criticism than that of those castigators of APL who scorn it because it "looks like Greek". Thirdly, Adrian Smith among others will, I am sure, be happy to provide instances of current commercial applications of J.

Fourthly, try as I will, I can find nothing on J in past numbers of Vector which can be described as "theoretical"; as for its being academic, APL itself began as an academic project prior to its commercial success, and J represents in many respects the continuation of the pursuit of innovation which APL began. Perhaps it is worth quoting Richard Procter's conference report on APL96 (see Vector 13.2, pp.80-81) — "J has a great future with its superior approach to array handling ... It is being embraced by APL people as a potentially useful array processing language, and seen by some as better than APL". Fifthly, Mr Mattern says that "IBM has never endorsed J". Subsequent to the introduction of APL, IBM has not been in the business of inventing new languages, so it is hard to see why IBM's endorsement (or not) is of any more significance than that of any other large commercial IT user or supplier. Sixthly, APL has nearly thirty years of published history behind it in the form of magazines, technical reports, books and conference papers. Admittedly this material is not always as accessible as one would like it to be but at least it represents at least a twenty year start over J, — the surprising thing is surely that Vector contains so little of the latter.

Mr. Mattern says that he subscribed to Vector to receive APL information — let him if he wishes tear out the thirty-odd or so percent of pages of J, and he still has just that, at a subscription which is roughly equal to the "voluntary" ACM donation which accompanies the dues for the corresponding American publication. That all this is so is a tribute to the enormous amount of voluntary effort on the part of the Vector production team (to mention names would be invidious) whose untiring efforts have ensured that Vector has appeared regularly, informatively and uninterruptedly for thirteen years, a record of which no other APL publication can come within striking distance.

From: A J Wilson 11th February 1997

I am writing to concur with Don Mattern's letter about J in the January edition of the Vector. The number of articles about J seems disproportionate to its usage.

I do not believe Vector is serving the best interests of its APL users.

From: Stefano Lanzavecchia 9th March 1997

Personally I like the attitude of "Vector" regarding J. I don't find the amount of papers about J as a language and about applications written in J excessive at all. On the contrary, I, as an APLer and as programming language freak, find it very stimulating and very valuable: trying to dive into the mind of a J programmer can give a completely new perspective when looking back at problems to be solved in APL. Notice that, due the quantity of work I have to do, I don't actually have time to play with J in front of a keyboard, and without Vector, that I can read lost in the sofa while sipping a cup of hot tea on a rainy Sunday morning, I would simply not have the possibility of improving my knowledge in that fascinating world.

P.S. in Vector Vol.13 No.3 Page 131 the definition of the verb `w` should be:

```
w =. ~. ^ #/.~
```

Please note the last "Reflex" that gives a completely different meaning to the train `#/.` .

A BAA Meeting at Network City

reported by Duncan Pearson

The BAA meeting at Network City in London on Thursday January 16th was an interesting departure from normal practice. Network City is a café that provides access to the Internet on its own machines for an hourly charge. The BAA hired a room for the evening with about twenty machines, and paid for the Internet access for the three hours of the meeting. It was possible to buy tea and coffee but unfortunately there was very little in the way of food (true to form I snaffled the last bun just before the meeting opened) and no alcoholic drinks at all. The machines were well configured and the staff very proficient and helpful. The only small problem was caused as usual by the APL character set. We needed to log in as network administrator on each machine in order to install the APL font before everyone arrived. The staff however were unruffled by this requirement and everything was achieved in time.

As the attendees were arriving we were treated to a demonstration of the Dyalog APL/W TCP/IP support with a Dyalog workspace on one of the machines running a "chat" session with a Turing machine in Basingstoke that purported to be Peter Donnelly. While the session was being set up by John Scholes in London he was having a concurrent phone conversation with Peter and could see the words appearing in his session at the same time that he could hear the keys being pressed in Basingstoke.

The meeting opened with a short talk by Ray Cannon about what we could hope to get from the internet and what resources were available to help us to get going. This was a little hampered by the lack of any demonstration facilities and we soon dispersed to separate machines while Ray and a couple of other relatively knowledgeable folk wandered round and helped people in trouble. This seemed quite successful and most of us learnt things from others.

One of the main interests was the display of APL characters on World Wide Web pages and we were shown how to configure both Internet Explorer 3 and Netscape 3 so that they could display the Vector Web pages correctly. There is an explanation of how to do this in Ray's article, which follows. We also saw the HTML (hyper-text mark-up language) elements that the pages were using to specify the APL font using Netscape's "view source" facility. Ray demonstrated a couple of search engines and used them to search for APL references on the Web. Most of them are mentioned in Jon Sandles' article in this issue on getting the

most from the Internet. Ray also demonstrated some Perl and some Java with which he had enlivened his home page.

Another feature of the meeting was the Internet Chat virtual meeting that was running at the same time and in which a number of us took part. Ray had set up a "room" and we were joined in it by a group at a satellite meeting in York as well as a couple of random passers by who dropped in, one of whom started to abuse us when nobody seemed to be talking to him. However he left as soon as it was explained that we were all APLers and after a lengthy explanation from me about what APL is, a sadly missed opportunity I feel, to add to the APL flock. The conversation between the York contingent and London was lively. The standard chat-up line seemed to be "Will you come and work for me?" to which the reply was "No, will you come and work for me?". There has to be an opportunity in there for someone.

On the whole I feel that the meeting was a success and that most of the attendees benefited from it. I enjoyed meeting in the evening when everyone seemed more relaxed. It is a pity that the venue was a little short of variety in the food and drink department.

APL Characters on the World Wide Web

by Ray Cannon

Background

World Wide Web pages are scripts written in a language called HTML (hyper text mark-up language: a sub-set of STML). A "web-browser" is a program that (amongst other things) interprets HTML to create a web page on your screen. In HTML, text is marked with "TAGS" which define how the page is to be displayed. Some "TAGS" have "OPTIONS". An "OPTION" normally consists of a string such as "keyword=value". Unknown tags or unknown options within tags are ignored by web browsers.

Until recently, it was not possible for the author of a web page to specify the font to be used, just its size and spacing (via the "FONT" tag). Now, a new option of the "FONT" tag has been added (FACE), that allows a web author to specify a list of preferred fonts by name, e.g. "<FONT FACE=APLNET,APLPLUS>"

The two most common types of web browser for the PC are currently Netscape Navigator and Microsoft Internet Explorer. The latest versions (3.0 and above) of these recognise the "FACE=font list" option.

Suggested Standard APL Font Face

This means it is now possible to put APL characters directly into web pages and specify a set of alternative fonts to view them with. The resulting page with APL text can then be seen by any recipient using a PC with a suitable browser if they have at least one of the specified fonts installed on their PC.

(See http://www.vector.org.uk/aplfont.html for information on how to configure some older browsers which do NOT support the <FONT FACE=xxxx> tag.)

It will be a great help if we (the APL community) can standardise on a single font encoding for displaying APL characters in web pages. By that I mean that *any* APL font meeting this standard will have the same APL characters in the same relative positions. Note, this is *not* the same as the ⎕*AV* order.

I propose that we adopt the font encoding shared by two of the most popular APL for Windows interpreters: APL+WIN and Dyalog APL/W. This is also the font encoding used by Adrian Smith's freely available APL2741 font.

To be able to produce HTML pages with APL, which are also compatible with older web browsers (via the method referenced above) the fonts used need to be (internally) flagged as Monospaced (i.e. fixed pitch) and ANSI (as against Symbol). Not all the fonts with the "proposed standard APL encoding" meet these two requirements. However, Adrian has made a Monospaced ANSI version of his APL2741 font available for Web use under the name of APLNET, which can be downloaded from the Vector web site (www.vector.org.uk). Dyadic have also recently modified their DYALOG STD TT font to match these requirements, which can be downloaded from their site (www.dyadic.com).

Since the "face" tag may contain a list of fonts, I suggest we should specify a minimum of 3 fonts, one from the APL2000/Manugistics/APL*Plus stable, one from Dyadic and one from Adrian.

For example: <font face="APLNET, DYALOG STD TT, APLPLUS">

Other APLs

Now, I know that this font encoding is not compatible with many other APLs (e.g. IBM APL2 , APL68000, APLIWIN or any EBCDIC-based mainframe APL), but then no single font is. However both APLNET and Dyalog STD TT are available for free over the Internet, whereas there is no APL2 encoded font that I know of in the public domain. (There is for instance no "TryAPL2 for Windows 95/NT" to the best of my knowledge.)

If say IBM APL2 users want to put up web pages for other IBM APL2 users, then it makes sense to use an IBM APL2 font (e.g. APL2ital). However, unless that font is freely available, the author cannot expect non-APL2ers to be able to read the APL on that page, since they will have no access to the IBM APL2 fonts.

Here are a few possible solutions to this problem:

1. We get IBM to change the font encoding they use for APL2 for Windows to match that used by APL+WIN or Dyalog APL/W. (Best solution for APL in general, but unlikely to happen.)

2. We ask Adrian Smith to produce a new font APL2NET so we can all at least read APL2 web pages. (A poor solution for APL in general but easily achieved.)

3. I put up on the Vector web site a "converter" written in Perl, which given some text and the name of the encoding, converts the APL text into ASCII (using Jim Weigang's transliteration) on a new web page. Workspaces are available to convert ASCII back to APL from:
   ftp://archive.uwaterloo.ca/languages/apl/workspaces/aplascii.
   (A little "messy", but workable.)

4. I put up on the Vector web site a "converter" written in Perl, which given some text and the name of the to and from font encoding, converts the text (and <FONT FACE=font name list>) and re-displays the modified page. (Sounds OK but will be very hard to get working correctly.)

Web References

Jim Weigang, *APL-ASCII transliteration technique,*
http://www.chilton.com/~jimw

Ray Cannon, *Demo of APL fonts on the Web,*
http://www.vector.org.uk/aplfont.html

Adrian Smith, *APLNET font,* http://www.causeway.co.uk

News from Sustaining Members

Compiled by Gill Smith

MicroAPL

MicroAPL has released a major upgrade to APL.68000/X for the IBM RS/6000 and compatible systems under AIX 4.1. The new interpreter has been re-engineered to take full advantage of the PowerPC architecture of the latest RS/6000 systems, and users have reported some spectacular speed improvements across the whole range of APL operations. Versions are available for Motif as well as for traditional 'dumb terminal' operation.

Meanwhile, and contrary to what you might expect from the series of gloomy press reports about Apple Computer, our APL products for MacOS continue to sell well. We are considering what opportunities for APL might arise out of the new Rhapsody operating system which Apple have announced, and we shall be visiting California in May to find out more.

Finally, following an internal re-organization and the merging of some of our non-APL activities with another company, some familiar faces are now less often seen in the MicroAPL offices. In particular, David Eastwood, who co-founded MicroAPL in 1979 and who has made a sustained and varied contribution to MicroAPL's activities over more than fifteen years, is no longer involved on a day-to-day basis in the company. Happily, he remains a non-executive director and so we still have the benefit of his input.

MicroAPL Ltd. South Bank Technopark, 90 London Road, London, London SE1 6LN, UK

WWW: http://www.microapl.co.uk PHONE: +44 171 922 8866
E-Mail: MicroAPL@microapl.demon.co.uk FAX: +44 171 928 1006

Insight Systems / Adaytum Software

New Offices

After six years at Nordre Strandvej 119A, there was no further room for expansion. On March 1st, Insight Systems moved into a building of our own. Instead of being spread out down a long hall in separate units in an "office hotel", we can now all see each other (and the view of Øresund) through glass walls. Only Gitte has a proper "fürerbunker" where she and Helene can hide from the rest of us. We did not move far, we are in the same complex, our

address is now number 119C. Fortunately, we did not have to change telephone numbers yet again — we are still at +45 49 76 20 20 (FAX +45 49 76 20 30).

Adaytum Planning Product News

We have rolled out the 32-bit version of Adaytum planning (using Dyalog APL version 8.0.11 and Borland C version 5), and can report that the 32-bit environment is an improvement in every way. The system is faster, more reliable, and handles larger models than before. This is such a novel experience of a conversion to new base software that it deserves mention!

The focus of our product development efforts is now providing support for new front ends to Adaytum Planning, in particular Microsoft Excel and other environments which support OLE Automation. We are producing an OLE Automation Server for Adaytum Planning, which will make it possible for virtually any Windows developer to drive Adaytum Planning from within other applications.

Also scheduled for release after the summer are a Web Server, which will allow access to Adaytum Planning over the Inter/Intranet from web browsers and other web-enabled applications.

If you would like to know more about Adaytum Planning, visit our web site www.adaytum.com, and come to our annual European User Meeting, which will be held in Birmingham (UK) on April 23rd & 24th. The web site contains registration instructions.

Products: APL Technology Group

The focus of technology development this quarter is our OLE Automation Server, which makes it possible to make function calls to APL interpreters. This product is much less sophisticated than the OLE Server support provided by Dyadic, but has some important advantages. Our OLE Server acts as a client of our APL Pipes product, which means that it can act as a front end for all the APL systems which are supported by APL Pipes: Dyalog APL, APL*PLUS, SHARP APL and APL2 under any of Windows 3.1, Windows 95, Windows NT or Unix. We believe that it will still be some time before network OLE will support Unix in a way which will satisfy our typical customer organisations. We expect to have the OLE Automation Server available for beta testing in May. Write to info@insight.dk for more information.

HMW Trading Systems Ltd.

As we stated in the October 1996 issue of Vector, we are attempting to broaden our product range by extracting a number of modules from 4XTRA. This issue of Vector contains an article describing Change Control, the software portion of the Configuration Management environment that we have developed. We aim to have a stand-alone Dyalog APL/W version available during the second quarter of this year. This will include the Object Manager CASE tools that we use for development. We are also looking to publish an article in Vector about the modular prototyping methodology that we use with Change Control.

The next item on the agenda is to extract and package the multi-processor sequence scheduler. In simple terms, this is an APL module that is capable of running self-contained APL tasks on separate PCs that are on the same network. It takes into account the machine resources as well as any task interdependencies. Tasks are designed to fit within a standard framework that is capable of producing transaction logs containing as much or as little detail as required. An associated module allows remote monitoring of the tasks.

Dyadic Systems Limited

Dyadic is pleased to report that the production releases of Dyalog APL/W Version 7.3 and 8.1 are now available. The product is now distributed on CD, although diskette versions may be obtained on request. Dyadic extends its thanks to all those who took part in the Beta Test program.

The new versions were a month or so later than planned as a result of a rather late decision to include *syntax colouring* for the function editor. This feature, popularised by our competitors, has been extended to allow you to set up a range of different colour schemes and be able to switch quickly between them. For example, in addition to your regular multi-purpose colour scheme, you could have a scheme designed only to highlight global references and another designed specifically to make labels stand out. Dyalog APL's enhanced syntax colouring also identifies unbalanced control structures and unbalanced braces in dynamic functions.

Another minor but useful last-minute improvement is the support for wild-card searches in the Find/Replace dialog boxes.

Having completed Version 8.1 for Windows, effort is being directed towards porting this code to UNIX platforms. The production release of Dyalog APL/M will therefore be Version 8.1 compatible.

Further details concerning Version 8.1 and other implementations of Dyalog APL are available at the new-look Dyalog APL web site www.dyadic.com.

Causeway Graphical Systems Ltd

One of the great joys of being a software company is in seeing your work exploited and extended in ways you could never have devised yourself. *NewLeaf* has the rather (as we thought) esoteric capability of including any completed page as a scaled image into another page. Partly we put this in to be consistent with PostScript, which has a completely scale-independent language model, but mostly we put it in because recursion is fun. Jan Karman (who knows a thing or two about bookbinding) immediately spotted the possibility of saving several hours work on the photocopier by printing booklets (facing pages, double-sided) directly on his duplex laser. We fixed a couple of bugs in `leaf.Fetch` and now NewLeaf is shipped with a simple `Booklet` capability which anyone can use to save paper and time.

The "engine" of *CausewayPro* is now looking very solid, completely namespace aware and completely relocatable (you can call it whatever you like, and hide its namespace anywhere). It runs the existing class table roughly twice as fast as shareware Causeway, so you will be able to migrate existing applications along with any classes which you have written for them. New developments can be based around a completely new class library (accessible entirely through the Dyalog WS Explorer) which makes it remarkably easy to modify existing classes and add new controls of your own.

CausewayPro also extends the Causeway paradigm with **multiple active properties**. Currently, a Causeway object has a single 'data' property which is linked to an APL variable or expression; the property is refreshed whenever any of a list of 'watched' variables changes. Now you can have as many of these 'active properties' as you like, each with its own list of dependencies. For example a *Form* might have 'Position' and 'Size' both set by APL expressions such as `Δpos+10 10` and updated on a change to `Δpos` or `Δsize`. If you make several such forms (and change the expression to give them all slightly different calculations) they will follow each other about in a very amusing way.

Useless? Well so was the ability to include a page within a page within a page ... until someone found a use for it!

THE EDUCATION VECTOR

April 1997

Editor Ian Clark

This Education Vector has been reprinted from VECTOR Vol.13 No.4. VECTOR is the Quarterly Journal of the British APL Association. For more information about the British APL Association, please contact: Sylvia Camacho, 11 Auburn Road, Redland, BRISTOL, BS6 6LS. Tel: 0117-973 0036.

Contents

Editorial

Mathematics departments in England are to lose 38% of their funding over the next three years, according to a recent circular from the Higher Education Funding Council for England (HEFCE). Due to a reclassification for funding purposes to 'partly lab-based', £9.2M will go down to £5.7M in 1999. *New Scientist* magazine (1 March, 1997) quotes John Kingman, the vice chancellor of the University of Bristol as calling it 'potentially disastrous'.

So Mathematics, once the Queen of Sciences, is an expendable luxury in the new republic of intellectual England. At one time the provenance of Mathematics was any formalism whatever. Yet most subjects formalise nowadays. They do it idiosyncratically and independently, to their lasting loss. But Mathematics has clearly lost its monopoly in the matter, just as Latin lost it 50 years or so ago as an essential requirement to enter University. You don't need a mathematical training to program nowadays, nor to be an electrical engineer. Nor even to teach mathematics in schools, but that's just scandalous.

We used to tell ourselves 'scientists seem to find mathematics so hard, yet you can teach a mathematician anything.' Belied by the facts, I fear. When you are admitting well-qualified students to an MSc Mathematics course, and you find most of them pretty bad at applying their impressive formal knowledge to solving realistic problems, you feel they might as well have graduated in Egyptology. Mathematicians have largely brought it upon themselves.

In my old department, mathematicians were unwilling bedfellows with computer scientists (by then I was numbered among the latter). They used to teach subsidiary mathematics on our courses and fail our best students by making them jump through hoops. We simply wrote Mathematics out of our syllabus, because we felt we could teach whatever formal principles our students needed with far more relevance and impact. Our head of department was an applied mathematician who used to resist attempts to refer to his subject as 'Maths'. 'Would you like yours called "Phys" or "Chem"?' he'd retort. I used to annoy him by describing myself as an ex-mathematician. Eight years at the cutting edge of human-computer interaction had cast a new and critical light on my erstwhile colleagues. It began to look as if mathematicians cultivated their reputation by obfuscating their subject.

I now look back and see I was being perhaps a trifle unfair. It is truer to say that mathematicians quickly learn that they are NOT doing their careers a favour by NOT mystifying their subject a little. When you share staff rooms and access to funds with fellow academics who are not ashamed to flaunt their mathematical ignorance as a touchstone of quality ('He can't be all that good — why even I could understand what he said!') you learn that to make your subject comprehensible to a donkey is to risk submitting yourself to the judgement of an ass.

Nevertheless I feel I'm witnessing the mass-extinction of an entire ecology. When I started applying computers seriously in central government, we used to joke that simulations were what you did to amuse yourself while you thought up an analytical solution. Now whole skills will disappear as practitioners retire. The youngsters who replace them will never know a better way. In the words of Wilfrid Owen: 'now men will go content with what we spoiled.'

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

Tel: 01388 526803
Email: 100021.3073@compuserve.com

Introduction to Functional Programming in J (Part IV)

by Howard Peelle (hapeelle@educ.umass.edu)

This is the last part of a four-part article. Part I (in Vector 13.1) presented fundamentals of J in a framework of functional programming strategies, with examples expressly aimed toward developing an illustrative program to generate prime numbers. Parts II and III (in Vector 13.2 and 13.3) demonstrated how to develop the program directly, while characterizing a J programmer's thinking, and continued development of the same program with higher-level strategies. Part IV (here) presents alternative algorithms for iterative, recursive, and array-processing programs, then ends with meta-strategies for packaging and documentation.

Parts I - III presented these functional programming strategies:

0. Think Arrays
1. Select Primitive Functions
2. Select Primitive Operators
3. Use Parallel Processing
4. Name Utility Functions and Variables
5. Compose Functions (Define Programs)
6. Test and Debug Programs
7. Modularize Programs
8. Simplify, Optimize, and Generalize Programs

We continue with two final programming strategies for the same programming task of generating prime numbers.

9. Consider Alternative Algorithms

It is wise to consider alternative algorithms as a strategy for improving on a programming task — even after completing a working program. Developing different programs may lead to new or better ideas. In any case, the programs may be compared and the best one selected.

J offers iteration, recursion, and array processing — three possible approaches for developing programs. We will present programs for each of these approaches and compare them.

Array Processing

First, consider generating primes by array processing. (The following algorithm is an alternative to the one developed in Parts I - III.) Begin with a list of consecutive integers greater than 1 (using utility verb `To =. }. i.@>:` from Part II):

```
   p =: 2 To 13                          NB. p is integers 2 To 13
   p
2 3 4 5 6 7 8 9 10 11 12 13
```

Then create a table of products of all pairs of these integers:

```
   p */ p                                          NB. p Times-Table p
 4  6  8 10 12 14 16 18 20 22 24 26
 6  9 12 15 18 21 24 27 30 33 36 39
 8 12 16 20 24 28 32 36 40 44 48 52
10 15 20 25 30 35 40 45 50 55 60 65
12 18 24 30 36 42 48 54 60 66 72 78
14 21 28 35 42 49 56 63 70 77 84 91
16 24 32 40 48 56 64 72 80 88 96 104
18 27 36 45 54 63 72 81 90 99 108 117
20 30 40 50 60 70 80 90 100 110 120 130
22 33 44 55 66 77 88 99 110 121 132 143
24 36 48 60 72 84 96 108 120 132 144 156
26 39 52 65 78 91 104 117 130 143 156 169
```

Here the adverb / uses * (Times) to produce a new verb */ (called Times-Table when used with two inputs) which then multiplies every item in p (on the left) by every item in p (on the right) in pairs. Define a verb for this:

```
Composites =. ] */ ]                 NB. Composites is
                                     NB. Input Times-Table Input
                                     NB. where ] is the Input
```

`Composites p` produces the table above. To get the primes, remove the composites from the list p:

```
   p -. Composites p                 NB. p Less (Composites p)
2 3 5 7 11 13
```

The verb -. (called Less when used with two inputs) removes the items on the right from the items on the left. Now define the main program:

```
Primes =. ] -. Composites            NB. Primes is
                                     NB. Input Less Composites
```

An example of its use:

```
   Primes 2 To 13
2 3 5 7 11 13
```

Since excessive space is required to create the table of composites, this program can certainly be improved by truncating the inputs to */ in about half. First, define a utility verb to obtain the last item in an input list:

```
   Last =. {:                    NB. Last is Tail
```

And define a subprogram:

```
   Half =. 2: To <.@-:@Last      NB. Half is Two To
                                 NB. ((Floor Atop Halve)
                                 NB. Atop Last)
                                 NB. where <. is Floor
                                 NB. and -: is Halve
```

This generates integers from 2 to the integer part of half of the last item in the input list. For example:

```
   Half 2 To 13
2 3 4 5 6
```

Then revise program `Composites` accordingly:

```
   Composites =. Half */ Half    NB. Composites is
                                 NB. Half Times-Table Half
```

This computes the composites needed (with much less excess). For example:

```
   Composites 2 To 13
 4  6  8 10 12                  NB. 2 * 2 3 4 5 6
 6  9 12 15 18                  NB. 3 * 2 3 4 5 6
 8 12 16 20 24                  NB. 4 * 2 3 4 5 6
10 15 20 25 30                  NB. 5 * 2 3 4 5 6
12 18 24 30 36                  NB. 6 * 2 3 4 5 6
```

The main program works the same as before:

```
   Primes 2 To 13
2 3 5 7 11 13
```

This program is quite efficient for such small input lists. However, for longer lists, the space required to to create such a table of composites can push a microcomputer's limits. We can do somewhat better by truncating one side of the */ table with SquareRoot (%:) instead of Halve (-:). Thus:

```
   Root =. 2: To <.@%:@Last      NB. Root is Two To
                                 NB. ((Floor Atop SquareRoot)
                                 NB. Atop Last)

Composites =. Root */ Half       NB. Composites is
                                 NB. Root Times-Table Half
```

This uses less space and is faster. Let's obtain actual benchmarks. First, replace the subprograms with their primitives, using Fix (as shown in Part III):

```
   Primes =. Primes f.          NB. Primes Fixed in primitives
```

Then use the utility verbs `Time =. 6!:2` and `Space =. 7!:2` (introduced in Part III):

```
   Time 'Primes 2 To 1000'
0.389                           NB. Seconds (approximate)

   Space 'Primes 2 To 1000'
265896                          NB. Bytes
```

These benchmarks were obtained on a 486 PC running JFW (*J Freeware for Windows*) Release 3. Although this program is significantly (about 11 times) faster than the previous array-processing program (optimized in Part III), space is a growing problem.

Recursion

As an alternative, now consider a recursive approach. The overall structure of a recursive program in J is:

```
Program =. Recurse Else Default When Test
```

where `Recurse` is a composed verb which uses the defined `Program` itself (and other verbs); `Default` is a verb which returns a result for the degenerate case; and `Else` is a conjunction which ties the two verbs together. `When` is a conjunction which executes the first verb (`Recurse`) if `Test` returns 0 (false) or the next verb (`Default`) if `Test` returns 1 (true). This is J's version of a "case structure" (see [1] for details). First, be sure to define the utility conjunctions in order to use names for J primitives ` (called Tie) and `@.` (called Agenda):

```
Else =. `
When =. @.
```

Then the common introductory example of computing a factorial recursively can be defined as follows:

```
Fac =. (] * Fac@<:) Else 1: When (] = 1:)
            NB. Fac is ((Input Times (Fac Atop Decrement))
            NB. Else One) When (Input Equal One)
            NB. where verb 1: returns the constant 1
            NB. and Decrement (<:) subtracts one
```

In other words, for positive integer input n, Fac(n) is n*Fac(n-1) recursively until n=1, when it defaults to 1. For example:

```
   Fac 5                          NB. Factorial 5
120
```

This result is computed as follows: `Fac 5` is `5*Fac 4`. `Fac 4` is `4*Fac 3`. `Fac 3` is `3*Fac 2`. `Fac 2` is `2*Fac 1`. `Fac 1` is `1`. So, `Fac 2` is `2*1`. `Fac 3` is `3*2*1`. `Fac 4` is `4*3*2*1`. And `Fac 5` is `5*4*3*2*1` or `120`.

Incidentally, Factorial (`!`) is a primitive verb in J:

```
   !5                             NB. Factorial 5
120
```

Another classic example of (double) recursion is computing the nth number in the Fibonacci sequence (1 1 2 3 5 8 13 21 34 55 ...) defined in J as follows:

```
Fib =. (Fib@<: + Fib@<:@<:) Else 1: When (] < 3:)

                    NB. Fib is (((Fib Atop Decrement) Plus
                    NB. ((Fib Atop Decrement) Atop Decrement))
                    NB. Else One) When (Input LessThan Three)
                    NB. where verb 3: returns 3
```

For a positive integer input n, Fib(n) is Fib(n-1) plus Fib(n-2) recursively until n<3, when the default is 1. For example:

```
   Fib 10                    NB. The 10th Fibonacci number
55
```

Now let's return to the task of generating prime numbers and develop (top down) a recursive program using the well-known algorithm called "Sieve of Eratosthenes": given a list of consecutive integers beginning with 2, take the first number, remove all its multiples, and repeat, taking the next remaining number, removing its multiples, etc. Begin with definition of a verb, named `Primes`, which will call itself:

```
Primes =. (First , Primes@Sieve) Else ] When Done

                          NB. Primes is ((First Append
                          NB. (Primes Atop Sieve))
                          NB. Else Input) When Done
                          NB. where Append (,) joins items
```

For an input list of integers, `Primes` returns the first integer followed by the primes remaining after doing `Sieve`, recursively until `Done`, when the default is the (successively reduced) input list.

`First` is a utility verb to get the first item in an input list:

```
First =. {.                NB. First is Head
```

`Sieve` is defined as follows, using `If =. #~` and `Divides =. | = 0:` (subprograms from Part I):

```
Sieve =. ] If First Not@Divides ]

                           NB. Sieve is Input If
                           NB. (First (Not Atop Divides) Input)
```

`Not` is a utility verb named for the primitive verb Not (`-.`) which converts every 1 to 0 and every 0 to 1:

```
Not =. -.                  NB. Not is Not
```

In other words, `Sieve` selects from its input those integers which are not divisible by the first integer. For example:

```
   Sieve 2 To 13           NB. Sieve 2 3 4 5 6 7 8 9 10 11 12 13
3 5 7 9 11 13              NB. Not divisible by 2
```

Doing `Sieve` again on that result:

```
   Sieve 3 5 7 9 11 13
5 7 11 13                  NB. Not divisible by 3
```

Note that sieving twice is sufficient to get all primes up to 13: 2 (the first integer in the first Sieve), 3 (the first integer in the second Sieve), and 5 7 11 13 (the integers remaining). It is not necessary to do `Sieve` again if the first number remaining in the list exceeds the square root of the last number; that is, it is not possible to eliminate any more. This suggests how to define subprogram `Done` to stop the recursion:

```
Done =. First > %:@Last    NB. Done is First GreaterThan
                           NB. (SquareRoot Atop Last)
```

In other words, `Done` compares the first number in its input list with the square root of the last number and returns a 1 (true) if it is greater, otherwise a 0 (false). For example:

```
   Done 3 5 7 9 11 13      NB. Test after first recursion
0                          NB. 3 > %:13 is false

   Done 5 7 11 13          NB. Test after second recursion
1                          NB. 5 > %:13 is true
```

A simpler definition of `Done =. # = 0:` would stop the recursion when the list input to `Primes` becomes empty, but this is far less efficient — increasingly so for longer lists. Now execute the main program:

```
   Primes 2 To 13
2 3 5 7 11 13
```

Benchmarks:

```
   Primes =. Primes f.          NB. Primes Fixed in primitives

   Time 'Primes 2 To 1000'
0.097
   Space 'Primes 2 To 1000'
49492
```

This is much faster and more economical on space than the previous array-processing program, but still requires a stack of program calls which would betray its efficiency for longer input lists. (Also see [2] for an alternative recursive definition.)

Iteration

Iteration is a common approach. In J, there are several ways to apply a verb repeatedly. To begin with, iteration is implicit in "parallel processing"; that is, the same verb can be performed on all items in an array — without looping and counters — as shown in previous examples (from Part I) such as:

```
   3 | 9 10 11 12              NB. Remainders of a list
0 1 2 0                        NB. Result is a list

   1 2 3 4 + 9 10 11 12        NB. (1+9),(2+10),(3+11),(4+12)
10 12 14 16                    NB. Parallel sums
```

Another way to iterate in J uses the adverb / (Insert) which applies a given verb between items of an array (one input). (See Part I for introduction.) For example, +/ adds up a list of numbers:

```
   +/ 1 2 3 4 5 6 7            NB. 1+2+3+4+5+6+7
28
```

J also permits explicit iteration of a verb on its own results. A function to a power such as $f^4(x)$ or f(f(f(f(x)))) in conventional mathematical notation would be `f^:4 x` in J. The conjunction `^:` (called Power) requires two inputs: a verb (on the left) and the number of times to repeat it (on the right). For instance, consider iterating cosine:

```
Cos =. 2: o. ]                 NB. Cos is Two Circle Input
```

The primitive verb o. (called Circle when used with two inputs) contains a family of trigonometric functions (see [1]), including cosine. For example:

```
   Cos 1                          NB. Cosine of 1 radian
0.5403023
```

Iterate this on itself four times — Cos(Cos(Cos(Cos 1))):

```
   Cos^:(4) 1                     NB. Cos Power 4 starting at 1 radian
0.7934804
```

Or use a list to show the sequence of iterations:

```
   Cos^:(i.5) 1                   NB. Cos Power 0 1 2 3 4 on 1 radian
1 0.5403023 0.8575532 0.6542898 0.7934804
```

We could iterate further, seeking convergence to a fixed point, but J offers a symbol (_) called Infinity which will cause Power to iterate as many times as needed until the result doesn't change:

```
   Cos^:(_) 1                     NB. Cos Power Infinity on 1
0.7390851                         NB. Its fixed point
```

In effect, this solves the equation cos(x)=x (in conventional notation).

Now let's address the task of generating prime numbers iteratively, based on "Eratosthenes' Sieve" algorithm. Here we will use the Power conjunction (^:) with two verbs — where the verb on the right calculates how many times to repeat the verb on the left — to define the main program:

```
Primes =. (Sieve , First) ^: Repeat

                          NB. Primes is
                          NB. (Sieve Append First) Power Repeat
```

Here the verb on the left of ^: is a composition of three verbs called a "fork" (see Part I for introduction) which appends the first item in the input onto the end of the list resulting from doing subprogram Sieve. The definition of Sieve is the same as earlier (in Recursion):

```
   Sieve                          NB. Display program Sieve
] If First Not@Divides ]
```

Now we need to compute how many times to iterate, given any input list. So, define Repeat based on the (whole) number of primes needed to sieve the list up to the square root of the largest integer (given last) in the input list — when it's not possible to eliminate any more:

```
Repeat =. Round @ Nprimes @ %: @ Last
                          NB. Repeat is ((Round Atop Nprimes
                          NB. Atop SquareRoot) Atop Lat
                          NB. where Last =. {: (from earlier)
```

Utility program Round assures that the result is a whole number by adding 0.5 and then taking the integer part:

```
Round =. <. @ (0.5&+)  NB.Round is Floor Atop (0.5 With Plus)

   Round 2.4
2
   Round 2.6
3
```

Now we need to estimate the number of primes. First, consider the following simple program (based on the famous prime number theorem [3]):

```
Np =. <: % ^.          NB. Np is Decrement Divide NaturalLog
                       NB. where ^. is logarithm (base e)
```

Unfortunately, this underestimates (badly) the number of primes for inputs over 4. (See [3] for a better heuristic based on the celebrated Riemann zeta function which is too computationally intense for our purposes here.) Np can, however, be used in the following program to compute a good upper bound (together with Round) for inputs from 2 to at least 10 billion:

```
Nprimes =. (+ Np) @ <: @ Np   NB. Nprimes is ((Input Plus Np)
                              NB. Atop Decrement Atop Np
```

Examples of its use with Repeat:

```
   Repeat 2 To 13
2                          NB. 2 primes up to SquareRoot 13

   Repeat 2 To 25
3                          NB. 3 primes up to SquareRoot 25

   Repeat 2 To 1000
11                         NB. 11 primes up to SquareRoot 1000
                           NB. (2 3 5 7 11 13 17 19 23 29 31)
```

Examples of the main program:

```
   Primes 2 To 13
5 7 11 13 2 3

   Primes 2 To 25
7 11 13 17 19 23 2 3 5
```

Note that while the primes are not in the usual ascending order, the results reveal where sieving stopped. It is easy enough (although more expensive) to re-order the result using the utility verb `Sort =. /: ]` (from Part III):

```
   Sort Primes 2 To 25
2 3 5 7 11 13 17 19 23
```

Better yet, simply rotate the smaller primes on the end of the list to the front using the primitive verb `|.` (called Rotate). For example:

```
   _3 |. Primes 2 To 25          NB. Negative 3 Rotate Primes
2 3 5 7 11 13 17 19 23
```

Benchmarks for the `Primes` program (by itself) are as follows:

```
   Primes =. Primes f.           NB. Primes Fixed in primitives

   Time 'Primes 2 To 1000'
0.06

   Space 'Primes 2 To 1000'
47900
```

Compared with the previous optimized recursive `Primes` program for the same input, this program is more efficient in time and space.

Finally, here is an additional alternative which uses J's control structures to define a multi-line iterative program explicitly (using variables):

```
   Primes =. 3 : 0                    NB. Open definition
input =. y.                           NB. Initial input
list =. i.0                           NB. Initial list (empty)
while. Not Done input                 NB. While true (1)
   do. list =. list , First input     NB. Do reassign list
       input =. Sieve input           NB.    reassign input
  end. list , input                   NB. End result
)                                     NB. Close definition
```

This explicit definition is initiated by the `:` conjunction with `3` (indicating that the program will be a verb) and a `0` (indicating that it will be entered at the keyboard). (See [1] for other options.) The first line assigns a (local) variable — called `input` — to be the initial input (`y.`) given to the program. The second line assigns another (local) variable — called `list` — to be an empty list (see Part II for introduction). The next line begins a loop using a `while.` — `do.` control structure. (See [1] for other control structures.) The expression following the `while.` controls the execution of what follows `do.`. Here, the expression is `Not Done input` (see program definitions earlier). `Done input` produces a `0` and `Not Done input` becomes a `1` when the first number in `input` is not greater than the

square root of the last number. In other words, while the program is not done processing the input, it does the next two lines: reassign `list` to become the list of primes found so far with the first number in `input` appended on the end; and reassign `input` to be the result of `Sieve` (which removes all multiples of the first number in `input`). The `end.` indicates the end of the control structure, and the result is the following expression — here, the `list` of primes appended to those remaining in the `input`. A final right parenthesis closes the definition. Now it can be used:

```
   Primes 2 To 13
2 3 5 7 11 13
```

Benchmarks for this program reveal that it is slow for the example inputs here but increasingly fast for larger inputs and more economical on space than all other programs here:

```
   Primes =. Primes f.          NB. (No effect here)

   Time 'Primes 2 To 1000'
0.326
   Space 'Primes 2 To 1000'
17952
```

Comparisons

Here are benchmarks for all the optimized `Primes` programs presented above:

```
                          Time     Space (for Primes 2 To 1000)
Array-processing          0.389    265896
Recursive                 0.097     49492
Iterative                 0.061     47900
Iterative (multi-line)    0.326     17952
```

Comparing these programs in general terms, the iterative program is the fastest (for the limited example here); the multi-line iterative program is the most economical in space (and the fastest for larger inputs); the recursive program is competitive in speed (more so for larger inputs); and the array-processing program is simplest in its definition yet impractical.

10. Package and Document Programs

The last programming strategy is to package and document programs so that they may be used productively by users and/or other programmers. J offers a facility for recording a "script". This captures all J expressions, including (program) definitions, (variable) assignments and any comments. Fundamental procedures are outlined below. (More convenient procedures are available via menus under Windows.)

First, assign a script (here, an arbitrary name):

```
script =. 0 : 0                    NB. Spaces around : required
```

This uses the : conjunction to define a variable (indicated by the first 0) entered from the keyboard (indicated by the second 0). From here on, everything you enter will be recorded. In order to abbreviate this illustration here, suppose you enter several lines, as indicated below:

```
NB. J expressions
NB. ...
```

When you are finished, enter a right parenthesis:

```
)
```

Now you have assigned a script.

You may also want to save the script, as follows:

```
Save =. 1!:2 <                     NB. Define utility verb

script Save 'WORK'                 NB. Save script in 'WORK' file
```

When you wish to retrieve this file (here arbitrarily named 'WORK'), the script may be reconstructed and executed, as follows:

```
Load =. 0!:11 & <                  NB. Define utility verb

Load 'WORK'                        NB. Load 'WORK' file
NB. J expressions
NB. ...
```

Now all your J expressions are there — just as if you had entered them. (See [1] for more options.) You may want to check the names of programs, etc. Use another utility verb:

```
Names =. 4!:1                      NB. Names is 4 Foreign 1
```

For example, list the names of programs (verbs) here in this article (Part IV):

```
Names 3                            NB. Names of programs (boxed)
```

Composites	Cos	Divides	Done	Fac	Fib	First	Half	If	Last	Load

Names	Not	NPrimes	Primes	Repeat	Root	Save	Sieve	Sort	Space

Time	To

Other names are accessible similarly (see [1]). Other inputs for the Foreign conjunction (!:) manage files; communicate with the host system; report storage types, time, and space; display representations; control font and screen attributes; query and set global parameters; offer information for debugging; etc. (See [1])

Finally, suppose you want to save just the optimized Primes programs (and subprograms) with documentation (using NB.). You can cut and paste selected lines, include comments as needed and record it all in a script. For instance:

```
    script =. 0 : 0                    NB. Begin script
                                       NB. One input: Program (y)
                                       NB. Two inputs: (x) Program (y)
                                       NB. Utilities:
First =. {.                            NB. First item of y
Last =. {:                             NB. Last item of y
To =. ). i.@>:                         NB. Integers x to y

                                       NB. Array-processing approach:
Primes =. ] -. Composites              NB. Primes in list y
Composites =. Root */ Half             NB. Table of products
Half =. 2: To <.@-:@Last               NB. First half of list y
Root =. 2: To <.@%:@Last               NB. Squareroot half of list y

                                       NB. Recursive approach:
Primes=.(First,Primes@Sieve)`]@.Done      NB. Primes in list y
Sieve =. ] If First Not@Divides ]      NB. Items y notdivisible by y1
If =. #~                               NB. Select x where 1s in y
Not =. -.                              NB. Change 1s to 0s, 0s to 1s
Divides =. | = 0:                      NB. 1s where x divides y evenly
Done =. First > %:@Last                NB. 1 (true) when y1 > sqrt yn

                                       NB. Iterative approach:
Primes =. (Sieve,First)^:Repeat        NB. Primes in list y
Repeat =. >.@Nprimes@%:@Last           NB. Integer Nprimes to sqrt yn
Nprimes =. <: % (2: * 10&^.) + %       NB. Approx # of primes to y

                                       NB. Multi-line iteration:
Primes =. 3 : 0                        NB. Open definition
input =. y.                            NB. input is y
list =. i.0                            NB. list is empty
while. Not Done input                  NB. Loop while Not Done is 1
   do. list =. list,First input        NB. Append next prime
       input =. Sieve input            NB. Remove multiples of input1
  end. list,input                      NB. Result
)                                      NB. Close definition;End script
```

This script provides a succinct package of the programs developed here in this article (comments could be expanded, of course). It may be saved and retrieved (as shown above), and individual programs may be selected for use at any time.

Note, however, that J now provides a primitive verb p: (called Primes) which generates the nth prime number. This obviates the above programs. For example:

```
      p: i.6                          NB. First 6 primes
2 3 5 7 11 13
```

J also provides a companion primitive verb q: (called Factors) which produces prime factors. For example:

```
      q: 12
2 2 2 3

      q: 13
13
```

Conclusion

This (four-part) article has presented functional programming strategies in J:

0. Think Array
1. Select Primitive Functions
2. Select Primitive Operators
3. Use Parallel Processing
4. Name Utility Functions and Variables
5. Compose Functions (Define Programs)
6. Test and Debug Programs
7. Modularize Programs
8. Simplify, Optimize, Generalize Programs
9. Consider Alternative Algorithms
10. Package and Document Programs

These strategies can guide a programmer in tackling tasks well beyond the one illustrated here (generating prime numbers). A J programmer can thereby develop programs and subprograms rapidly in order to build a "toolbox" for solving problems, prototyping projects, and designing applications.

References

[1] Iverson, K. E. *J Introduction and Dictionary*, Iverson Software Inc., Toronto 1995

[2] Hui, Roger K.W. *An Exchange on Primes*, Vector, Vol.9, No.4, April 1993 pp.130-4

Howard A. Peelle
University of Massachusetts
Furcolo #10
Amherst, MA 01003 +1 (413) 545-1114
hapeelle@educ.umass.edu

APL Product Guide — Fully Revised

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.

Over the past few months we have attempted to contact all the vendors listed in the Vendor Address section of the guide. Only those who replied have been retained in this edition. If you would like your entry restored in time for APL97, please contact us as soon as possible – 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
- World Wide Web and FTP Sites

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 16" 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 16" 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)
Optima	IBM Compatible	poa	Complete networked or stand-alone solutions including configuration installation, maintenance and commissioning.

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.
Beautiful Systems	Dyalog APL/W for Windows	poa	US Distributor of Dyalog APL products from Dyadic.
	Dyalog APL for Unix	poa	See Dyadic listing for product details.
The Bloomsbury Software Company	APL+PC Version 11	260	Upgrade to version 11 gives free runtime (£120 from any version)
	APL+Win v1.8	1350	A 32-bit Windows-hosted interpreter that runs under all Windows platforms including Windows 95. **Note:** Any user purchasing APL+Win during 1996 will receive free updates to Vn 1.8 and Vn 2.0 (user to pay carriage)
	Upgrade to Version 1.8	540	From earlier versions of APL+Win. Free update to Version 2.0 (user to pay carriage)
	Migration to APL+Win	620	from APL*PLUS II versions 4/5. Free update to Version 2.0 (user to pay carriage)
		750	from earlier versions of APL*PLUS II
	APL+DOS	1300	APL*PLUS II DOS is renamed to APL+DOS.
	Migration to APL+DOS	620 / 390	from APL*PLUS/PC or APL*PLUS II

	APL*PLUS II for UNIX	poa	APL2000'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.
Dinosoft Oy	Dyalog APL/W for Windows	poa	Finnish distributor of Dyalog APL products.
	Dyalog APL for Unix	poa	See Dyadic's listing for product details.
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	ISO conforming interpreter. Supplied only with manual (see 'Other Products' for accompanying books).
	I-APL/BBC Master	8	As above
	I-APL/Archimedes	8	As above
	Strand Software Inc		Strand Software Inc has the sole selling rights to Iverson Software Inc products. I-APL stocks a few of these (mainly APLIWIN and the personal J products and books), but is no longer an agent.
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, Manuglstics, IBM
	APL2	poa	under: Windows, OS2 and Unix
Iverson Software Inc.	J on the Web online registration ...		
	J Educational Edition	$50	
	J Standard Edition	$150	
	J Professional	$325	
	Books and accessories (discounts for reg users)		
	J Dictionary	$50	
	J User Manual	$50	

	J Phrases	$40	
	J Primer	$40	
	Set of the above 4 books	$160	
	Concrete Math	$40	
	Fractals, Visualization & J	$50	
	Exploring Math	$50	
	J User Conference Proceedings	$35	
	Mugs, T-shirts, Mousepads	$10 each	
	APLIWIN	$70	For 386/PC under Windows 3.1
J Austria	J	poa	Distributor for Austria and Switzerland
	Dyalog APL	poa	Distributor
	Causeway Products	poa	Distributor
	Structural Analysis Software	poa	Complete package by IG Zenkner&Handel to perform structural analysis/engineering calculations. Also suitable for dynamic problems, e.g. earthquake simulation.
MasterWork Software	Manugistics Products and ISI	poa	New Zealand distributor
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	260	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	260	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	
Oasis Systems	Dyalog APL	poa	Dyadic Systems
	APL*PLUS	poa	Manugistics
	APL.68000	poa	MicroAPL Ltd
	APL2	poa	IBM
Optima	Dyalog APL/W	995	Fully fledged Windows development environment.
RE Time Tracker Oy	APL+PC (APL*PLUS/PC)	poa	Complete APL+ and Statgraphics product range and links to various 3rd party products.
	APL+DOS (APL*PLUS II)		
	APL+Win (APL*PLUS III), APL+Link		
	APL+UNIX		
	APL*PLUS Sharefile		
Soliton Associates	SHARP APL for MVS	poa	for IBM MVS mainframes
	SHARP APL for Unix	poa	for IBM RS/6000 and Sun SPARC

Strand Software	Canada		
	All APL*PLUS Products	poa	All APL*PLUS products including upgrades and educational.
	Dyadic and ISI products	poa	
	USA		
	Dyadic and ISI products	poa	
Uniware	APL+PC	poa	Uniware is the exclusive APL2000 distributor in France and also
	APL+Unix	poa	distributes in Switzerland and Belgium. Call for price quotes.
	APL+DOS	poa	
	APL+Win	poa	
	Dyalog APL/W	poa	French distributor for Dyalog

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.)
Adaytum Software	Adaytum Planning	poa	Full-featured Budgeting and Financial Planning system for medium to large enterprises.
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
Beautiful Systems	ASF_FILE	$399	Dyalog APL/W auxiliary processor for access to APL*PLUS/PC APL component files (*.ASF).
	NAT_FILE	$299	Dyalog APL/W auxiliary processor which emulates the APL*PLUS/PC quad-N native file subsystem for access to the DOS file system.
	DBF_FILE	$299	Dyalog APL/W auxiliary processor for efficient block mode access to dBASE format files. Designed to get large amounts of data in and out of dBASE. Not suited for random access to small amounts of data (it does not handle keys).
	SF_READ	poa	Dyalog APL/W functions to read APL*PLUS data objects of any type or structure from *.SF style component files created by APL*PLUS II or III.
The Bloomsbury Software Company (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.
Causeway	Causeway for Dyalog/W	35/$50	Registration for the Causeway platform supplied with Dyalog APL/W as shareware.
	Rain Graphics (shareware)	35/$50	Annual registration covers full on-line documentation (Windows Help) and handy-reference for the *Rain* graphics workspace supplied with Dyalog/W.

	RainPro Graphics (professional)	250	The ultimate graphics toolkit for the APL developer. Adds 3D charting capability, Web publishing and clipboard support to the shareware product. Charts can be included in *NewLeaf* reports. Functionally compatible across Dyalog/W and APL+Win.
	NewLeaf for Dyalog and +Win	400	Frame-based reporting tool with comprehensive table-generation and text-flow support. Offers multiple master-page capability, bitmap wrap-around and on-screen preview with pan and zoom. Fully supported on Dyalog/W and APL+Win (1.8 and above)
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. Handles large multi-dimensional tables, data analysis, EIS presentations; generates HTML and Latex output. Platforms: DOS, APL+II, Windows 3.1/95 for Dyalog APL, LAN support. Ideal for APL customisation, more than 100 installed.
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	SPARKS	poa	Educational simulation of electric circuit (for Apple Mac.)
	EPIDEMIC	poa	Educational simulation of spreading infection (for Apple Mac.)
	COINS	poa	Educational simulation (KS3) of coin-tossing experiment with simple stats (for Apple Mac.)
	FIBONA	poa	Educational simulation of Fibonacci's rabbits (for Apple 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!
IBM APL Products	A Graphical Statistical System	$250	for DOS, Product Number 5764-009
	(AGSS)	$500	for Workstations (OS/2, Aix, Solaris), Product Number 6764-092
		$2500	for CMS, Product Number 5764-011
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.
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.
RE Time Tracker Oy	UIT/W	poa	Comprehensive high-level Windows User Interface library for APL+Win and +II v 5.1. Comprehensive spreadsheets, replicated fields, special field types, etc. 16 and 32 bit versions available.
	AJGRAPH	poa	Graphpak-compatible 2D graphics package for +Win and +DOS. Includes multi-window support, print and metafile support. No DLLs required.
	ECCO PRO with APL	poa	Leading group and personal information management system with comprehensive customising. Supplied with sample +Win workspace to interface to ECCO databases via DDE.
	NEWT TCP/IP SDK with APL	poa	Lead TCP/IP SDK with interfaces to all protocols. Supplied on 3 CD ROMS together with a sample +Win workspace.
	DB+	poa	Database interface for APL+DOS under Windows. Allows combining character-based APL applications with ODBC-compliant databases such as Oracle and SQL-server..
Soliton Associates	LOGOS	poa	Application Development Environment
	MAILBOX	poa	Electronic Mail
	VIEWPOINT	poa	Report generator with interfaces to DB2 and MVS data
UNIWARE			
	APL+Win Monthly Training Program	$600	Download 50+ page document about APL+ programming each month. You also get one or more workspaces full of re-usable APL code and sometimes additional files or products.
	Advanced Windows Programming	...$95	200-page book plus companion disk on interfacing APL and Delphi. Contains full coverage of Delphi-2, +Win and Dyalog.
	DLL parser for +Win	$250	Parse any Visual Basic DLL declaration file into a set of quadNA definitions. Turn constants and structures into APL variables. Available for APL+Win and Dyalog/W.
	Delphi Forms Translator	$195	Design forms with Delphi and turn them automatically into APL programs which recreate the same form (+Win and Dyalog/W).
	APL+Link Pro	poa	ODBC interface for APL+Win
	SQAPL Pro	poa	ODBC interface for Dyalog APL/W
	RainPro	poa	Highly customisable 2D and 3D publication graphics for APL+Win and Dyalog APL/W
	NewLeaf	poa	Page layout and printing tools for APL+Win and Dyalog
	GraphX and ChartFX	poa	High-quality business graphics for APL+Win
	Formula One and Dyalog APL	$95	100-page book + companion disk on how to use the Formula One VBX with Dyalog APL/W
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	488pp. 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 AND DEVELOPMENT

COMPANY	PRODUCT	PRICES(£)	DETAILS
Adtee	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 Solutions Inc	Consultancy	poa	APL systems design, development, maintenance, documentation, testing and training. Providing APL solutions since 1969.
AUSCAN Software	Consultancy	poa	APL software development, training
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
Ray Cannon	Consultancy	poa	APL, C, Assembler, Windows, Graphics: PC and mainframe
Causeway	Consultancy and Training	poa	On-site training for Causeway, RainPro and NewLeaf. Customisation and enhancement to meet local needs. Code review and pre-implementation check of Causeway applications.
Paul Chapman	Consultancy	250-500	24-hour programmer: APL, Smalltalk, C; Windows front end design a speciality.
CODEWORK	Consultancy	poa	Development, maintenance, migration, documentation of APL applications. Speciality: info systems for top executives, internet applications.
Dinosoft Oy	Consultancy	poa	Specialised in very large databases.
Dyadic	Consultancy	poa	APL and Unix system design, consultancy, programming and training.
Evestic AB	Consultancy	poa	Excellent track record from 15+ years of APL applications in banking, insurance, and education services. All dialects, platforms and project phases. SQL expertise.
General Software	Consultancy	from 120	
Godin London Inc	Software Development	poa	We have applications in the food manufacturing field, travel agency and airline bookings field and in product lease management.
Entropy Software 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.
Hoekstra Systems Ltd	Consultancy	poa	APL consultancy, programming, etc. Also UNIX system administration
Michael Hughes	Consultancy	poa	Consultant with 10+ years experience with various APL interpreters and C.
IAC/Human Interfaces	Consultancy	poa	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	Moving applications between platforms. Client/server development. Multilingual user interface.
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.

JAD Software	Consultancy	poa	Systems design and development, project management, technical manuals, financial and actuarial expertise in APL.
Phil Last	Consultancy	poa	
Mackay Kinloch Ltd	Consultancy	170-320	Design, analysis and programming for banking, insurance, financial planning and modelling, corporate performance and legal reporting
MicroAPL	Consultancy	poa	Technical & applications consultancy.
Ellis Morgan	Consultancy	250-500	Business Forecasting & APL Systems.
Oasis Systems	Consultancy	poa	Expertise in APL system design, Project management, conversion, migration, tuning; for all APL versions (10+ years experience)
Optima	Consultancy	poa	A range of consultants specialising in all areas of pharmaceutical, industrial and financial systems with 5-15 yrs experience on both PC and mainframe.
RE Time Tracker Oy	Consultancy	poa	APL application conversions, APL Windows interfaces, APL to API-level interfacing to any system under Windows, TCP/IP network and database connectivity.
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
Shepp & Associates	Consultancy	poa	APL applications development and consulting, especially in the travel industry, especially on small computers. 25 years experience in APL programming.
Snake Island Research Inc	Consultancy	poa	APL interpreter and compiler enhancements, intrinsic functions, performance consulting. APL parallel compiler APEX is giving very good initial performance tests with convolution somewhat faster than FORTRAN.
Strand Software	Consultancy	poa	Advice on migrating to and from all flavours of APL and hardware platforms. Full-screen interface implementation, APL utilities, benchmarking, efficiency analysis, actuarial software, system development tools, valuation, pricing and modelling systems.
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.
Uniware	Consultancy	poa	A range of consultants, experts in Windows programming, with APL+Win and Dyalog APL/W. More than 100 major APL applications already developed. We all have additional expertise in Formula One and Delphi.
Stephen Wynn	Consultancy	poa	Most experience of financial planning, and mathematical areas: operational research, quality control, experimental design.

OTHER PRODUCTS

COMPANY	PRODUCT	PRICES(£)	DETAILS
Adfee	Employment	poa	Contractors and permanent employees
APL-385	Typefaces	poa	Variants of the APL2741 typeface available to specification.
Bloomsbury Software	Training	poa	Contact the company for details.
ComLog	Comic-Logger	$25.95+p&p	APL*PLUS II comic-book inventory system. Shareware version available on America OnLine.
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
	Programming in J	10	75pp by Ken Iverson
	Arithmetic	12	118pp by Ken Iverson
	Tangible math	8	36pp by Ken Iverson
	Sharp APL Reference Manual	42	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		
Oasis Systems	Training	poa	Introductory courses in APL Advanced courses for different APL versions
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.
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	DM50
Ass. Francophone pour la promotion d'APL	France	Les Nouvelles d'APL		FF350 (private) FF2800 (Company)
BACUS	Belgium	APL-CAM	Conferences & Seminars	£18 ($30)
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)
Rome/Italy SIG	Roma, Italy			
RusAPL	Moscow, Russia	APL Club	Seminars and Annual Meeting	100,000R (students 20,000)
SE APL Users Grp	Atlanta, Georgia	SEAPL Newsletter	Quarterly meetings	$10
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

ADDRESSES

ORGANISATION	CONTACT	ADDRESS, TELEPHONE, FAX, EMAIL etc.
Adaptable Systems	Lois & Richard Hill	49 First Street, Black Rock 3193, Australia. Tel: +61 3 9589 5578 Fax: +61 3 9589 3220 Email: adsys@ibm.net
Adaytum Software	Douglas Rowley	13 Great George Street, BRISTOL BS1 5RR UK Tel: 0117-921 5555
Adfee	Bernard Smoor	Dorpsstraat 50, 4128 BZ Lexmond, Netherlands. Tel +31 347 342 337 Fax: +31 347 342 342 Email: adfee@concepts.nl
Andrews	Dr Anne D Wilson	23, The Green, Acomb, YORK YO2 5LL, UK Tel: 01904-792670
APL-385	Adrian Smith	Brook House, Gilling East, York YO6 4JJ UK. Tel: 01439-788385 Fax: 01439-788194 Email: 100331.644@compuserve.com
APL Bay Area 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	Harald F. Nelson	c/o N-TECH, Siebenbrunnenfeldg. 4-6, A-1050 Wien, Austria. Tel: +43 1 5458063 Fax: +43 1 5458063-17
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 Solutions Inc	Eric Landau	1107 Dale Drive, Silver Spring, MD 20910-1607 USA Tel: +1 (301) 589-4621 Fax: +1 (301) 589-4618 Email: elandau@cais.com
Association Francophone pour la promotion d'APL	Ludmila Lemagnen	174 Boulevard de Charonne, F-75020 Paris, FRANCE Email: lemagnen@aol.com
AUSCAN Software Ltd	Richard Procter	8 Springmount Ave, Toronto, Ontario M6H 2Y4 Canada Tel: +1-416-651-4037 Email: rjp@interlog.com Web: http://www.interlog.com/~rjp/auscan/
BACUS	Joseph De Kerf	Rooinberg 72, B-2570 Duffel, Belgium. Tel: +32 15 31 47 24
Beautiful Systems, Inc.	Jim Goff	308 Old York Road, Suite 5, Jenkintown, PA 19046, USA Tel: +1 (215) 886-2636; Fax: +1 (215) 886-4888
Bloomsbury Software	Peter Day	3-6 Alfred Place, Bloomsbury, London WC1E 7EB UK. Tel: 0171-436 9481; Fax: 0171-436 0524; CompuServe: 100010,1467
Camacho	Anthony Camacho	11 Auburn Road, Redland, Bristol BS6 6LS UK. Tel: 0117-9730036. email: acamacho@cix.compulink.co.uk Reuternet (Sharp): ACAM
Ray Cannon		21 Woodbridge Rd, Blackwater, Camberley, Surrey GU17 0BS UK Tel: 01252-874697 Email: 100430.740@compuserve.com
Paul Chapman		51B Lambs Conduit Street, London WC1N 3NB UK. Tel: 0171-404 5401. Compuserve: 100343,3210
Causeway Graphical Systems Ltd	Adrian Smith,	5 The Maltings, Castlegate, MALTON, North Yorks YO17 0DP UK Tel: 01653-696760 Fax: 01653-697719 Compuserve: 100265,1564
Cinerea AB	Rolf Kornemark	Skyttegatan 25, S-193 00 Sigtuna, Sweden.
CODEWORK Italia srl	Mauro Guazzo	Corso Cairoli 32, 10123 Torino, Italy. Tel: +39 11 885168 Fax: +39 11 812 2652 Email: codework@inrete.it
ComLog Software	Jeff Pedneau	PO Box 5570, Derwood, MD 20855 USA Tel: +1 (301) 990-7063 Email: jeff@softmed.com
CPCUG	Lynne Sturtz	Capital PC User Group, 51 Monroe Street, Suite PE-2, Rockville, Maryland 20850-2421, USA. Tel: +1 (301) 762-9372 Fax: (301) 762-9375.
Danish User Group	Per Gjerløv	Email: gjerlov@ibm.net
Dinosoft Oy	Pertti Kalliojärvi	Lönnrotinkatu 21C, 00120 Helsinki, FINLAND. Tel: +358 9 70028820 Fax: +358 9 70028824 Email: dinosoft@dinosoft.fi
Dutch APL Association	Bernard Smoor (Sec)	Postbus 1341, 3430BH Nieuwegein, Netherlands. Tel: +31 347 342 337 Fax: +31 347 342 342
Dyadic Systems Ltd.	Peter Donnelly	Riverside View, Basing Road, Old Basing, Basingstoke, Hants RG24 0AL UK. Tel: 01256-811125 Fax: 01256-811130
Entropy Software Ltd	George MacLeod	Bertrum House, Ravens Lane, Berkhamsted, Herts, HP24 2DY UK Tel: 01442-878065 Email: 100412,1305@compuserve.com
Evestic AB	Olle Evero	Bertellusvagen 12A, S-146 38 Tullinge, Sweden Tel&Fax: +46 778 4410 Email: olle.evero@mailbox.swipnet.se
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 UK. Tel/fax: 0181-864 9537
Godin London Incorporated	Gaëtan Godin	12 Gerrard St., London, Ontario, Canada N6C 4C5 Tel: +1 (519) 679-8290 Fax: +1 (519) 438-6381 Email: info@godin.on.ca
H.M.W.Trading Systems Ltd		Hamilton House, 1 Temple Avenue, Victoria Embankment, London EC4Y 0HA UK. Tel: 0171-353 8900; Fax: 0171-353 3325; Email:100020.2632@ compuserve.com
Hoekstra Systems Ltd	Bob Hoekstra	5 Thorsden Court, Guildford Road, Woking, Surrey, GU22 7QS UK Tel: 01483-771028 Email: bob@khamsin.demon.co.uk
HRH Systems	Dick Holt	3802 N Richmond St, Suite 271, Arlington, VA 22207 USA Tel: +1 (703) 528-7624; Email: dick.holt@acm.org
Michael Hughes		28 Rushton Road, Wilbarston, Market Harborough, Leics. LE16 8QL UK. Tel: 01536-770998 Email: 101740.1203@compuserve.com
IAC/Human Interfaces	Ian A. Clark	9 Hill End, Frosterley, Bishop Auckland, Co. Durham DL13 2SX UK Tel: 01388-526803. Compuserve: 100021,3073
I-APL Ltd	Anthony Camacho (for queries, order forms)	11 Auburn Road, Redland, Bristol BS6 6LS UK. Tel: 0117-9760036 email: acamacho@cix.compulink.co.uk Reuternet (Sharp): ACAM
	J C Business Services (for pre-paid orders only)	56 The Crescent, Milton, Weston-super-Mare, Avon, BS22 8DU UK Tel: 01934-625181
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
INFOSTROY	Alexei Miroshnikov	3 S. Tulenin Lane, St. Petersburg 191186 Russia. Tel:+7 812 312-2673 Fax:+7 812 311-2184 Email:aim@infostroy.spb.su
Insight Systems ApS	Morten Kromberg	Nordre Strandvej 119C, DK-3150 Hellebæk, Denmark Tel:+45 49 76 20 20 Fax: +45 49 76 20 30 Email: info@insight.dk
Iverson Software Inc.	Eric Iverson	33 Major Street, Toronto, Ontario, Canada M5S 2K9 Tel: +1 (416) 925-6096; Fax: +1 (416) 488-7559 Email: info@jsoftware.com
J Austria	Joachim Hoffmann	Münzgrabenstr. 68, A-8010 Graz, Austria. Tel: +43 (0)316 814529 Fax: +43 (0)316 816683 Email: JoHo@magnet.at
JAD Software	David Crossley	580 Eyer Drive, #81 Pickering, Ontario, Canada L1W 3B7 Tel: +1 (905) 837-1895 Fax: +1 (905) 831-5172
Phil Last	Phil Last	146 Crossbrook Street, Cheshunt, Herts, EN8 8JY UK. Tel: 01992 633807
Mackay Kinloch Ltd	Alastair Kinloch	519 Webster's Land, Edinburgh EH1 2RX, Scotland, UK Tel/Fax/Answerphone: 0131 228 3580 Pager/Voicemail: 01426 98 3858 Email: 100010.33@compuserve.com
Mercia Software Ltd.	Gareth Brentnall	Holt Court North, Heneage Street West, Aston Science Park, Birmingham B7 4AX UK. Tel: 0121-359 5096. Fax: 0121-359 0375
MicroAPL Ltd.	Richard Nabavi	South Bank Technopark, 90 London Road, LONDON SE1 6LN UK Tel: 0171-922 8866 Fax: 0171-928 1006 Email: MicroAPL@microapl.demon.co.uk
Ellis Morgan	Ellis Morgan	Myrtle Farm, Winchester Road, Stroud, Petersfield, Hants GU32 3PE UK. Tel: 01730-263843 Email: Ellis@mrtlfrm.demon.co.uk
Oasis Systems B.V.	Theo Zwart, Louis Rijkse	Lekstraat 4, 3433 ZB Nieuwegein, Holland Tel: +31 30 60 66 336 Fax: +31 30 60 65 844 Email: oasisbv@pi.net or zwart@oasis.nl
Optima Systems Ltd	Paul Grosvenor	115 Brighton Road, Purley, Surrey CR8 4HE UK Tel: 0181-763 2490 Fax: 0181-763 2491 Email: 100551.1401@compuserve.com
Renaissance Data Systems	Ed Shaw	PO Box 421, Georgetown, CT 06982, USA. Tel: +1 (212) 864-3078
RE Time Tracker Oy	Richard Eller	Mikonkatu 8 A, 2.krs, PL 363, 00101 Helsinki, Finland. Tel: +358 9-621 3900 Fax: +358 9-621 3378 Email: re@rett.fi
The Rochester Group Inc.	Robert Pullman	50 S.Union St., Rochester NY 14607-1828, USA. Tel: not known. Fax: +1 (716) 271-1230
Rome/Italy SIG	Mario Sacco	Casella Postale 14343, 00100-Roma Trullo, Italy Email: marsac@vnet.ibm.com
RusAPL	Boris Makeev	box 971 (for Makeev), Dmitrovskoe Sh.,2, 127434, Moscow, Russia Tel/fax: +7 95 210-7783 Email: makeev@atom.ai.x-atom.net
SE APL Users Group	John Manges	413 Comanche Trail, Lawrenceville, GA 30244, USA Tel: +1 (770) 972-3755 Email: seapldoc@aol.com

Shepp & Associates LLC	Andrew Shepp	1312 Washington Avenue, 6th Floor St. Louis MO 63103, USA Tel: +1 (314) 621-3272 Fax: +1 (314) 621-4267 UK Address: Claridge House, 29 Barnes High St, London SW13 9LW Tel: 0181 8768666 Fax: 0181 8768660
Snake Island Research Inc	Bob Bernecky	18 Fifth Street, Ward's Island, Toronto, Ontario M5J 2B9 Canada Tel: +1 (416) 203-0854 Fax: +1 (416) 203-6999 Email: bernecky@eecg.toronto.edu
SOCAL (South California)	Roy Sykes Jr	Sykes Systems Inc, 4649 Willens Ave, Woodland Hills, CA 91364-3812 USA 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:sales@soliton.com
SovAPL	Alexander Skomorokhov	PO Box 5061, Obninsk-5, Kaluga Region, Russia Tel: +7(08439)31463 Email:askom2@kaluga.rosmail.com
Strand Software Inc	Anne Faust	19235 Covington Court, Shorewood MN 55331 USA Tel: +1 (612) 470-7345 Email: amfaust@aol.com
Rex Swain	Rex Swain	8 South Street, Washington, CT 06793 USA. Tel: +1 (860) 868-0131 Fax: +1 (860) 868-9970 Email: rhswain@acm.org
SwedAPL	Christer Ulfhielm	Novator Consulting Group AB, Svärdvägen 11C, S-182 33 Danderyd Sweden Tel: +46 8 622 63 50 Fax: +46 8 622 63 51 CServe: 100341,404
Swiss APL User Group		Swiss APL User Group, CH-3001, Bern 1, Switzerland Email: sl@ifi.unizh.ch
Sydney APLUG	Bob Bykerk	Bob Bykerk, 100 Lamrock Ave, Bondi, NSW 2026, Australia Tel:+61 2 9365 2520
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	Richard Procter	PO Box 55, Adelaide St. Post Office, Toronto Ontario M5C 2H8, Canada Email: rjp@interlog.com www.sigapl.mtnlake.com/sigapl/welcome.html
Uniware	Eric Lescasse	Tour Neptune, Cedex 20, 92086 Paris la Défénse 1, France. Tel: +33 (1) 47-78-78-00, Fax: +33 (1) 40-90-04-11 Email: lescasse@uniware.fr
Stephen Wynn		8 Clarence Gardens, Brighton, Sussex BN1 2EG Tel: 01273-327238 Email: centre@mistral.co.uk
Zark Incorporated	Gary A. Bergquist	23 Ketchbrook Lane, Ellington CT 06029, USA. Tel: +1 (860) 872-7805

WORLD WIDE WEB SITES

ORGANISATION	URL
AFAPL	www.ensmp.fr/~scherer/langlet/ (Journal available on line)
APL2000	www.APL2000.com
APL-385	www.demon.co.uk/apl385
AUSCAN	www.interlog.com/~rjp/auscan
Capital PC User Group	http://cpcug.org
Causeway	www.causeway.co.uk
CODEWORK	www.codework.de
COSY (Bob Armstrong)	www.cosy.com
Dinosoft Oy	http://yritys.kolumbus.fi/dinosoft
Dyadic Systems Ltd	www.dyadic.com
FinnAPL	http://personal.eunet.fi/pp/apl
Godin London Inc	www.godin.com
IBM APL2	www.torolab.ibm.com/ap/apl/apl2.html
Insight Systems ApS	www.insight.dk
Iverson Software Inc	www.jsoftware.com
Mackay Kinloch Ltd	ourworld.compuserve.com/homepages/Alastair_Kinloch
MicroAPL Ltd	www.microapl.co.uk
RE Time Tracker Oy	www.rett.fi
Shepp & Associates	www.digitravel.com
SigAPL	www.acm.org/sigapl
Rex Swain	www.pcnet.com/~rhswain
The APL Group Inc	www.aplgroup.com
Toronto SIG	www.sigapl.mtnlake.com/sigapl/welcome.html
Uniware	www.uniware.fr (www.uniware.fr/uk - English)
Jim Weigang	www.chilton.com/~jimw

FTP SITES

ORGANISATION	DOMAIN NAME
IBM APL2	ps.boulder.ibm.com/ps/products/apl2/
Toronto toolkit	see Toronto SIG home page
Waterloo Archive	archive.uwaterloo.ca/ftparch/languages/apl
APL-to-ASCII	archive.uwaterloo.ca/languages/apl/workspaces/aplascii

Getting Started on the Internet – a Guide for APL and J Enthusiasts

surveyed by Jon Sandles (jon_sandles@compuserve.com)

These days everybody has access to the Internet somehow or other – either at work or at home – and if you do not have that, you can pop down to your local Internet café and have a quick look there. But as APLers is there anything actually useful out there? The most obscure topics can all be found on the Internet, and APL and J are no exception. But just like everything else on the Internet there are few quality controls and no guarantee that what you see is true.

In this article I will be discussing a number of Internet resources and may even pass some comment as to the quality of the resource, but, and this is the crux of the Internet, you have to remember I am only commenting on right now, and by the time you read this article a number of the resources will have been modified, updated, improved or even deleted. Also a number of new ones may have appeared (although the rate of change for APL/J related resources is probably quite slow when compared to the Internet as a whole!). This is the magic of the Internet. The only true way to know what is good and bad is to go and check it out for yourself on a regular basis. This is where Vector comes in.

At the end of the product guide you will find a list of current APL Internet resources. This should be kept up to date and any new resources will be added. We will also try and indicate whether a resource has been changed significantly in the past three months in such a way that makes it a must for a revisit. To a certain extent we will be relying on you to keep us up to date with these changes.

So what APL is actually on the Internet?

- e-mail. The ability to communicate with APLers all over the world is one of the most important features of the Internet (for APLers anyhow – probably one of the least relevant for everyone else!). There are also a number of APL resources that involve the use of e-mail as their distribution medium.
- World Wide Web access. There is plenty of APL on the WWW. This is what I will look at in the most detail.
- FTP. This provides access to huge libraries of files for downloading onto your PC. There are a few specific file libraries dedicated to APL.

- USENET/news. There is a dedicated USENET news group for APL called comp.lang.apl.

- Added value areas of some providers. Compuserve provides a portable programming forum for languages that can easily be moved across different platforms and APL is often discussed here. Compuserve also provides an APL2 forum.

Getting Started - Internet Providers and Cost

The basic requirements for communicating with the Internet are a computer with a modem and a phone link. A modem will probably give you either 28,800 bps or 14,400 bps access. Both of these speeds are acceptable - although if you are going to be doing a lot of work on the Internet then go for the fastest modem possible. An even faster option is ISDN access although this is still prohibitively expensive for most people to make it worth while. Once you have got the kit at your end you can't just 'dial the Internet' - you need to get an account with an Internet provider - who in return for taking money out of your bank account on a monthly basis will give you a phone number you can ring and an account and password that will give you access to the Internet. They will also give you lots of software to do all the exciting things that you want to do. Internet providers are all much of a muchness in terms of how much they charge per month for their service. There are several other factors that are normally much more important in deciding which provider to choose.

- The phone numbers they provide for their service need to be as cheap i.e. as local as possible. The majority of charges incurred in using the Internet will be via the phone - not to the Internet providers themselves. The telecommunications industry is the real winner in the Internet boom. Some providers will only cover certain areas - and if you are going to need to access the Internet from outside those areas you will incur heavy phone charges making long distance calls to access your e-mail. The major providers Compuserve and AOL provide access numbers in most of the major countries at local rate. This is a major advantage if your work involves longish spells of travel abroad.

- The length of time you can access the Internet before additional charges are incurred. Most providers will allow an amount of time on the Internet for free and then charge by the hour for additional usage in any month. If you are going to use the Internet a lot then you will probably want a flat rate that gives you unlimited access per month. If you are only going to use it 'every now and again' then as a guide 5 hours should be enough to get you going.

- Added Value. How many e-mail accounts will you get - normally one, but sometimes they offer more which might come in handy for the wife and kids! e-mail address format - what will your e-mail address format look like? For

years Compuserve users have had to put up with unwieldy addresses like 102337.1231@compuserve.com. This is impossible to remember both for you and your friends. Something like jon_sandles@compuserve.com makes a lot more sense. (Compuserve have recently introduced this non-numeric style of addressing – thank goodness!)

- Web page space – some providers will give you an amount of space on their servers for designing Web pages. This is useful to get you started in HTML (the language used to write Web pages), but if you want to seriously use the Web for commercial purposes check the following: how much space you get; how much does it cost to get more space; can you use CGI programs? Can you use your own registered domain name (this is a bit like the mail addresses – http://ourworld.compuserve.com/homepages/jon_sandles is not quite as memorable as http://www.sandles.com (to find out more about registering domains check out http://www.isi.edu/div7/iana/).

- In addition to this some providers (as a result of their non Internet based past) provide their own set of online services/forums and chat areas – Compuserve, AOL and the Microsoft Network stand out in this respect. I can only comment on Compuserve, whose forums are by far the fastest way to find out technical information or get shareware programs. Compuserve are able to organize their forums into areas and groups making navigation easy. There is also a global file find facility. Also if Compuserve crashes during a file download, you can restart it from the point where it got to. You do not have to start again. But, Compuserve's forums are beginning to get a bit left behind as many corporations find it too hard to maintain both an Internet and Forum presence. The Internet presence is bound to win out in the end.

Most Internet providers appear to be quite helpful and there do not seem to be quite the same number of pitfalls associated with choosing an Internet provider as there are say with purchasing a mobile phone. Once you have decided all this and got yourself connected you will want to start exploring the Internet for real.

e-mail

Most people have seen or used e-mail so I will not dwell for too long on this topic. The first thing you will want to do is talk to some APLers via e-mail (of course). How do you find out their e-mail address if you have not already got it ? There is no organized telephone directory although SIGAPL have an APL 'white pages' which is nearly as good (available off the SIGAPL Web pages). If they are not in there, then try mailing someone who is who might know them – they will be only too pleased to pass the information on. You can also subscribe to mailing lists via e-mail which automatically send e-mails to a group of interested people whenever anything is posted to them. An example of this is Tony Chan's DyalogAPL mail server.

If you wish to subscribe, send

To: t-chan@u-aizu.ac.jp
Subject: DyalogAPL

and

Subscribe YourFirstName YourLastName

in the main body of the mail. This can be a nice way of getting a problem solved - although I have to say that this particular mail-server seems to be somewhat under-utilized! In fact all the resources (WWW, FTP etc.) are accessible via e-mail. (This technique of getting other Internet resources via e-mail was developed for people in developing countries where they only had e-mail gateways on the Internet. Hence, you are discouraged from overloading these resources if you are in a country that has a gateway to the resource you are looking for.)

Browsing the Web

To get on the Web (WWW) you will need a Web Browser. Fortunately these are easy to get hold of as virtually every magazine carries a copy of the two major browsers on the market. There is little to choose between Netscape 3.0 and Microsoft IE3. If you are using Windows 95 and regularly purchase Microsoft software you probably have already had IE3 installed on the back of some other product.

When you first use your Web Browser you will almost certainly attempt what is known as the 'random surf'. You will go to your Internet provider's home page, because that is the only Web address you know and you will try clicking on various 'links' which take you to other pages. You will find loads of information. You will read loads of information. None of it will make much sense. After about half an hour your head will hurt. Your wife and kids will be screaming to use the phone. You will go to bed and wonder what all the fuss was about. Welcome to the WWW. Fortunately there is a better and quicker way of finding information.

Web Search Engines

Web Search engines provide a means of searching the Web for pages that contain certain keywords. You can even combine keywords using simple Boolean logic to narrow the search further. Some search engines even provide ratings for the quality of information within a page, whereas others simply provide the most recently changed ones first - indeed there is no guarantee that what you get back

will still be present on the Web and you will get the infamous '404 File Not Found' error message. (This is known as a 'dead link'. Basically the Web page you tried to find existed at some point in the past and the search engine duly catalogued it. It has since been deleted - but because search engines are very bad at finding out when links have been deleted they are still reported as being in existence.)

I'm not going to tell you which Search engines are best or which to use, as like everything else on the Web they are changing all the time. Check out the latest reviews in specialist Internet magazines or PC magazines [1]. A good place to start is **Alta Vista [www.altavista.com]** or **Infoseek Ultra [ultra.infoseek.com]** - both of which seem to have the largest databases of Web Sites.

It should be noted that search engines are not exhaustive in their searches. They will frequently have not heard about new pages and some find pages that others do not. If you do not find what you are looking for then go to a page that is similar or as close as possible and look for links from that. Most definitive pages on a particular topic will link up to other similar pages. (If they did not they would not be definitive!)

Remember to add any pages that you are likely to want revisit to your bookmark list in your browser. This will save you having to find the page all over again.

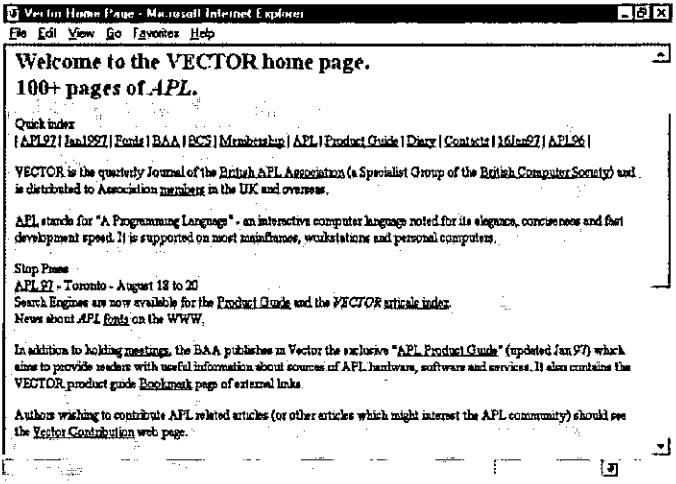
Using search engines is a secret art and each one has its own features and quirks. Make sure you check out the advanced options and read through the help as this could reduce your on-line time exponentially.

The Minder Robot

Once you have found a bunch of Web Pages that you like you will want to revisit them whenever they have new information on them. Some Web Pages change all the time. Some do not change for years and years. You do not want to have to keep checking "just in case" they have changed. The minder robot is a useful tool that allows you to tag pages that you like, and it will notify you via e-mail when they have changed or been updated in any way. You can try this out on many of the Vector Pages which have a special minder robot section. (At the moment it checks them about once a week.) This is a free resource (nothing to do with the BAA/Vector) and can be used on any Web pages.

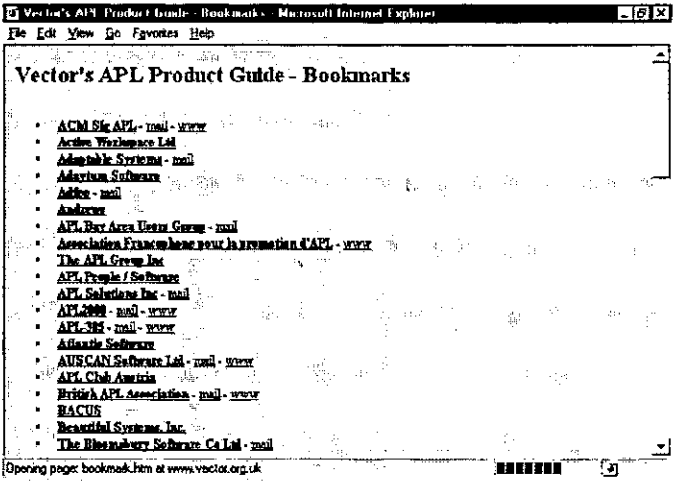
APL Pages on the WWW

Search engines seem to be particularly bad at finding APL resources. I tried both the major engines mentioned above and got a real mixture of pages back. Few of them had anything to do with APL. Hence the recommended starting point for any APL related search would (at the moment) be a visit to any of the following three sites.



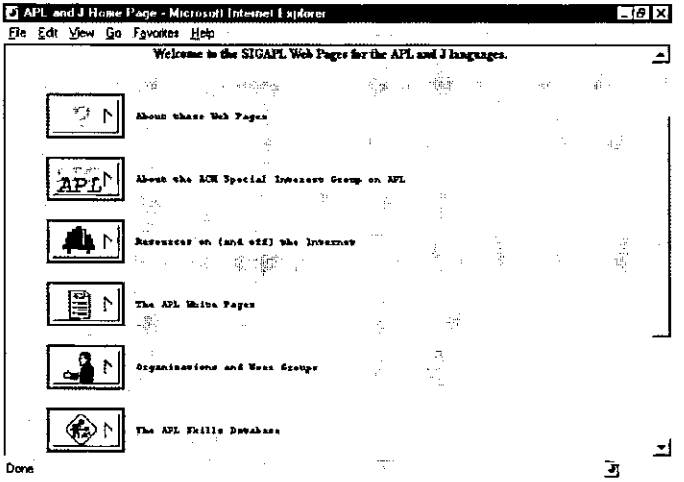
Vector (www.vector.org.uk)

The Vector pages are well laid out with a quick index at the top of most pages which should provide easy navigation. The best list of APL links is found by going to the Bookmark page ....



Here you can see a number of organizations and groups who have registered either e-mail or Web addresses and these are then posted as links.

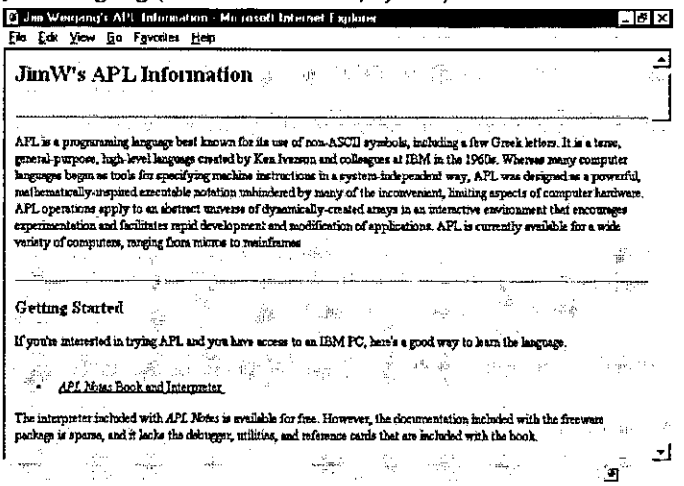
SIGAPL (www.acm.org/sigapl)



The SIGAPL pages have an interesting graphical twist to them, but they load quite fast and have good links to other pages. The APL white pages is a list of e-mail addresses (much like the Vector bookmark page) and the Resources page leads to lots of other links.

The skills database is a good place to go if you are looking for jobs, or indeed if you have jobs to offer.

Jim Weigang (www.chilton.com/~jimw)



Jim's pages are widely regarded as being the definitive APL 'home' pages, partly because of the length of time they have been around, but mainly because they have a nice informal, but informative layout.

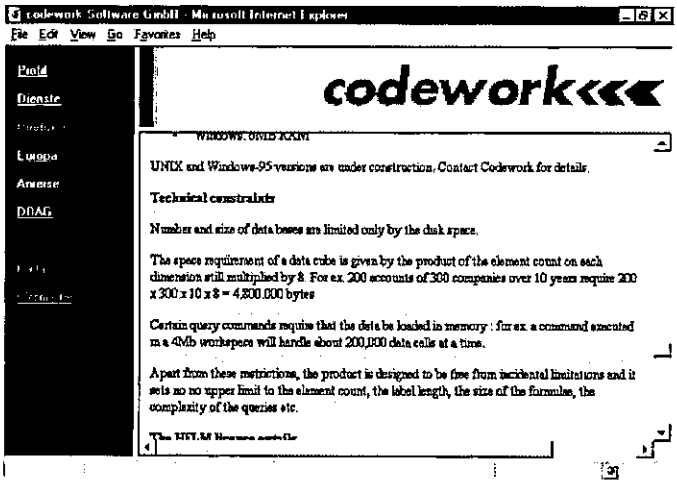
APL Characters on the WWW

Much has been said on this topic, but little agreement is ever made. To see an overview of approaches all the above three Web sites carry advice on how to tackle the problem. Ray Cannon's article in this Vector reports on recent advances in the HTML specification (FONT FACE =) and suggests ways we can use them to our advantage (see page 14).

Some Random APL Sites Reviewed

There are at least 20 areas of web pages which are directly focused on different aspects of APL. Of these about 40% are commercial organizations that currently use APL as a problem solving/ programming tool. The next 40% are APL vendors and consultants who are selling APL products. The remaining 20% are APL groups like SIGAPL and the Vector pages. I will thus focus on a random sample of pages in these proportions.

Codework (www.codework.de/cwk_en.html)

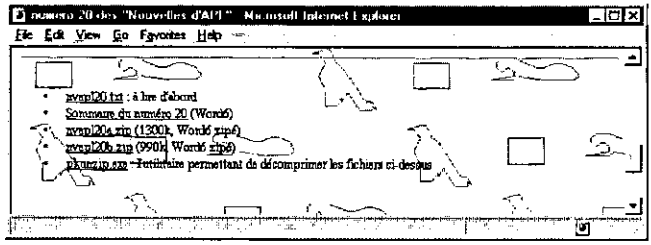


The Codework pages are a good example of a company that is using APL in one of its products. I do not think APL is ever mentioned in any of these pages. They are well laid out and available in three different languages.

You have to dig quite deep in the pages to discover that it is HELM, their OLAP product that is APL based. To an APLer the signs should be obvious ....

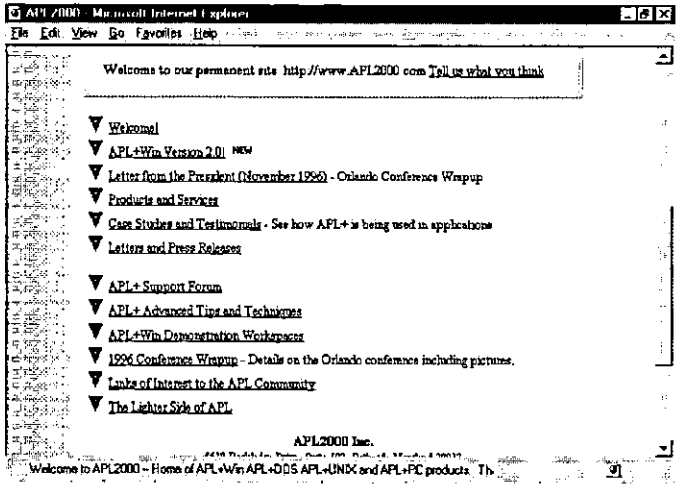
Association Francophone pour la Promotion d'APL (www.ensmp.fr/~scherer/langlet/)

This page acts as an overview of the latest contents of Les Nouvelles d'APL - the French APL Association's tri-monthly journal. The pages are in French (as is the magazine) and you can download the whole thing in Word 6 format if you want. This is a genuinely useful free resource with some very good articles often aimed at the professional developer (you might need a translator of course!)



APL2000 (www.APL2000.com)

Here is the index page of the APL2000 Web presence. Notice the ticker line which rolls past at the bottom.

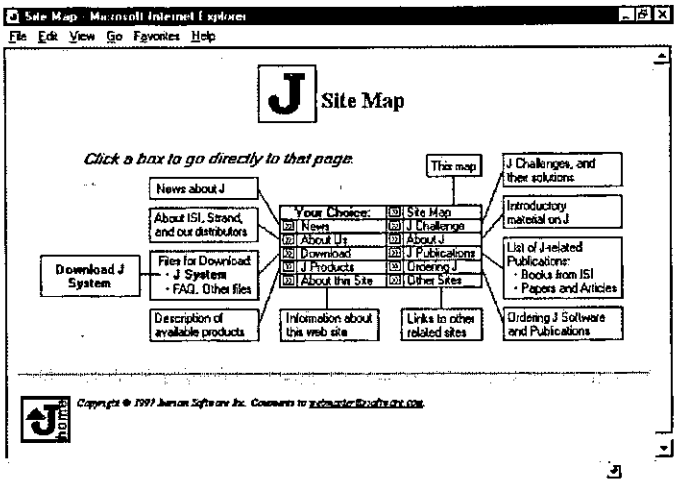


As you can see this provides an excellent overview of the APL2000 company and product portfolio. They also provide an interesting links page.

Mackay Kinloch Ltd.
(http://ourworld.compuserve.com/homepages/Alastair_Kinloch/)
Mackay Kinloch provide computer software services, including APL systems analysis, design, programming and maintenance.

These pages are concise and simple and are a good place to visit if you want to know what to aim for when knocking your first Web pages together. But, there is little dynamic content, and the last changed flag on the main page (February 96) indicates that it won't be worth visiting again for a while.

Strand Software Inc. (www.jsoftware.com)
These are basically the home pages of the J language. J, as you probably know by now, has been blessed with the ASCII character set and hence fits into the Web paradigm very well. J code can just be presented on screen with no playing about with browser settings and no funny transliteration schemes. They also have an excellent site map for easy navigation.



Off-line Browsing

If you are going to be doing some intensive Internet research over a reasonable period of time you may want to invest in an off-line browsing tool. These allow you to build collections of web pages off-line and whenever you log on you can tell them to check out certain pages to see if they have changed. You can then decide (off-line) which of these pages you want it to retrieve to be viewed at your

leisure (off-line). Hence most of the browsing/decision making is done when you are not wasting money on the phone. These tools will only make much difference if you are needing to browse quite extensive sets of pages - and it is also important to know when the information is updated.

One of the best off-line browsing tools currently on the market is Quarter-deck WebCompass [www.qdeck.com]. You can maintain active areas of interest and build a schedule of updates, of when to check for updated information.

Usenet News Groups

Your internet service provider will give you access to a Usenet news feed. It is possible that they will remove certain groups for your own protection. It is very unlikely that they will prevent you from accessing comp.lang.apl. It is possible that the frequency with which your news groups get updated, and hence the timeliness of the information, is unreliable. You might want to try another news feed; there are a few public ones about. Try news.sprynet.com as a useful alternative. You might want to run two newsfeeds side by side for a while to see which one works the best. If you are a professional programmer I would advise perusing all the comp.lang.* news groups every now and again as it can be very illuminating to see what discussions are going on within each language's area. Some corporations provide their own public news servers. An excellent and useful example of this is the news server at msnews.microsoft.com.

It is often commented on that comp.lang.apl contains too many discussions on J. In reality, comp.lang.apl is an open and non-moderated resource so it contains whatever we want it to. If there is not enough APL then that is because there are not enough APLers contributing to it - not because there are too many J programmers contributing too it - comp.lang.apl can hardly be said to be a high volume/over busy news group (try looking at comp.lang.c++).

FTP

FTP gives you an interface to libraries of files that you can download from the internet. No fancy graphics here, just a list of directories and files, and if you click on a file it will end up on your hard disk about three hours later. Great stuff!

These libraries are mostly maintained by universities and there is often very little to tell you what a file contains or what it does. Always look out for a readme file either in the base directory (in which case it will describe the contents of all the files in the whole library) or in the directory where you are looking.

The current IBM APL2 beta program has been delivering the new software for testing via ftp. You simply receive an e-mail with the new file names and with a supplied account and password, then you log on and download them from an IBM ftp server.

Downloading Files from the Internet

One final caution before you all go Internet crazy. Most of the quality information available on the Internet appears with equal timeliness within the computer press or via journals like Vector. The software you can download that is 'free' is mostly shareware so you should be paying for it anyhow. Once you register shareware you will start receiving disks through the post just like normal. It is normally a lot quicker to go and buy the latest computer magazine with a free CD ROM on the front and get the latest free shareware off that, than to waste half a day attempting to download it off the Internet. Downloads are expensive - they are slow - they tie up your time, they tie up a phone line and they frequently crash at the last minute. Remember to check for viruses, and try to remember not to use downloaded data files that may contain embedded macros (Word/Excel documents).

Summary

If you are an APL professional and you do not have access to the Internet make sure you get it. There are too many useful resources like Uniware's APL+Win training program that are only available via the Internet.

If you are an APL manager then make sure your staff have access to the Internet, but also make sure you install a firewall that is capable of controlling and monitoring Internet access.

If you are going to put files on the Internet, make sure they are virus checked by an up to date virus checker.

References

[1] *Catching Sites*, PC MAGAZINE, February 1997, Page 109.

[2] *Know your net*, Digital Corporation. ED-H010B-95

"Advanced Windows Programming With APL and Delphi" by Eric Lescasse

reviewed by Jonathan Barman

Overview

If you have just bought Delphi and want to start using it immediately with APL then this is the book for you. It is written in the style of a recipe book with very simple step by step examples showing you how to create a dish of Delphi DLLs which can be called from APL.

Every example gives APL code for Dyalog APL/W Version 8 and for APL+Win Version 2.0 (Beta). The primary platform is Delphi 2 under Windows 95 running 32-bit code, but there are examples of 16-bit code using Delphi 1 so that older versions of APL can be used under Windows 3.x.

The coverage is quite complete. Pretty well all of the fundamental ways in which you can make good use of a DLL are touched on in the examples. However, there is little that shows you how to progress from the very simple to the more complex routines that would be actually required in practice.

As all but one of the examples are so very simple, the word "Advanced" in the title is rather misleading. A more informative title would have been "A Delphi Primer for APL Programmers".

First Impressions

Last year I bought Delphi as it appeared to be a well designed GUI product which would complement the power of APL, or for that matter J. However, I never seemed to have time to get to grips with using it in the way I had intended. So, as soon as the book arrived I enthusiastically started on Chapter 1 which shows how you can use the powerful Delphi form designer to create a form which can then be translated into either a Dyalog APL or APL+Win function. I started to faithfully follow the step by step procedure:

1. start Dyalog APL/W 8
2. load the workspace which contains the `ParseDFM` and `ParseDFMObject` functions

Step 1 – easy. Step 2 – what workspace? I loaded the only workspace, *BOOK*, on the companion diskette and there was no sign of any *Parse* functions. Now, I have to confess that I had skipped the introduction, which is usually just blah, blah, blah, to get my hands on the interesting bits. So I retraced my steps to carefully read every word in case there was some warning that you had to pay extra for this particular goodie. But no, the only direct reference was a little footnote stating that the parser tool is currently only available for Dyalog APL/W 7 and 8. Consulting the Vector Product Guide disclosed that UNIWARE sells the Delphi Forms Translator for $195.

First impressions were therefore of utter disgust. Here was a tutorial where you cannot even get started without spending a whole lot more money. However, having skipped over Chapter 1, the remainder of the book contains a lot of useful information on how to make good use of the combined power of APL and Delphi. If Eric had placed the chapter at the end of the book and had had the grace to issue a warning that you had to purchase the software to make it work, then first impressions would have been very different.

This review therefore starts at Chapter 2 and leaves a description of the Forms Translator to the end.

Non-GUI DLLs with Delphi

Eric describes in detail, and with great clarity, how to create the simplest possible DLL in Delphi and then shows you how you can call it both from Dyalog APL and from APL+Win.

Creating a Delphi DLL is simply a matter of selecting File/New then clicking on the DLL Object icon. Delphi creates a template containing the basic code required for a DLL. All you have to do is to type in your code. The following is Eric's first example where I only had to type the middle 6 lines defining the function `intArg` and declare it in the `exports` clause:

```
library firstdll;

uses
  SysUtils,
  Classes;

function intArg(X: Integer): Integer; stdcall;
begin
  intArg:=X div 2;
end;

exports
  intArg index 1;
```

```
begin
end.
```

In both APLs the DLL is accessed using ⎕NA. The DLL took about 5 seconds to complete, but I spent ages fiddling about trying to get ⎕NA to work. In Dyalog APL/W Eric gives the following example:

```
⎕NA'I4 T:\DYALOG95\BOOK\FIRSTDLL.DLL.C32|intArg I4'
```

but because my DLL was in a different directory I typed:

```
⎕NA'I4 c:\vector\review\delphi\firstdll.dll.c32|intArg I4'
```

and got a *DOMAIN ERROR*. In the end I found that the file name can be in upper or lower case, but the C32, which indicates the 32 bit C calling convention, must be upper case.

The example for APL+Win was:

```
'DLL I4 ← FIRSTDLL.intArg(I4)' ⎕na 'intArg'
```

which I typed in faithfully but got a 0 return code indicating that ⎕NA had not worked. Examination of the help for ⎕NA disclosed an example with no spaces around the assignment arrow. When I entered the left argument as *'DLL I4←FIRSTDLL.intArg(I4)'* then everything worked fine. APL+Win does not seem to mind if the key word *DLL* or the DLL name are in upper or lower case. The difference could be due to the my using Version 1.8 rather than Version 2.0 (Beta) which was used for the book, but I am a little suspicious as the book is full of minor typing errors.

Once these little hurdles were surmounted everything went very smoothly. I was grateful to have Eric's book in front of me assuring me that everything ought to work and all that might be wrong was a typing error.

The chapter contains detailed step by step descriptions of every possible way in which scalars and arrays of various types can be passed to, and updated by, a DLL. Everything is simply and clearly explained and is a useful tutorial for a beginner in Delphi as well as understanding how to link APL with DLLs. I was interested in the technique of switching off Delphi's array bound checking so that a pointer to an array can be passed to the DLL. Obviously this could result in some spectacular errors as in APL one gets to rely on *INDEX ERROR* coming up when one makes a stupid mistake. Particular care will be needed in any Delphi DLL code when switching off the equivalent facility.

The chapter ends with a description of a utility DLL containing functions which carry out tasks that are difficult or impossible to carry out in APL. There is a function to create a new directory which would be useful for Dyalog APL/W 8 users as this facility does not seem to be directly available, unlike APL+Win which has ⎕*MKDIR*. Another function returns the day of week of a date. This is reasonably easy in APL, and most people create APL utilities which carry out date manipulation tasks, but why bother to create an APL utility when Delphi has the function built in?

GUI DLLs with Delphi

The tutorial part of this chapter shows you how to create a Delphi form in a DLL and call it from APL. The example form is populated with a set of controls which require all the data types discussed in the previous chapter to be passed to and from the form, so there is an easy learning curve based on previous experience.

The controls on the example form include an edit field and a memo field, both of which are populated from APL character vectors. As these controls allow you to type in any amount of text there is a problem should you type in any additional characters which would extend the length of the original APL vector. For example, if you pass in a pointer to `V` where `⍴V` is `10`, then you cannot enter any more than 10 characters. The 11th character will overwrite memory in APL and will normally cause a GPF.

I was not very happy with Eric's suggested solution to the expanding character vector problem. He suggests that you pad out your character vector with spaces and insert and Ascii zero (`⎕AV[⎕IO]`) at the end of the text that is to appear in the control. For example, you could create a vector with `A←256↑'Text', ⎕AV[⎕IO]` then this allows the user to type a total of 255 characters, allowing room for a trailing `⎕AV[⎕IO]`. The main problem with this solution is that the Delphi code does not know the overall length of your character vector and therefore cannot stop the user from typing too many characters by setting the `MaxLength` property of the control. The technique used by the ODBC driver interfaces seems to be much cleaner. There are usually three pointers: a pointer to the character string for the initial text, a pointer to the buffer to be filled by the function, and a pointer to the length of the buffer. On return the pointer to the buffer length contains the actual length of the string placed in the buffer.

The chapter ends with some rather nice examples of simple forms that could be called from APL. All these forms must, or course, be modal. APL has to be in control and must wait until the user has completed entering data on the form

before continuing. There is no way at present for Delphi to be in control and to call APL as a DLL; you need J for that.

Creating a Resources DLL

Not only can DLLs contain functions and forms but they can also contain bitmaps. As Eric points out you can easily accumulate a couple of hundred small bitmaps to display on the toolbars of an APL application. Storing the bitmaps in a DLL rather than as separate files on disk makes a lot of sense.

Bitmaps, icons, cursors, fonts and text constants can all be stored in a Resource file which can then be included in your DLL. The Borland Resource Workshop is a powerful tool for maintaining Resource files and is very simple to use. There is also an Image Editor which is not nearly as powerful and is that much more difficult to use. The detailed instructions on how to use both systems seem a little over the top, but do get you going quickly.

The really useful information as far as I was concerned was the instructions on what you have to do to load a bitmap into APL+Win. This requires the use of `⎕WCALL` to run the Windows API functions `LoadLibrary` and `LoadBitmap`. It was also useful to be informed of the size of the bitmap required for toolbar buttons, which is slightly different in Dyalog APL/W and APL+Win.

The chapter ends with an example of a little APL form displaying a toolbar (or toolbox) with bitmaps loaded from the DLL.

Other Delphi Techniques

The main topic of this chapter is an explanation of how to display a splash screen while APL is loading.

The trick is to run a tiny Delphi program which immediately displays a splash screen. It then reads a `.INI` file and parses the command line string to see which APL workspace is to be loaded. The Delphi program issues the appropriate call to load the workspace and then goes into a wait loop checking to see when APL is ready. As soon as the wait loop terminates the Delphi program closes down.

I was impressed with this application. It appeared to be the live code for Uniware Toolkit/W and must be very near to code which is actually delivered to customers. This is the one topic in the book which merits the word 'Advanced' in the title.

The other example in the chapter, how to play a .WAV sound file in Delphi, seems just too simple this far into the book. Also, you can perfectly well play a .WAV file directly from APL.

Accessing Databases via Delphi

When I first got my copy of Delphi I went through almost exactly the same steps that Eric describes in this chapter. The first step is to use the Delphi wizard to crate a simple database application linked to one of the sample Paradox databases. Having seen what the wizard produces one can carry out much the same task using the underlying controls and facilities. Finally, you can start using ODBC to access data in a Microsoft Access database.

The Delphi Database Form expert makes things look incredibly simple. Unfortunately, life is not like that and when you start having to get to grips with ODBC, tedious little problems tend to cause considerable angst. At least having a worked example in front of you makes the initial learning phase easier. For example, although I was familiar with adding an ODBC data source, it took me some time to work out how to use the Borland Data Engine (BDE) Configuration Utility. Eric tells you what to do in a step by step example but does not tell you why the steps are necessary. I still find the BDE Configuration a bit of a mystery, but so far the steps Eric explains are all that I have needed. However, I have not actually implemented an ODBC application in Delphi. When I do I feel sure that I will have to spend quite some time to really get to grips with the many powerful facilities which appear to be available.

The Delphi Forms to APL Translator

Having had my gripe about the lack of the translator function let's get back to explaining what Chapter 1 is all about.

Eric feels that the forms designers in both Dyalog APL and APL+Win are a pale shadow in comparison with the Delphi forms designer. He therefore likes to initially design his forms using Delphi and then translate the design into APL. I am not entirely sure that Dyalog APL users would agree with Eric on this score. The *wdesign* workspace, version 4.2 (Beta), is a big improvement on the old versions. However, APL+Win users would probably agree with Eric, as the]*wed* facilities in APL+Win are really rather thin. In either case I really cannot see Causeway enthusiasts bothering as the facilities are reasonable and there are considerable advantages in the integrated environment.

A brief description is in order as you might find it worthwhile considering using the translator that Uniware offer, or even consider building one of your own.

Delphi stores the information about a form and its controls in a `.dfm` file. You can view the contents of the file in a Delphi edit window, but not, unfortunately in an ordinary text editor as you can with VB. I created a very small example to give you an idea of what is going on. Of course, not having the *ParseDFM* function I have had to type in the supposed APL output by hand, so please forgive any typing errors.

The following is the display of the `.dfm` file for a Delphi form with just 4 controls:

```
object frmTest: TfrmTest
  Left = 200
  Top = 108
  Width = 207
  Height = 110
  Caption = 'Test'
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  PixelsPerInch = 96
  TextHeight = 13
  object lblDesc: TLabel
    Left = 8
    Top = 16
    Width = 53
    Height = 13
    Caption = 'Description'
  end
  object txtDesc: TEdit
    Left = 72
    Top = 16
    Width = 113
    Height = 21
    TabOrder = 0
  end
  object cmdOK: TButton
    Left = 48
    Top = 48
    Width = 49
    Height = 25
    Caption = '&OK'
    TabOrder = 1
  end
  object cmdClose: TButton
    Left = 112
    Top = 48
    Width = 49
    Height = 25
    Caption = '&Close'
    TabOrder = 2
  end
end
```

All the controls are located on the form and there are no controls within controls. Normally there would be many controls within controls, for example, a *File* menu bar control will usually contain *New, Open* and *Exit* menu controls.

The whole of the displayed `.dfm` text is copied to the clipboard. Running the *ParseDFM* function in Dyalog APL/W would produce something like the following:

```
[0]   R←(FORM)Test A;B;C;D;E;F;G;H;I;J;K;L;FNT_1;⎕IO;⎕ML
[1]
[2]   ⎕IO←1 ⋄ ⎕ML←2
[3]   ⍎(0=⎕NC'FORM')/'FORM←''#.FRM_form'''
[4]   FORM←'#.',(2×'#.'=2↑FORM)↓FORM←FORM←' '
[5]   'FNT_1'⎕WC'Font'('Size' 15)('PName' 'MS Sans Serif')
[6]         ⍝ →(⎕nc FORM)⍴z
[7]   (FORM)⎕WC'Form'('Coord' 'Pixel')('Accelerator' 27 0)
       ('Event' 1001 1)('3D' 'Dialog')('Border' 2)('Posn' 108 200)
       ('Size' 110 207) ('Caption' 'Test')('Font' 'FNT_1')
[8]   (FORM,'.lblDesc')⎕WC'Label'('Posn' 16 8)('Size' 13 53)
       ('Caption' 'Description')
[9]   (FORM,'.txtDesc')⎕WC'Edit'('Posn' 16 72)('Size' 21 113)
[10]  (FORM,'.cmdOK')⎕WC'Button'('Posn' 48 48)('Size' 25 49)
       ('Caption' '&OK')
[11]  (FORM,'.cmdClose')⎕WC'Button'('Posn' 48 112)
       'Size' 25 49)('Caption' '&Close')
[12]  ⍎FORM,'.FORM←FORM'
[13]  ⎕SE.encaps
[14] z:
[15]  FORM ⎕WS'Visible' 1
[16]        ⍝ ⎕nq(FORM,'.___')'GotFocus'
[17]  R←3>(⎕DQ FORM),3⍴¯32768
[18]  →0
[19]
[20]  ∇
[21]  A←Close                 ⍝ ⍎FORM
[22]  ⎕NQ FORM 1001 ¯32768
[23]
[24]  ∇
[25]  Select;A                ⍝ ⍎FORM
[26]  ⎕NQ FORM 1001 ¯32768
[27]
[28]  ∇
[29]  Select A;B;C;D;E;F;G;H;I;J;K;L ⍝ ⍎FORM,'.cmdOK'
[30]  ⎕←A
[31]
[32]  ∇
[33]  Select A;B;C;D;E;F;G;H;I;J;K;L ⍝ ⍎FORM,'.cmdClose'
[34]  ⎕←A
```

The *⎕SE.encaps* function, on line [13], is a utility which fixes the dummy callback functions which are defined after the →0 on line [18]. The code for *⎕SE.encaps* function is listed as a footnote in the chapter on Resource files, but does not appear to be included on the companion diskette.

In APL+Win *ParseDFM* would produce:

```
∇ A Test B;⎕wself
[1]
[2]   :select B
[3]   :case ''
[4]       'Test'⎕wi 'Delete'
[5]       ⎕wself←'Test'⎕wi'New' 'Form' 'Close'
[6]           ⎕wi 'scale' 5
[7]           ⎕wi 'border' 3 16 32
[8]           ⎕wi 'where'(PixelstoLC 108 200 110 207)
[9]           ⎕wi 'caption' 'Test'
[10]
[11]      ⎕wself←'Test.lblDesc'⎕wi 'New' 'Label'
[12]          ⎕wi 'where' 16 8 13 53
[13]          ⎕wi 'caption' 'Description'
[14]
[15]      ⎕wself←'Test.txtDesc'⎕wi 'New' 'Edit'
[16]          ⎕wi 'where' 16 72 21 113
[17]
[18]      ⎕wself←'Test.cmdOK'⎕wi 'New' 'Button'
[19]          ⎕wi 'onClick' 'Test''cmdOK'''
[20]          ⎕wi 'where' 48 48 25 49
[21]          ⎕wi 'caption' '&OK'
[22]
[23]      ⎕wself←'Test.cmdClose'⎕wi 'New' 'Button'
[24]          ⎕wi 'onClick' 'Test''cmdClose'''
[25]          ⎕wi 'where' 48 112 25 49
[26]          ⎕wi 'caption' '&Close'
[27]
[28]      0 0⍴'Test'⎕wi'Wait'
[29]
[30]  :case 'cmdOK'
[31]
[32]  :case 'cmdClose'
[33]
[34]  :end
```

I could not find the *PixelstoLC* function on the companion diskette, but eventually realised that it converts pixels to characters using:

```
R←A÷(⍴A)⍴2⍴'#'⎕wi'units'.
```

The *onClick* events were not properly defined in the book so I have substituted the sort of thing that should have been produced by the *ParseDFM* function.

From the above it should be easy to see how the Delphi definitions have been converted to APL. Writing your own translator would be tedious but not, in my opinion, very difficult. The main question is whether you really feel that it is worthwhile.

Conclusion

At $90 this is an expensive little book. It is fractionally bigger than Vector and has 196 pages. There are a large number of screen pictures and the examples are quite repetitive, so the actual content is relatively low.

I cannot make up my mind if I would have bought this book at a UK cost of something like £60. I have certainly enjoyed working through the examples and have learned a fair amount. If I was writing a large APL application and was under some time pressure then I would definitely buy the book as it would save some 2 to 3 days' work. However, I am not currently writing an APL application and am not under any time pressure so the cost seems to be just too much.

Another way to justify a high price for a book is if it is going to be used as a reference manual. The examples are so very simple that having worked though them once there would be little reason to go back and carry out any further study. On the other hand, if Delphi was only used very occasionally then you might want to use the book as a reference manual to refresh you memory on how to program in Delphi.

Thanks

Many thanks to Dyadic Systems Ltd and Bloomsbury Software Ltd who kindly let me have review copies of Dyalog APL/W 8 and APL+Win to help in writing this review.

Review of NewLeaf from Causeway Graphical Systems, Version 1.2 available for Dyalog APL/W and APL+Win

by Robert Byers (adapted from Gimme Arrays, January 1997)

What NewLeaf is not

Sorry, Toronto Maple Leaf hockey fans, this "NewLeaf" is not the team's saviour to lead the team from out of the basement and into the playoffs.

What then is NewLeaf?

NewLeaf is a clever piece of software that greatly simplifies report writing in APL. With NewLeaf you can produce desktop-publishing results and even prepare output ready for the Internet.

NewLeaf is available in Dyalog APL/W and APL+Win versions. I use NewLeaf with Dyalog APL/W versions 7.2.7 and 8.0.11 for Windows 3.1 and Windows95 respectively. NewLeaf has been available in beta since the spring of 1996 from the Causeway Web site. It was officially released last August and an update was made available last November. I tested NewLeaf on a variety of personal computers. At the office, I use APL/W in a WIN-OS/2 session on a Pentium Pro-200 running the new, voice recognition and Java-enabled OS/2, version 4.0 (an impressive new release of this under-appreciated O/S). At home, I have a 486-33 running Windows95 and a 386-25 PC running OS/2 3.0. As expected, NewLeaf runs noticeably better on faster processors. NewLeaf on my 386 machine is usable but a little slow.

NewLeaf is a product from Causeway Graphical Systems Ltd. The two men behind Causeway are Adrian Smith and Duncan Pearson. Last spring Adrian spoke to the Toronto APLSIG about NewLeaf at a meeting which I attended.

NewLeaf ships as a single disk with three files. Those files include a workspace, a new APL font and an excellent help file. You also get a 100 page manual co-written by Adrian and Marc Griffiths. NewLeaf sells for £400. Updates and bug fixes are available from the Causeway Web site.

As we all know, printing in APL has never been the language's greatest feature. The lack of easy, good reporting features has not helped APL. Dyalog's APL/W offers a GUI Printer object that can produce good-looking output but it requires a fair bit of programming. Before NewLeaf, I would almost always bypass printing a report directly in APL. Instead, I copied my output to the clipboard or DDE out my variables to something like Lotus for final presentation and printing. NewLeaf offers an APL solution.

In Dyalog APL, NewLeaf takes advantages of "namespaces" which are unique to APL/W. Namespaces allow you to store groups of functions, variables and other objects in separate sub-workspaces All the NewLeaf code is contained in these namespaces. You can simply copy in the entire workspace or just the individual namespaces as required. Output from NewLeaf can be printed directly to your printer or you can save it in a workspace, component file, Postscript format, Encapsulated Postscript format or as a Windows metafile. Alternatively, output can be copied to the Windows clipboard as a bitmap or metafile and pasted into another application.

Instructions are included on how to convert your output into HTML format for Internet publishing. A separate HTML namespace is included but is only experimental at this time. If you have Adobe's *Distiller* software you can use the output from NewLeaf to create a PDF file (Portable Document Format) that can be viewed with Adobe Acrobat readers.

NewLeaf is described as a frame-based publishing tool. Everything revolves around what you put into a frame. Every page contains a least one frame. You can create as many frames on a page as you like. Once you have established a page layout you then place or flow text as paragraphs (ie. nested arrays) into the frame. Besides flowing in text you can insert tables, Rain graphs, fixed text and bitmaps. Under program control you can direct specific APL variables into a frame.

Tables are easily handled in NewLeaf. NewLeaf offers two table functions. The first one, `leaf.table.List` handles simple columnar reports. If you want more control over formatting and cell wrapping you use the `leaf.table.Spread` function. You have control over grid lines and horizontal and vertical rule lines. For tables with numbers you can format each column with a left argument to function `leaf.table.Qfmt` that is the same specification used with `⎕FMT`.

As stated earlier, the quality of the output is excellent. NewLeaf was designed to work with text, tables and *Rain* graphs. You can quite easily produce short or long reports. You can also design your output to look like letters, invoices,

reports, multi-column newsletter styles and even books. The APL newsletters produced by other APL SIGS like the Toronto APLSIG "Gimme Arrays", a three column newsletter, can now be produced in NewLeaf.

To produce a report you write a function containing lines of code referencing the appropriate NewLeaf functions. NewLeaf functions set properties that control the look of your output. Properties not referenced will assume their default settings. To produce a simple report, here is a sample set of code which would produce an attractive report with a heading, some text paragraphs and a table:

```
leaf.Use ''                                  ⍝ Start with default page layout
leaf.Style 'Heading'                         ⍝ Take the supplied style
leaf.Place 'Here is the Heading'             ⍝ Assume it will not wrap
leaf.PopStyle                                ⍝ Revert to previous style
leaf.Flow (text)(text)(more text)            ⍝ Wrap paragraphs of text
leaf.table.Set ('Coltitles' 'text' 'text')   ⍝ Set up table header
leaf.table.Spread matrix                     ⍝ Do the report
PG←leaf.Close                                ⍝ Saves Postscript code to variable PG
PostScrp.View PG                             ⍝ Preview the result
```

As you can see from the above example this is how NewLeaf essentially works. For those not familiar with APL/W the period you see in "`leaf.`" is how APL/W namespaces are referenced within the workspace. NewLeaf offers a host of properties you can set. Setting properties can either be done by direct reference to individual functions or you can group properties by using the `leaf.Set` function. If you have "paragraphs" or table formats that you wish to re-use you can save these customized styles and call them when you need them. You can save your own paragraph styles.

NewLeaf does not offer an integrated GUI interface but it does include some GUI tools. To start with, you can create customized page layouts with the page layout designer (`Layout.Design`). This tool assists you in defining and placing the frames on a page. You can save these layouts as a variable and re-use them later. This designer is quite powerful and can do most of your report design. In addition to creating frames, the designer has features for aligning frames properly and for defining other page properties such as the page size, orientation, page numbering, placeholder etc.

Another NewLeaf GUI tool is the Viewer. This function allows you to view the report on your screen. The viewer allows the user to print, zoom in and out or scroll through multi-paged reports. This tool is very useful and displays your output in an attractive and professional style. The viewer can be easily integrated into your application as a view before you print facility.

A third GUI tool, still under development, is a debug feature that shows the creation of your report as each NewLeaf statement is executed. I found this tool very helpful in seeing the build- up of your report. It is especially useful when something unexpected happens in your report.

A key feature of NewLeaf is the ability to include *Rain* graphs. *Rain* is separate Causeway product, written again by Adrian, that produces high-quality business graphics. A shareware version ships with APL/W. The latest version has a similar namespace design to NewLeaf. To get the documentation or to upgrade to a more powerful version, called "*RainPro*", you need to purchase it separately. I will be reviewing RainPro next issue. For now, any graph produced by Rain or RainPro can be inserted into a NewLeaf frame. NewLeaf will automatically fit the graph into the frame you have defined for it.

NewLeaf can handle small and large volume reporting. By default, NewLeaf will hold the report as a variable in your workspace. This is fine when the report is relatively small but it is not very efficient when the report gets larger. The solution is to hold the report's pages as a component in a component file. You can also specify the report to be spooled to a file which is outside the workspace. The viewer tool can still access the resulting report.

NewLeaf vs other Reporting Tools

Both APL/W and APL+Win can support VBX and OCX add-ons. Therefore they both can use reporting packages written for Visual Basic such as Crystal Reports, ReportSmith and others. These packages offer more features than NewLeaf but they are more difficult to use because of the lack of documentation for the APL programmer. NewLeaf has the advantage of being written in APL and therefore uses APL concepts (i.e. nested arrays, matrices etc.).

Conclusion

If you purchase NewLeaf, I recommend you work through the tutorials contained in the manual and examine the sample functions that ship with NewLeaf. The help file is also an excellent reference.

Error checking could use some improvement. Sometimes an error occurs (not often). When it does, you a get a deeply nested series of functions displayed where the error occurs. As you get more familiar with NewLeaf you get better at deciphering the error. This can be frustrating if you are a new user. Sometimes you have to experiment a bit to get around errors. Sometimes the error occurs is because you are setting a property in the wrong order.

I would like to see more extensions that follow the Dyalog GUI so that NewLeaf essentially allows you to do WYSIWYG output. Multi-level Row titles for example are now a feature of the APL/W grid object. Adrian has already included basic HTML support. I look forward to seeing this feature enhanced further. Another suggestion relates to table formatting. I would like to be able to control individual row as well as column formatting. In fact individual cell attribute specification would be a useful extension. I know Adrian plans to address these issues and add other enhancements with future releases. He requests and is very responsive to suggestions.

Overall, I rate this product an 8.0 (out of 10). There is no doubt it fills a valuable role and void in APL programming. I have not covered all the functionality already in NewLeaf and more is sure to come. I can easily recommend this one for anyone wanting to do some serious report writing in APL.

More information is available about NewLeaf and other Causeway products by visiting their Web site at:

http://www.causeway.co.uk
E-mail: causeway@compuserve.com

Controlling APL Software Releases

Marc Griffiths, Chris Hogan, John Jacob, Phil Last
(HMW: 100020.2632@compuserve.com)

Developing variants of a software product for different customers is never the easiest of jobs. And when development takes place in three different locations, none of which is a customer site, you can imagine the difficulties that exist in terms of co-ordinating system development, and the subsequent assembly and testing of software releases. This was the situation that HMW Trading Systems found itself in several years ago. It prompted the development of procedures and software—collectively known as Change Control—as a way of preventing a disaster that was waiting to happen. During the last three years, Change Control has continued to evolve as it has proven to be reliable and robust. It has been accepted by those who use it largely because it places very few constraints on individual development styles. In fact, developers have come to appreciate the additional control over the testing, packaging and shipping of software releases that has resulted from the adoption of Change Control—no one likes emergency support calls!

The description of Change Control that follows is drawn largely from its development within an MS DOS environment as part of the *4XTRA* foreign exchange trading application written in APL*PLUS/PC.

What Does Change Control Do?

Change Control monitors and controls the migration of changes to the application code throughout a system, archiving old code before overwriting it with revised versions or adding new items. It also supports the administrative activities that are interleaved with system development and testing. Each time a change is promoted, the integrity of the system is checked and the history of the change is updated and saved. The history is used to create documentation that accompanies the release. Change Control can browse the state and history of any active change, restore archived objects, and delete changes altogether.

Part of a Bigger Picture—Configuration Management

Before we describe the structure and operation of Change Control, we need to situate it within a broader context. Writing, testing, packaging and distributing software releases are only part of a much bigger topic, Configuration

Management. CM, as it is popularly known, deals with the management of software development in its broadest sense, from formal methods through software and hardware products to staff training. Change Control is insufficient in itself to provide a complete CM solution. However, by touching briefly upon three aspects of CM, we can provide some insight into the more important parts of the framework that surrounds Change Control.

Analysis and Design Method

The phrase *modular prototyping* best describes the analysis and design method that has been used for *4XTRA*. It has much in common with Rapid Application Development. RAD is characterised by extensive user involvement and the division of proposed system development into manageable, functional units that can be delivered as working modules within short time scales. The analysis that we carry out as part of modular prototyping provides input to a resource estimation algorithm that we have implemented as a spreadsheet template. We use this to determine the resource requirements for all releases, whether they are simple bug fixes or major enhancements. This process highlights dependencies between different releases. It also allows the subdivision of a release into smaller work packages of logically or functionally related objects. This is especially important when a release contains objects that are used by more than one process, or when work packages can be developed in parallel by different people.

Within Change Control, a software release comprises one or more work packages, and a developer is allocated one or more of these. Each release has a unique identification number, easily identified work package components, clear dependencies, and specific resource requirements and commitments.

System Implementation Model

4XTRA has been implemented using an overlay model and indirect file references. It has a single workspace together with one or more program or system files and a separate set of application data files. A program file is an APL component file that contains control variables and canonical or vector or object representations of functions. Each program file contains a namelist of the items stored in it, together with additional information about each object on the file, including a pointer to the name of the person who last wrote it to file. A second namelist and an associated array of pointers are used to refer to and define groups of objects. All access to the program file is controlled through cover functions. Only the most widely used utilities reside in the workspace.

In operation, each functional module in the system is associated with a group name on the program file. When the user runs a module, the system reads from file and defines in the workspace the objects that comprise the group name associated with that module. When the module terminates, the objects that were read from file are expunged.

Indirect file references have been implemented by using cover functions for all file creation and file ties. All application code contains only token file names. This allows runtime mapping of each token name to a real file name, directory or drive.

Change Control is itself an overlay module. It also uses the program file structure for storing (change files) modifications to, and saving (archive files) old versions of, individual objects and group definitions. Indirect file references make it possible to avoid the duplication of entire read-only datasets that would otherwise be necessary for development and testing.

Development Environment

Change Control succeeds only if everyone uses it. To encourage its adoption, we devised a fairly free-form coding and development environment to accommodate the full range of work styles among a group of system developers. We have provided ourselves with a set of tried-and-tested tools, and mandated the use of only one of them. The required tool is a cover function used to edit functions. When the cover function terminates, it inserts or updates the version number, developer name, current date and time in a comment at the top of the edited function. As a safeguard, the function that writes objects to a program file will not write a function without a comment as its first line. In addition to these tools, there is on-line documentation that describes naming conventions, utilities, etc.

As described in the previous section, the system uses overlays to define, execute and expunge objects in the workspace. For development and testing, this behaviour has been modified so that a functional module defines in the workspace only those objects in the group that do not already exist there. Similarly, when a module terminates, only the objects previously read from file are expunged. This allows the developer to copy functions and variables into the workspace, set stop and trace controls, and to edit freely without worrying that the system will get rid of the product of several hours, or even days, of hard work!

The resulting development environment allows experimentation, coding and testing in personal workspaces without fear of side effects. More importantly, it leaves control in the hands of the developer. However, the propagation of changes

through an application from the point at which a developer has finished work until the changes are ready to be applied to the production system is managed through Change Control.

Change Control in Operation

Application Structure

Change Control utilises an application directory structure that separates the application code from the data, and provides a storage location for release documents (Figure 1). Beneath the code and data are four separate subdirectories, each with a specific purpose:

DEVT For development and unit testing.

TEST For suite, system and volume testing.

RLSE A holding area pending release.

LIVE For parallel runs with a copy of the production system data. Generally used for client acceptance testing.

This does not imply that there are four complete copies of the application system in existence simultaneously. Where appropriate, indirect file references at each level can point to the same read-only files.

Development and Testing Activities

Every software release requires analysis to identify work packages and to allocate resources. Generally, the developer who does the coding and unit testing is also involved in the analysis. The release is then given a unique identification number. The Change Control module is invoked at this point to initiate the release at the work package level, i.e., to add the release ID, work package number (1 to 99) and title to the main Change Control registry file. Change Control constructs a unique name for the change file that will be the repository for all modified and new objects that form this work package. The file name has the form *Snnnnnxx*, where *nnnnn* is the release ID number and *xx* is the work package number within the release. For example, the name of the change file for the first work package of release 371 is S0037101. The same scheme is used for temporary (*Annnnnxx*) and permanent (*Rnnnnnxx*) archive files that are created as a work package is promoted from one level to another.

A summary follows of the sequence of activities undertaken by a developer working alone in a single application environment (we describe working with multiple developers and sites a little later). Words in bold correspond to

commands within Change Control selected by the developer (steps 1, 2, 4, 5, 7, 8). The figures in brackets refer to the paths in Figure 2, which illustrates what Change Control does to objects and to change and archive files for each activity. Only the objects about to be changed are archived, not the entire system file.

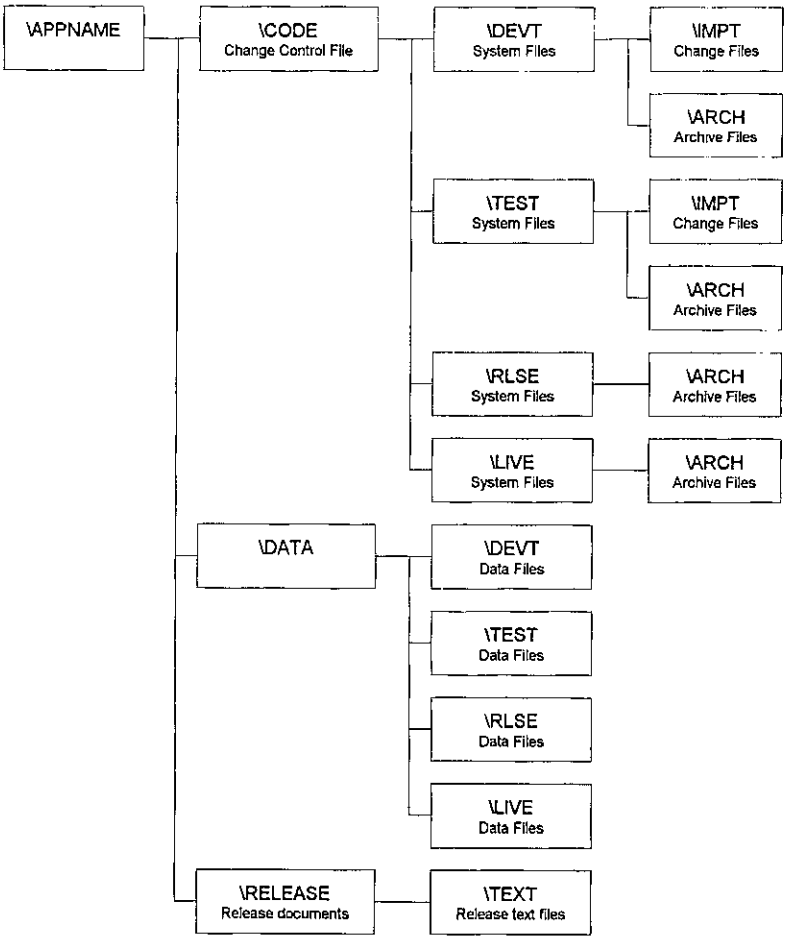


Figure 1. Application Directory Structure

1. Initiate a new work package.
   The developer enters the release ID, work package number and title, then selects the objects. They are copied from the DEVT system file to a new change file [1] ready to be imported to TEST.

2. **Reject** the changes.
   Move the change file [2] ready to be imported to DEVT.

3. Create or modify any application data files in DATA\DEVT. Bring the objects in the change file into the workspace. Edit and unit test them. When finished, store the modified objects back in the change file.

4. **Import** the changes.
   Archive objects from DEVT system file [4a];
   Copy objects from change file to DEVT system file [4b];
   Move change file [4c] ready to be imported to TEST.

5. Prepare to **Test** the changes.
   Archive objects from TEST system file [5a];
   Copy objects from change file to TEST system file [5b].

6. Create or modify any application data files in DATA\TEST. Modify the standard test data. Run the tests.

7. **Accept** the changes.
   Archive objects from RLSE system file [7a];
   Copy objects from change file to RLSE system file [7b].

8. **Release** the changes.
   Archive objects from LIVE system file [8a];
   Copy objects from change file to LIVE system file [8b].

9. Create or modify any application data files in DATA\LIVE. Conduct one or more parallel runs.

10. Copy the LIVE system file to the Production system. Create or modify any application data files in the Production system.

Change Control manages the migration of objects and group definitions but certain activities are handled manually. These generally deal with one-off modifications to data files that are required as part of a software release. A mechanism for handling these has been built into Change Control but has seldom been used because these changes occur infrequently and tend to be highly idiosyncratic. Instead, the software release includes a supplementary program file that contains routines that are run only once in the workspace by development or operations staff. These routines are developed and tested in the usual way, and instructions are included in the release documentation.

In operation, Change Control allows the developer to use the mouse or keyboard to select items directly from list boxes and to employ various search mechanisms to reduce the lists to a more manageable size. Each Change Control activity shown above also allows the developer to add comments to the change history, an on-line audit trail that is stored in the change file. The change history is updated automatically with information about all the objects in the change file, when it was last promoted and by whom.

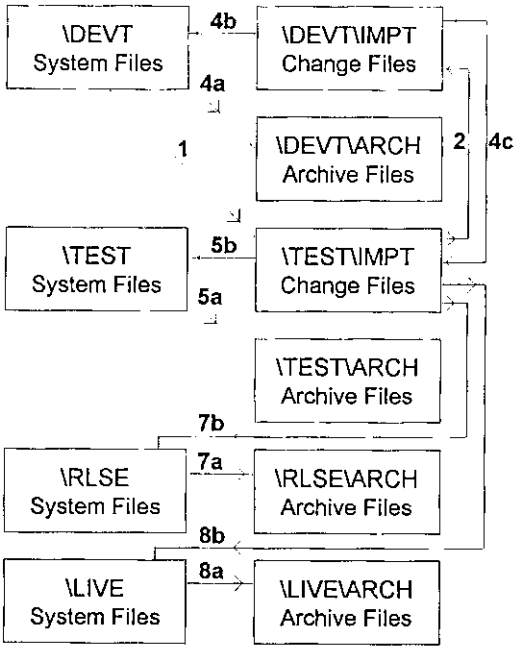


Figure 2. APL Object and Program File Movement within Change Control Activities

Whenever a change file is imported, Change Control extracts from the change history various items of information that it writes to a text file. The developer creates a release document for the work package in Microsoft Word, using a defined template and macro to fill in much of the form from the text file. The developer then enters additional details about the new or changed objects in the release such as interdependencies, special cases and restrictions, installation instructions, and any changes to system outputs, user documentation, technical support procedures, etc.

At first glance, all of this probably appears intimidating or unnecessarily tedious. The activities performed through Change Control require negligible time but it is imperative that they be done correctly the first time. We have found that the overall procedure imposes a discipline that has increased the time available for analysis and development while reducing the time spent on support. The process also includes an audit trail, specifications and amendments to documentation. Developers test each other's coding changes in-house and check the additional information in the release documents before anything gets near the client. And, not only do clients receive a better quality product, they are able to test it themselves within their own Change Control structure (see below).

Other Change Control Facilities

There are some additional commands available in Change Control.

Reject
When applied to a change at any level, **Reject** restores, at all levels, the previous versions of all archived objects that are associated with the change file. As described above, the change file is then moved from TEST\IMPT to DEVT\IMPT. The state of the change becomes inactive.

Delete
Removes all evidence of an inactive (Rejected) change from Change Control.

Browse
Displays a summary of all known changes and allows viewing of the change history of any one of them.

Freeze
Prevents further changes from being promoted to the LIVE level. This removes concern about the possible effects of other changes on extended parallel testing.

Unfreeze
Removes the "lock-out" imposed by **Freeze**. At this point the LIVE system files can be moved to the production system. Change Control deletes any change file that has been promoted to LIVE as well as all associated temporary archive files in the \ARCH directory at the TEST, RLSE and LIVE levels. The archive files for these changes in DEVT\ARCH become permanent archives by changing the first character (for example, S0037101 is renamed R0037101).

Change Control contains several safeguards. Whenever it promotes changes to another level, it looks in the system file that it is about to update and reports any unauthorised changes. This procedure ensures that no-one can update a system

file directly at any level without detection. In a way, it protects developers from themselves!

Change Control also compares the list of objects in any new change file with the lists of objects in all other active changes. This is designed to prevent a change from being initiated or imported if it contains an object in another change file that has not yet been accepted. If an object is part of a release that has been accepted or released, Change Control issues a warning.

Finally, it is possible, after initiating or importing a release, for the developer to realise that one or more additional objects need to be included in the work package. Change Control can handle this situation in one of two ways. Either the current change file is rejected and the process started anew, or a second change file (a new work package within the same release) is initiated or imported.

Using Change Control across Multiple Sites

From the outset, Change Control was designed to coordinate activities across multiple sites, whether they are physically separate or merely different logical partitions on a single disk drive. In its simplest form, all development and integration testing takes place in one location, and the client site conducts testing but no development. This allows clients to set up their own data sets for testing. This is especially important for parallel runs, as these generally contain highly sensitive data.

Two additional considerations arise when development occurs on more than one site. First, there is the need to avoid duplicating release and work package IDs. These can be allocated centrally as each work package is defined. Alternatively, the IDs can be allocated in blocks to different developers, especially when the developers conduct the analysis for the work package in the first place. When the block of IDs is exhausted, they simply ask for another. The second consideration is the increase in traffic that is required to ensure that all of the developers have an up-to-date copy of the system. Every release that is exported from the main development site to the client must also be sent to all of the remote development sites.

Work package details are distributed to developers at remote locations by the most appropriate method (electronic mail, fax, snail mail). If the analysis is to be carried out remotely, only the change request information is passed on. When completed, the change files and release documentation are sent from the developers' sites via floppy disk, electronic mail or a direct machine-to-machine link to the main development environment where they are imported. The ideal migration path is shown in Figure 3 with a solid line, but an alternative (and more

frequently used route) is shown with the dashed line. The main development site integrates the changes before testing. The quality of the release documentation is also checked to ensure that the operations staff at the customer site will be able to test and install the changes. After everything is approved, the entire software release is shipped to the customer site, using one of the same methods. Local staff then test the release to satisfy themselves that it works as described before installing it on the production system.

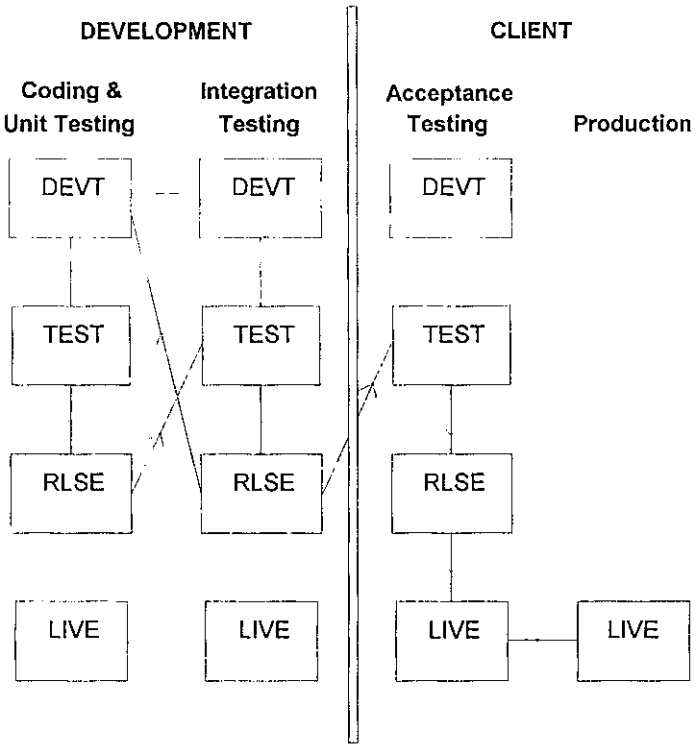


Figure 3. Change Migration across Multiple Sites

Current and Future Work

Change Control together with the RAD-type design method has provided us with the ability to meet the demands of quite diverse groups. Developers are largely free to use any toolset and work however it suits them. Developers can also spend more time developing and less time supporting clients or themselves (*4XTRA*'s predecessor system had 110 workspaces—migrating code was almost a job for life!). Project managers have access to a visible process with an audit trail that is maintained automatically. Their staff are able to test new releases locally with their own data sets before the software becomes part of the production system. Finally, users take delivery of robust software in reasonably short time frames. They spend more time working and less beta-testing.

As a module within *4XTRA*, Change Control has been ported to Dyalog APL/W together with the rest of the application code. This has presented an opportunity to isolate it as a free-standing module (an exercise that will be completed shortly) that might be able to bring some order to legacy systems as well as supporting new system development. With this in mind, there are plans to investigate and, where feasible, to modify Change Control to:

- automate the creation and amendment of on-line help
- improve on-line and hard copy reporting
- work with, and within, third party products such as Causeway
- support workspace-based applications
- support mainframe or hybrid mainframe/PC applications
- run under APL2000 and APL2/Win interpreters in addition to Dyalog
- accommodate namespaces
- serve as a component within an ISO 9000-compliant environment

So, watch this space!

C Source Code Modification using APL with Limited Memory

by Adam Weber (100537.1037@compuserve.com)

Introduction

Modification of C source code using APL is not unheard of. In this case, the problem was to update a C header file using a C MAP file.

MAPFILE.MAP contains a symbol table with variable names and their respective addresses. The header file HFILE.H contains #define pre-processor commands for variable names and respective addresses. New addresses in MAPFILE.MAP had to replace the corresponding old addresses in HFILE.H.

Easy (you think)! Read the files into 2 variables, then write a couple of one liners to find occurrences of the variable names of MAPFILE.MAP in HFILE.H, and replace the respective addresses.

But you only have APL*PLUS PC which only uses 640K. Still easy!

Ahh, but the files have lots of duplicated stuff, and are vectors containing carriage returns — slightly harder! When you process the huge variables containing the files using your one liners, you get the dreaded WS FULL. Optimising memory still gives WS FULL. Not so easy after all!!!

The answer lies in solving the problem using a more conventional programming approach, as you would using a language such as C. Basically, you have to process the files line by line. Granted, this means using loops, hence slower execution in APL, but you'll have bags of memory to spare for a decent front end. Also, few lines of code are required.

The File Structure

The symbol table map file looks like this:

```
[listed by asc name]
[address] [var name]
 00811321 _ER1
 0081322f _stack
 00812532 _UDIV30
```

```
[listed by asc address]
[address] [var name]
 00811321 _ER1
 00812532 _UDIV30
 0081322f _stack
```

The header file looks like this:

```
        [var name]                  [address]
#define ER1                         0x0081132a
#define stack                       0x00813121
#define UDIV30                      0x0081a23f
```

As can be seen, the map file has everything duplicated. Both files are one long vector.

How the System Works

The processing method is very straightforward.

LOOP:

1. Read a line from MAPFILE.MAP.
2. Extract the variable address and name.
3. Search all lines in HFILE.H for the variable name extracted.
4. Replace original respective HFILE address with the new MAPFILE one.

→*LOOP*

This is actually a little simplified because, as you'll see later, this isn't very fast, and there are enhancements to make it 5 times faster.

Reading a Line

You can't simply read a "line" from a native file as you would using a text editor. A line (usually) contains text followed by a carriage return. You have to find the end-of-line character, □*TCNL*, (Return Key). Let's assume that the maximum length of a line is no longer than 95 characters. Reading 100 characters would guarantee reading a whole line. From the 100 characters, find the first occurrence of □*TCNL*, then take all the characters before it and, hey presto, you have a line of text. Then you would repeat the process starting from the position of the previously located □*TCNL* to get the next line. The function *GETLINE* does precisely this, and you can specify the file tie number, the length of the line to read, and where to start reading.

Extracting the Variables from MAPFILE.MAP

Remember that MAPFILE.MAP contains two columns of duplicated variable names and addresses, so we only need bother with one of them, the first one.

When a line is read, we need to extract the first address and name. Fortunately, all addresses in MAPFILE.MAP are 8 characters long, so taking 8 characters is easy. Then dropping the address (and a trailing space) leaves the variable name followed by at least one space. So, find that space and take everything before it. The *READVARS* function does this and returns a string with an address catenated to a variable name.

Replacing Old Addresses with New Ones

Armed with the address name, all we have to do is use ⎕*SS* to search for an occurrence of that variable name in the header HFILE.H, using the *GETLINE* function to retrieve one line at a time from HFILE.H. Note that this time, the lines are shorter, so only 80 characters need to be read to guarantee a whole line read. If a line does contain a matching variable name, find the position in the line of the old address, and replace it with the new address. Then simply use ⎕*NREPLACE* to overwrite the old line with the new one, hence ending up with an updated variable address. See the *SYSTEM* function, lines 25 to 27.

The *SYSTEM* function is basically a simple loop which reads new variable addresses and replaces the old ones. It calculates the position of the next line from the length of the line currently returned (+2 to compensate for ⎕*IO*←0 and to skip the ⎕*TCNL*).

More Speed

Originally, when the above method was implemented using a MAPFILE.MAP of 35k, and a 70k HFILE.H, the updating of all addresses took over 5 minutes, and this was using a VX motherboard with a P120 CPU!

Obviously, this is too slow. There are ways to speed it up using various search methods, however a quick solution is to only search the HFILE for lines which have not been updated. Basically, once a variable address has been updated, make sure that line is not in the search in future. A simple way of achieving this is to store the positions of ALL the lines in HFILE.H in a variable (called LINES). Lines 7 to 11 in the *SYSTEM* function do this. Use this list to read one line at a time. Then, when you update a line in HFILE.H, simply remove that line position from the list of lines in the LINES variable. The pre-searching of line positions is a small time overhead to pay for the overall decrease in processing time, which turns out to be just over 1 minute, hence giving a 400% improvement in speed.

Summary

Although this is not the fastest method for file processing, it requires very little memory, which is a precious resource if you're using APL*PLUS PC to process large files. You can speed things up by refining the methods, but you will usually have to trade speed in order to gain memory. Of course, you may be wondering why bother doing it this way? Why not simply use an APL interpreter that supports huge workspaces? Well, this is simply an exercise to show that there are ways around the limitations in APL*PLUS PC, methods which require a slightly different approach to the usual "one liner" APL solution.

Function Listings

```
    ∇ SYSTEM;CT;LINES;MPOS;HPOS;ADR;VAR;VARS;LINE
[1]         ⎕IO←0
[2]         'HFILE.H' ⎕NTIE ¯99
[3]         'MAPFILE.MAP' ⎕NTIE ¯100
[4]         MPOS←0 ⋄ LINE←' '
[5]
[6]        ⍝ Find all lines in header, storing index for speed later.
[7]         LINES←0 ⋄ HPOS←0
[8]     LR0:LINE←¯99 GETLINE(80,HPOS)
[9]         →(0=⍴LINE)/LOOP1
[10]        HPOS←HPOS+2+⍴LINE
[11]        LINES←LINES,HPOS ⋄ →LR0
[12]
[13]       ⍝ Main processing loop
[14] LOOP1:LINE←¯100 GETLINE(100,MPOS) ⍝ Get line from map file.
[15]        →(0=⍴LINE)/END
[16]        MPOS←MPOS+2+⍴LINE ⍝ Increment position in map file.
[17]        VARS←READVARS LINE ⋄ ⍝ Extract variable from map file.
[18]        ADR←8↑VARS ⋄ VAR←8↓VARS
[19]
[20]       ⍝ Loop to replace old variable addresses with new.
[21]        CT←0 ⋄ HPOS←LINES[CT]
[22]    LR1:LINE←¯99 GETLINE(80,HPOS) ⍝ Get line from header file.
[23]        →(0=⍴LINE)/LOOP1
[24]       ⍝ If line contains matching variable, update address.
[25]        ⍎(~((8↑((8↓LINE)⍳' ')↑8)↑LINE)≡VAR))/'HPOS←LINES[CT←CT+1] ⋄ →LR1'
[26]        LINE[((((LINE ⎕SS '0x')/⍳⍴LINE)+2)+⍳8)]←ADR
[27]        LINE ⎕NREPLACE(¯99,HPOS)
[28]       ⍝ Remove the line number updated, to skip in future.
[29]        LINES←(HPOS≠LINES)/LINES
[30]        →LOOP1
[31]    END:⎕NUNTIE ¯99 ¯100 ⋄ ⎕IO←1
    ∇
```

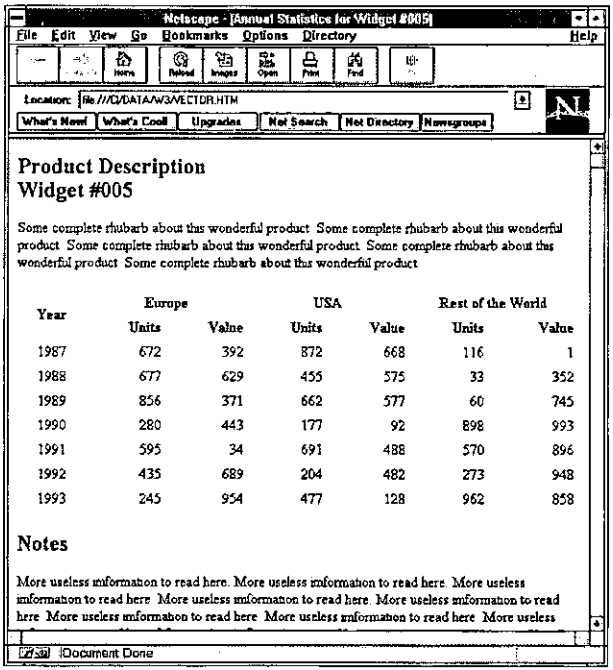
```
    ∇ R←file GETLINE params;L
[1]         ⍝ Read x chars from last pos. If index error, read max possible
[2]         ⍝ Return line read up to first encountered carriage return.
[3]         ⍝ SYSTEM fn will work out last pos from length of line returned.
[4]         ⍝ params = x chars, last pos
[5]         ⍝ eg. ¯100 GETLINE 100 0 = read NTIE ¯100, 100 chars, start at 0.
[6]         ⍝ Choose realistic value for params[1] ie larger than longest line.
[7]
[8]          →(((⎕NSIZE file)-(1↓params))≥(1↓params))/NORMAL
[9]
[10]        ⍝ Do line below if you can't read x chars near end of file
[11]         L←⎕NREAD(file,82,((⎕NSIZE file)-(1↓params)),(1↓params)) ⋄ →LCONT
[12]
[13] NORMAL:L←⎕NREAD(file,82,params)
[14]  LCONT:L←(1↑((L=⎕TCNL)/⍳⍴L))↑L
[15]         R←L
    ∇

    ∇ R←READVARS line;adr;var
[1]  ⍝ Read the first address in the line with corresponding variable.
[2]  ⍝ R = 8 char address catenated to arb. length variable name.
[3]   adr←8↑line
[4]   var←9↓line ⋄ var←(var⍳' ')↑var
[5]   ⍎('_'=(1↑var))/'var←1↓var'
[6]   R←adr,var
    ∇
```

HTML Basics for APLers

by Adrian Smith (causeway@compuserve.com)

Introduction

This is the first article in a set of at least two, and probably three tutorial introductions to the HTML standard, and to the APL code that I use to generate HTML output from applications. This first one covers the reasons why you might want to do this stuff, and goes far enough to let you create a very simple Web page from either Dyalog or APL+Win. In the next issue, we will go on to tackle tables, so to give you a feel for where this is taking us:



Product Description
Widget #005

Some complete rhubarb about this wonderful product. Some complete rhubarb about this wonderful product. Some complete rhubarb about this wonderful product. Some complete rhubarb about this wonderful product. Some complete rhubarb about this wonderful product.

Year	Europe		USA		Rest of the World	
	Units	Value	Units	Value	Units	Value
1987	672	392	872	668	116	1
1988	677	629	455	575	33	352
1989	856	371	662	577	60	745
1990	280	443	177	92	898	993
1991	595	34	691	488	570	896
1992	435	689	204	482	273	948
1993	245	954	477	128	962	858

Notes

More useless information to read here. More useless information to read here. More useless information to read here. More useless information to read here. More useless information to read here. More useless information to read here. More useless information to read here. More useless

Throughout this series, all the sample code will be in APL+Win 1.8. However it will translate trivially to a Dyalog namespace, and is available on the Causeway web site (www.causeway.co.uk/html.zip) in both formats. The code may be freely downloaded and used with no restrictions.

Motivation

One of the best attended workshops at Lancaster, and the most exciting topic at Orlando, was the TCP/IP hookup that can now be used to enable APL systems to act as purpose-built web servers, probably connected to an internal company network or **Intranet**. Both Dyadic and APL2000 are well on the way to providing TCP/IP sockets as a native component of the interpreter, so it is now up to you to develop applications which take full advantage of standard browsers such as *Netscape* or Microsoft's *Internet Explorer*.

Essentially, this means formatting your application output to the HTML standard. Of course you could give up and just return a formatted array as a simple text file - most browsers will display this in a fixed-pitch font with spaces preserved - but do you really want to maintain APL's reputation as a cranky old mainframe language which cannot even present a table decently?

Of course you probably want to play with this stuff in the comfort of your own front-room on a portable with no connections to anywhere. You probably don't want to understand all that TCP/IP jargon just yet, so how do you begin? Fortunately the web browsers make no distinction between a page that has come in from the net, and a simple text file formatted to the HTML standard. This makes it very easy to get started - grab a copy of Netscape (version 1.1 is entirely adequate for anything we need to do) and write your output to a simple ASCII file with the .HTM extension. Open it in Netscape, and park the window somewhere convenient. Then all you need to do is to keep overwriting the same file from APL and hitting Netscape's "Reload" button to view the results. You will soon get quite proficient at generating very acceptable web pages, and will very probably prefer APL to any of the over-engineered (over-priced) web-site generators that you see advertised in the magazines.

What is HTML Anyway?

Not an easy question to answer! A year ago the philosophy was clear, and the definition matched the philosophy; now everything is becoming a complete mess as Microsoft throw more and more junk into a language that was never designed to take it. The basic idea was that you describe **content**, and have your user (via his/her browser) determine the **presentation**. The beauty of this approach is that content is completely independent of the target machine, so you can publish material which anyone with an Internet connection can read. The downside is that you cannot specify that your headings should be 14-point Book Antiqua Bold,

because you have no idea if the page will be viewed on a machine which can display TrueType fonts, let alone one which has that font installed.

Now consider the market where the serious money is - the so-called "Intranet" of internal company networks. Here you know exactly what your target machines are, and you probably know that they are all running NT 4.0 with OfficePro 97. Now it makes perfect sense to specify font, and even to make some assumptions about screen resolution (surely everyone at least has SVGA), so suddenly HTML is being loaded up with a raft of 'presentational' tags which go directly against the original philosophy of the language. At the moment, my advice is to ignore these, as even the latest Microsoft browser (IE3) falls flat at many apparently legal constructs. However, there is nothing in the sample code which you cannot easily extend, so by all means give it a try if your HTML has a strictly local audience.

The key word in the above paragraph is *tag* which is just a name for instructions embedded in the text to tell the browser how you want it shown. The **bold** tag is an interesting example - I should really say **for Strong Emphasis** as it is up to the browser how to show it. Similarly I should use <cite>*for citations*</cite> and so on. Most normal people give up and use bold and italic!

The more important tags describe overall content, and the formatting for each paragraph. These are things like <h1>Major Heading</h1> and such. Probably the best way to introduce these is to work through a simple example, and explain the tags as we hit the APL code which generates them. There are plenty of good reference books around [1], or you can grab any of a number of excellent shareware HTML editors (I like Kenn Nesbitt's *WebEdit*) and browse its help file. Finally, there are tables, which are by far the trickiest thing to attempt by hand. The second article in this series will be mainly about tables, and again all the examples will work in Netscape and Explorer.

Back to Basics

Now you know what a tag looks like, and you have probably guessed that tags tend to come in pairs! They can also be nested, and you will find that there are 'structural tags' as well as the formatting markers that I used on the previous page. We should start at the outermost level of structure - the tags that flag the entire content as conforming to the HTML standard ...

```
<html>
... anything that is acceptable HTML
</html>
```

Then we mark out two major sections in our document; it should have a **heading** and a **body** ...

```
<html>
      <head>
      ... descriptive stuff about the document as a whole
      </head>

      <body>
      ... the bit the browser shows in its window
      </body>
</html>
```

... the indenting is just for clarity and ease of manual editing. Browsers ignore all spare white space and carriage-returns so when you come to write this from APL you can ignore virtually all the layout conventions and the results will look just the same to the user!

The Heading Section

Let's start with the code to make a valid heading section, and leave the body to its own devices for the minute:

```
<head>
      <title>A Title for My Page</title>
</head>
```

... the title is required, and is used by the browser in its own window title. That really is all you need, unless you want to influence the way search engines catalogue and present your page. So the first APL function is really quite trivial:

```
     ∇ ttl htmUse dummy
[1]    ⍝ Start off our HTML report
[2]    ⍝
[3]     ⍎(0=⎕NC'ttl')/'ttl←''Sample HTML Report'''
[4]     htmInit
[5]     htmtInit        ⍝ Table setup - see next article!
[6]     htm_cat '<HTML><HEAD>'
[7]     htm_cat '<Title>',ttl,'</title>'
[8]     htm_cat '</HEAD>'
[9]     htm_cat '<BODY bgcolor="#FFFBF0">'
[10]    htm_cat ''
     ∇
```

The syntax of this is inherited from Causeway's *NewLeaf* workspace [2], hence the dummy right argument where a page-layout would normally be given. Maybe one day it will be possible to specify (for example) the frames to be used, so I left

this in for future use. The first thing it does is set up any required temporary variables, then it begins to build our page:

```
    ∇ htmInit
[1]     htmStyle'Body'    ⍝ See later!
[2]     htm_PG←''
    ∇

    ∇ htm_cat tv
[1]   ⍝ Catenate text to back of global HTML buffer
[2]   :if 2>≡tv
[3]     htm_PG←htm_PG,tv,⎕TCNL
[4]   :else
[5]     htm_PG←htm_PG,↑,/tv,¨⎕TCNL
[6]   :end
    ∇
```

... which simplifies as necessary and accumulates the text in the workspace as we proceed. Having run `htmUse`, the buffer will look like:

```
'Annual Statistics for Widget #005'htmUse''

      htm_PG
<HTML><HEAD>
<Title>Annual Statistics for Widget #005</title>
</HEAD>
<BODY bgcolor="#FFFBF0">
```

Notice that some tags (in this case <body>) can include extra attributes. These are listed in any order as simple "property=value" pairs, separated by blanks. Unrecognised properties are simply ignored, which is just as well given the divergence between browsers. In this case I have overridden the default grey background with a gentle parchment shade using the colour string "#FFFBF0" which gives the red,green,blue components on a scale of 0-255. You might prefer plain white, which is easy to remember, being simply "#FFFFFF".

On to the Body

The main part of the page will typically include some headings, some free text and very likely a table of figures. You can stay with the basic styles offered by the browsers, or you can choose to implement your own mapping between 'Major Heading' and the low level tags.

Because I find the majority of Netscape's built-in headings ugly, I chose to define my own styles and generate the tags from these:

```
      'Subhead'htmPlace 'Product Description' 'Widget #005'
      htm_PG
<HTML><HEAD>
<Title>Annual Statistics for Widget #005</title>
</HEAD>
<BODY text="#004040" bgcolor="#FFFBF0">
<h2>Product Description<br>Widget #005</h2>
```

What *htmPlace* has done is to look up its left argument in my style table:

```
    ∇ htmStyle id;pos
[1]   ⍝ Look up requested style in ∆gallery and action it.
[2]    htm∆tag←,⊂,'p'
[3]    pos←(htm_uc¨(⍴id)↑¨htm∆gallery[;1])⍳⊂htm_uc id ⋄
        →(pos>1↑⍴htm∆gallery)↑0
[4]    htm∆tag←pos⊃htm∆gallery[;3]
    ∇

      htm∆gallery
 Body    Normal Paragraph          p
 Indent  Simple Indent            ul
 Code    APL Listings            pre
 Heading Major heading     h1 center
 Subhead Minor heading             h2
```

... and store the tag (or tags) which implement it in a global variable. As you learn more HTML, you can experiment with more exotic tags, such as setting the font size and face for your major headings. The nice thing about using a style table is that existing application code just keeps running.

Back to that *htmPlace* function. It expects either a matrix or vector of text vectors and formats the output as HTML, preserving newlines by inserting
 tags to force the browser to make a line break:

```
    ∇ r←sty htmPlace txt
[1]   ⍝ Simple text placement, taking account of style
[2]   ⍝ Breaks at the end of each line.
[3]    htmUseDflt ⋄ htmStyle'Body'
[4]    →(0=⎕NC'sty')↑Deflt ⋄ htmStyle sty
[5]   ⍝ Ensure correct depth
[6]   Deflt:→(2≤≡txt)↑Fmt
[7]    :if 2≤⍴⍴txt
[8]      txt←⊂[2]txt
[9]     :else
[10]     txt←⊂,txt
```

```
[11]    :end
[12]   Fmt:txt←,⍕¨txt ⋄ →(0∈⍴txt)↑Exit
[13]    htm_cat (↑,/'<',¨htmΔtag,¨'>'),
          (¯4↓↑,/txt,¨⊂'<br>'),(↑,/(⊂'<'),¨(⌽htmΔtag),¨'>')
[14]   Exit:
    ∇

    ∇ htmUseDflt;sink
[1]   ⍝ Check if we have initialised and do so if not!
[2]    →(0≠⍴htm_PG)↑0
[3]    sink←htmUse''
    ∇
```

Line[13] is the only interesting one! It applies the tags which implement our style, then unapplies them in the correct (i.e. reversed) order. This means that a major heading will be wrapped with <h1> Heading is Here </h1> so that the tags are correctly nested.

Let's end it there and have a look at a completed HTML report:

```
      htmPlace '⍝ International Widgets Inc 1996'
      ⎕←qq←htmClose
<HTML><HEAD>
<Title>Annual Statistics for Widget #005</title>
</HEAD>
<BODY text="#004040" bgcolor="#FFFBF0">
<h2>Product Description<br>Widget #005</h2>
<p>&#169; International Widgets Inc 1996</p>
</body></HTML>
```

... which shows the overall structure quite clearly. The report has two sections, delimited by the <head> and <body> tags. Within each section, paragraph tags come in strict pairs, and can be nested as long as you back out in the order you came in. Within a paragraph, most tags again pair, but there are a few special ones such as
 which signals a newline, which have no closing partner. Almost everything here is plain ASCII, but one snag is that any characters above 127 must be specially encoded, in this case the © symbol which is decimal 169, and happens to be the APL comment symbol in the APL+Win and new Dyalog mappings:

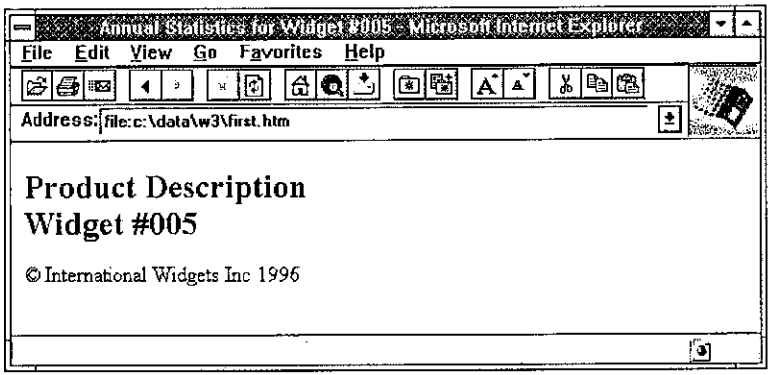
```
    ∇ r←htmClose;nl;sink
[1]   ⍝ Return completed job to calling fn
[2]    htm_cat '' '</body></HTML>'
[3]    r←htm_ToASCII htm_PG
[4]   ⍝ Tidy any working vars (htm_ prefix only)
[5]    nl←'h' ⎕nl 2
[6]    sink←⎕ex (nl[;⍳4]∧.='htm_')⌿nl
    ∇
```

```
    ∇ r←htm_ToASCII vec;asc;hi;hex
[1]   ⍝ Ensure all paragraphs go out as ASCII text.
[2]   ⍝ Hi-bit characters get the hex translation option!
[3]    asc←'1234567890','ABCDEFGHIJKLMNOPQRSTUVWXYZ'
[4]    asc←vec∈asc,' abcdefghijklmnopqrstuvwxyz
                    ,./;:<>?\!''"#£$%^&*()-_=+[]{}~@|',⎕TCNL
[5]    r←vec ◊ hi←(~asc)/⍳⍴vec ◊ →(⍴hi)↓0
[6]    hex←(⊂'&#'),¨⍕¨127+htm⍳hibit⍳vec[hi]
[7]    r←(1+5×~asc)/vec
[8]    hi←hi++\¯1↓0,(⍴hi)⍴5
[9]    r[,hi∘.+0 1 2 3 4 5]←↑,/hex,¨';'
    ∇
```

If you have been following along with the code so far, you might like to save this to disk and have a look at it in your browser ...

```
'first.htm' htmPut qq

    ∇ fi htmPut txt;fh;nl;lf;msk;pos;⎕elx;cr
[1]   ⍝ Put a ⎕TCNL vector to file <fi>
[2]   ⍝ Errors if failed.
[3]    fh←¯1+⌊/0,⎕nnums ◊ (lf cr)←⎕tclf ⎕tcnl
[4]   ⍝ Look out for existing file ...
[5]    ⎕elx←'→New'
[6]    fi ⎕NTIE fh ◊ ⎕elx←'⎕DM'
[7]    0 ⎕NRESIZE fh ◊ →Append
[8]   New:⎕elx←'⎕DM'
[9]    fi ⎕NCREATE fh
[10]  ⍝ Nest and pair CR/LF from ⎕TCNL
[11]  Append:msk←txt=cr ◊ pos←msk/⍳⍴msk
[12]   txt←(1+msk)/txt
[13]   txt[1+pos+0,+\¯1+(⍴pos)⍴1]←lf
[14]   txt ⎕NAPPEND fh ◊ ⎕NUNTIE fh
    ∇
```

Here is the result when viewed in IE2 (Windows 3.11):

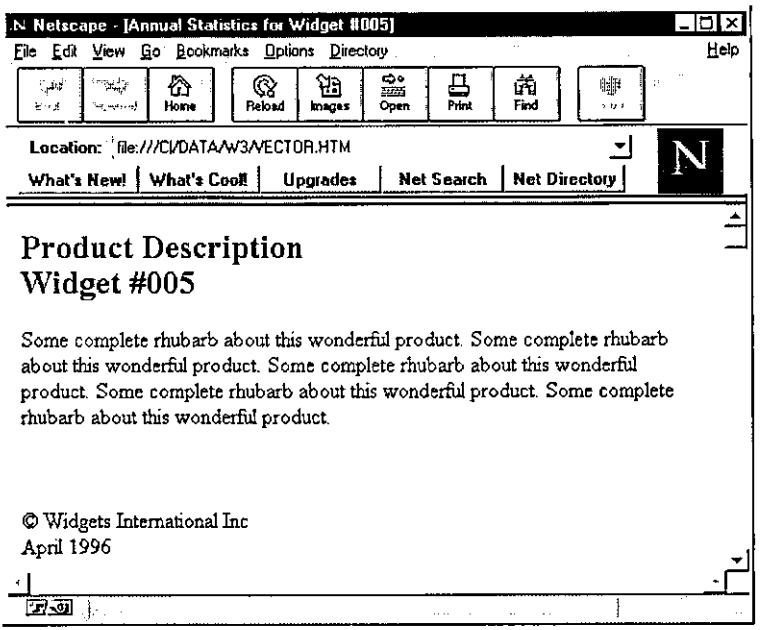


... and Netscape will show something remarkably similar. Just for completeness, let's add a paragraph of free text and a rule, and have a look at the result in Netscape 2.0:

```
    ∇ r←Vector;mat
[1]   ⍝ HTML example for Vector 13.4
[2]    mat←(1986+⍳7),??6⍴1000
[3]   ⍝ Page title and product info ...
[4]    'Annual Statistics for Widget #005'htmUse''
[5]    'Subhead'htmPlace 'Product Description' 'Widget #005'
[6]    htmFlow ∈5⍴⊂'Some complete rhubarb about this wonderful
        product. '
[7]    htmRule 2
[8]    htmPlace'⍝ Widgets International Inc' 'April 1996'
[9]    PG←htmClose
[10]   r←'''vector.htm'' htmPut PG  ⍝ to see it in Netscape'
    ∇

    ∇ sty htmFlow txt
[1]   ⍝ Simple text flow, taking accont of style
[2]    htmUseDflt ⋄ htmStyle'Body'
[3]    →(0=⎕NC'sty')↑Deflt
[4]    htmStyle sty
[5]   ⍝ Ensure correct enclosure
[6]   Deflt:→(2=|≡txt)↑Encl ⋄ txt←⊂,txt
[7]
Encl:txt←(,/'<',¨htm∆tag,¨'>'),¨txt,¨,/(⊂'</'),¨(⌽htm∆tag),¨'>'
[8]    htm_cat txt
    ∇
```

```
      ∇ htmRule wt
[1]     ⍝ Rule across (no parameters that we know about)
[2]     htm_cat '<HR>'
      ∇
```

As you can see, there is very little magic here, and some simple tags can produce something very acceptable:



Now all we need is a table, and perhaps a chart or company logo just to liven it up a little. If you don't want to wait for the July Vector, just grab the code from the web site and take it apart. I'm sure you'll find plenty to improve upon.

References

[1] *The HTML Sourcebook*, Ian S. Graham, John Wiley 1995

[2] *NewLeaf User's Manual*, Causeway Graphical Systems Ltd, 1996.

Turtlegraphics with APL2

by Hendrik Rama and Martin Barghoorn (barg@cs.tu-berlin.de)

Turtlegraphics are well known from the LOGO-System [1]. These graphics are also vector graphics not made by setting the absolute X-Y-coordinates but by relative increments of distances and angles. Perhaps with some new APL2-Idioms we have developed in a short time more than 1500 graphics from different classes.

Derivation of the Formulas

We start with the APL2-Syntax (polar form of complex numbers), five points, angle 90 degrees.

```
      5⍴1D90
0J1 0J1 0J1 0J1 0J1
```

Rotation by multiplication and accumulation

```
      +\×\ 5⍴1D90
0J1 ¯1J1 ¯1 0 0J1
```

Splitting the real and the imaginary part

```
      9 11○.o+\×\ 5⍴1D90
0 ¯1 ¯1 0 0
1  1  0 0 1
```

Transpose for drawing X-Y-graphics

```
      ⍉9 11○.o+\×\ 5⍴1D90
 0 1
¯1 1
¯1 0
 0 0
 0 1
```

We get a quadrat (square) Q1 and then we take *TRANSLATE* and *MAGNIFY* from GRAPHPAK to find the right position on the centre of the screen (100 × 74 pixels).

```
DRAW 60 30 TRANSLATE 25 MAGNIFY Q1← ⍉9 11○.o+\×\5⍴1D90
```

Without the complex-numbers-syntax we take Cosine and Sine.

```
Q2← ⍉ +\2 1 ○.o +\5⍴ o 90÷180    ⍝ COS, SIN
```

An alternative method is

```
      Q3← ⍉ +\2 1 ∘.○ +\5⍴ .5          ⍝ COS,SIN,ALTERNATIVE
```

Proof

```
      (Q1≡Q2) (Q2≡Q3) (Q1≡Q3)
1 1 1
```

Different shapes of polygons (different classes)

```
      +\×\ 4⍴1D120  ⍝ Triangle
      +\×\ 5⍴1D90   ⍝ Quadrat (square)
      +\×\ 6⍴1D72   ⍝ Pentagon
      +\×\ 7⍴1D60   ⍝ Hexagon
      +\×\ 9⍴1D45   ⍝ Octagon
```

Rosettes

The application of the replicate-function generates multiple sets of data.

```
      ⍉9 11∘.○+\×\ 5 9/1D90 1D¯45
```

To get *closed* figures we need replications and further manipulations.

```
FIG 1: ⍉+\2 1∘.○+\∈12/⊂(7+⍳7)/7⍴2 ¯2÷3 4
FIG 2: ⍉+\2 1∘.○+\∈17⍴⊂(3+⍳4)/4⍴2 ¯2÷8 4
FIG 3: ⍉+\2 1∘.○+\∈8⍴⊂(5+⍳9)/9⍴3 ¯1÷6
FIG 4: ⍉+\2 1∘.○+\∈8⍴⊂(5+⍳7)/7⍴3 ¯1÷6
FIG 5: ⍉+\2 1∘.○+\∈7⍴⊂(5+⍳8)/8⍴3 ¯2.5÷6
```

Trees [2]

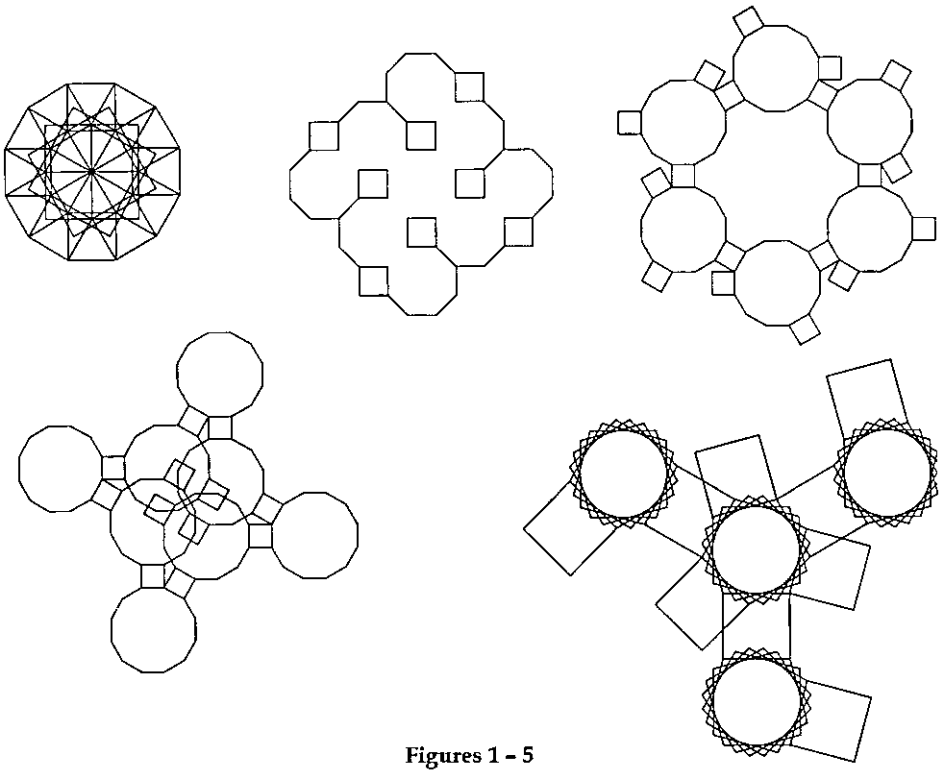
VIS: Visibility, first column of the matrix

```
FIG 6: AX←,⊃.7×∘.,/4⍴(⊂1 1D15 1D¯15 1D30 1D¯30)
       AXX←⍉9 11∘.○AX1←2.5D90×∈0,+\×\⊃AX
       VIS←~0=AX1 ◊ ERASE ◊ DRAW VIS,30+8×AXX

FIG 7: AX←,⊃∘.,/5⍴⊂.5×(.03+(1 2D90 2D¯90.01)(1 2D¯90.01 2D90))
       AXX←⍉9 11 ∘.○ AX1←2.5D.0010×∈0,+\×\⊃AX
       VIS←~0=AX1 ◊ ERASE ◊ DRAW VIS,30+8×AXX

FIG 8: AXX←⍉9 11 ∘.○1D90×∈0,+\×\⊃,/,/.8,[2.5](.8D55 1D36
1D¯65)[⍉1+(5⍴2)⊤¯1+⍳32]
       VIS←~(0=AXX[;1])∧(0=AXX[;2])
       ERASE ◊ DRAW VIS,40 +10×AXX
```

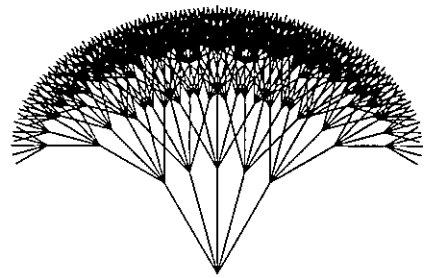
```
FIG 9: AXX←⍉9 11 ∘.○1D90×∈0,+\×\⊃,/,/.8,[2.5](.8D52 (.7D¯33
1D14).9D5)[⍉1+(4ρ3)⍳¯1+ι81]
       VIS←~(0=AXX[;1])∧(0=AXX[;2])
       ERASE ◊ DRAW VIS,30+10×AXX

FIG 10: AXX←⍉9 11 ∘.○1D90×∈0,+\×\⊃,/,/.8,[2.5](.8D52 (.7D¯33
.5D14 ¯1.1D14).9D5)[⍉1+(4ρ3)⍳¯1+ι81]
       VIS←~(0=AXX[;1])∧(0=AXX[;2])
       ERASE ◊ DRAW VIS,30+10×AXX
```

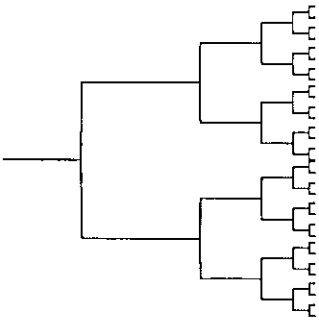
Patchworks

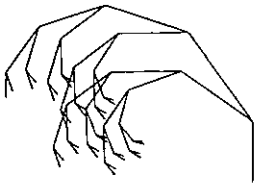
```
FIG 11: ⍉+\2 1∘.○○+\∈11ρ⊂(6+ι10)/2÷(10ρ1 2 1)×.6
FIG 12: ⍉+\2 1∘.○○+\∈11ρ⊂(6+ι16)/2÷(16ρ1 2 1)×.6
FIG 13: ⍉+\2 1∘.○○+\∈11ρ⊂(2+ι12)/2÷(12ρ1 2 1)×.6
FIG 14: ⍉+\2 1∘.○○+\∈11ρ⊂(0+ι7)/2÷(7ρ1 2 1)×.6
FIG 15: ⍉+\2 1∘.○○+\∈11ρ⊂(2+ι6)/2÷(6ρ1 2 1)×.6
```

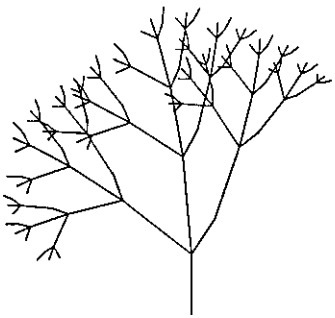


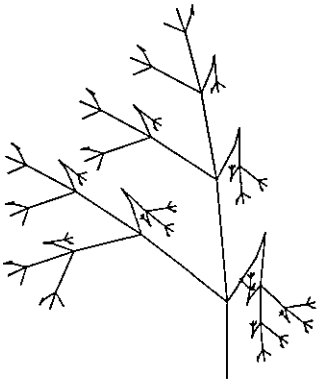
Figures 1 - 5



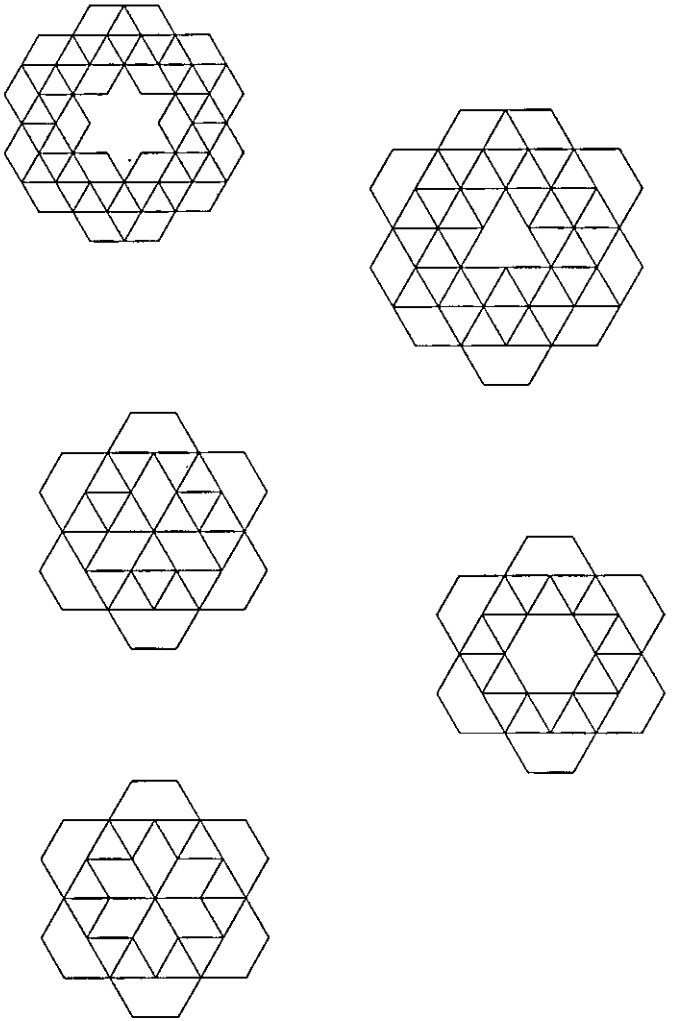








Figures 6 – 10



Figures 11 - 15

References

[1] Harold Abalson & Andrea diSessa: *Turtle Geometry,* MIT Press, 1986

[2] Manuel Alfonseca: *Representation of Fractal Curves by Means of L Systems,* APL96, Lancaster (see www.demon.co.uk/apl385/apl96)

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

Errata

From: Norman Thomson

Chris Burke has asked me to point out that my attribution to him of the J Phrase book is misleading (see Vector 13 No.3 p.117). The principal author of the J Phrase Book is Roger Hui; Ken Iverson and Eugene McDonnell were also major contributors, and other contributions were made by Chris Burke and Donald McIntyre. Also Roger is revising the Phrase Book to correct known errors, and to accommodate language changes in J3.03.

Chris would also like me to point out that a propos the article on windows programming in J-ottings 12, the windows driver was completely revised in J3, and is now much simpler. In particular `wd 'wait'` is unnecessary, and not recommended.

In Ray Cannon's article on cyphers (Vector 13.3 page 88) we unfortunately ran the three functions into one paragraph. Here they are again, done correctly:

```
  r←enc text;⎕IO
⍝Converts <text> string as encyphered numeric vector
 ⎕IO←0
 r←public modpower(,⍉16 16⊤⎕AV⍳text)

 r←dec num;⎕IO
⍝Converts numeric vector to deciphered text
 ⎕IO←0
 r←(⎕AV)[16⊥⍉(((⍴num)÷2),2)⍴private modpower num]

 r←k modpower c;s
⍝Returns mod|c*k
⍝without domain errors if k is large
 s←'mod|',(⍕c),'×'
 r←⍎(((⍴s)×k-1)⍴s),⍕c
```

Line-numbers may be unfashionable, but they do give the production team some clues where things begin and end! Authors are encouraged to use ⎕*VR* format where possible.

Stefano Lanzavecchia pointed out a crucial missing tilde in Gene McDonnell's "At Play with J" on page 131 where we should have: `w =. ~. ∧ #/. ~ .`

□PW and Resizing the Session Window under Dyalog APL/W

by Ray Cannon

With Dyalog APL/W, I found it quite inconvenient that I have to reset the □PW every time the development session window changes or I change the size of my session font.

So here is the function I came up with that I use to reset □PW which I saved in my session.

```
{r}←SetPW;old;chars;sbw;ww;cw;ver;dll;GSM
⍝ Set □PW according to APL font size and Screen dimensions
⍝ Shy result is the width set.

⍝ Uses Windows System function GetSystemMetrics to get the
⍝ width of a vertical scroll bar

⍝ This function, if placed in the Session namespace,
⍝ can be automatically run whenever the session window is
⍝ resized by setting the folowing event:-
⍝  '□se.tb'□WS'Event' 'Configure' '∆□se.SetPW'

⍝ Save environment and initialise to units of pixels
 old←'□se'□WG'coord'            ⍝ Save current coord system
 '□se'□WS'coord' 'pixel'        ⍝ Set new Coord system

⍝ Build call to Windows function GetSystemMetrics (GSM)
 ver←⍕3↑⎕wa.'□WG'AplVersion'    ⍝ Get Version of Windows
 bit16←ver≡'3.x'                ⍝ True if 16 bit Windows
 dll←⊃bit16⌽'32.C32' '.P16'     ⍝ user 16 or 32 bit dll
 'GSM'□NA'I User',dll,'|GetSystemMetrics I'

⍝ Get widths in pixels of:-
 sbw←GSM 2                      ⍝ Scroll Bar
 ww←2⊃'□se'□WG'size'            ⍝ Session Window
 cw←2⊃'□se'□WG⊂'Textsize' '□'   ⍝ Quad Character

⍝ Calculate the available width in characters
 chars←⌊(ww-sbw)÷cw             ⍝ Available window width
```

```
⍝ Reset the environment
 r←30⌈chars                    ⍝ Minimum ⎕PW value is 30
 ⎕PW←r                         ⍝ Set new Page Width
 '⎕se'⎕WS'coord'old            ⍝ Reset old coord system
```

The next job was to set this as the callback function to be triggered on resizing the screen width. However:

```
'⎕se'⎕WS'event' 'configure' '⍎⎕se.SetPW'
```

does not work. According to Dyadic, it is not possible to set the CONFIGURE event on the session. However you *can* set it on the session toolbar:

```
'⎕se.tb'⎕WS'event' 'configure' '⍎⎕se.SetPW'
```

and that does the trick.

APL and J (4): Function Application and the Axis Operator

by Chris Burke

This series of articles entitled *APL and J* will try to explain the J way of thinking, by contrasting APL and J coding examples. The term APL will be used generically to refer to facilities found in most commercial implementations of APL.

Continuing the discussion of APL's operators, we now look at the axis operator, and the whole question of how functions apply to arrays.

Housekeeping

As usual, we use the letters *V*, *M*, and *A* for vectors, matrices and arrays of any dimension, optionally followed by numbers to denote their shape. Thus *V* for any vector, *V*5 for a vector of length 5, *M*45 for a 4-by-5 matrix, *A*245 for a 2-by-4-by-5 array. Where actual values are required, *V*5 will be defined as ⍳5, *M*45 as 4 5⍴⍳20, for example:

```
      M45
 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15
16 17 18 19 20
```

APL terminology will be used, except where J terminology is required for clarity. Here, ⎕*io*←1 is assumed throughout.

Note that the behaviour of the axis operator is not identical in all APLs. This does not affect the gist of this article, but you may find that, with your copy of APL, the examples are not exactly as shown.

Overview

In this article, we cover what is perhaps the most important single difference between APL and J - the way functions apply to arrays. The experienced APLer may be surprised that there is anything to say at all, since it is not obvious where J could possibly differ from APL. Yet there are major differences that can be quite difficult to grasp. To explain what is going on, we will construct a model for APL, that can also be applied to J.

Function Application in APL

We start off with a description of function application in APL. We consider the monadic application of *f* to *A* and ask whether it is possible to split *A* up into subarrays in such a way that applying *f* in turn to each subarray, then assembling the results, would be the same as applying *f* to *A*. If so, we could consider *f* as being, in effect, defined on those subarrays. The following examples should make this clear.

Signum

Consider the signum function applied as below:

```
      □← M← M23 - 3
¯2 ¯1 0
 1  2 3

     ×M
¯1 ¯1 0
 1  1 1
```

Clearly we could apply signum to each scalar element of *M* in turn, with the same effect. Thus ×*M* is equivalent to the following array of function applications:

```
(×¯2) (×¯1) (×0)
(×1)  (×2)  (×3)
```

Since signum can apply to each scalar in turn, we say that it *applies to scalars*, or equivalently, since scalars have rank 0, we say that it is a *rank-0* function.

Note that this is not the usual meaning of the APL term *scalar function*, which means a function that not only applies to scalars, but also returns scalars. In fact, signum is such a function. However in this discussion, we are classifying functions by how they apply, and not by the results they return.

For example, we could have a function that applies to scalars, but which does not return scalar results. For such a function, we would need some way of determining how the overall result is assembled. There are essentially two such *assembly schemes* in use:

1. Suppose *sf* is the shape of the array of function applications (for example, *sf* would be 2 3 in the signum example above), and that *sr* is the shape of the result of each function application, then we could assemble the overall result as an array of shape *sf* , *sr*. This scheme works fine as long as the result of each function application does indeed have a common shape, *sr*. If not, then the

individual results would first have to be coerced into the same shape, typically by prefixing unit axes, and padding with fill elements.

2. In the second scheme, the result of each function application is enclosed, and therefore is a scalar. Consequently, `sr` is empty and the overall shape of the result is `sf`.

Both these schemes are implemented in APL, but only the first in J. Also, in APL, the result shape in the first scheme is not always `sf,sr`, but can be some permutation of this.

The two schemes are easily interchangeable. The second scheme can be obtained from the first by using a function that does an explicit enclose of its result; while the first scheme can be obtained from the second by doing a disclose of the final result (plus a transpose where the APL result is a permutation of `sf,sr`).

Sum

Now let's look at `+/` and note that unlike signum, it does not apply to scalars. If we tried to apply it to scalars, as in `+/` applied to `M23` below, this would result in `M23` unchanged:

```
(+/1) (+/2) (+/3)
(+/4) (+/5) (+/6)
```

We could say that `+/` applied with rank 0 is the identity function.

On the other hand, it should be clear that `+/` applies to vectors, and is therefore a *rank-1* function. For example, we could apply it as:

```
   (+/1 2 3) (+/4 5 6)
6 15
```

Not only can we do this, but in fact in modern APLs, this is the *definition* of `+/`. Strictly speaking, the definition has an implied enclose after each function application (and so uses the second assembly scheme described above), but since the enclose of a scalar is the same scalar, the enclose has no effect:

```
   (⊂+/1 2 3) (⊂+/4 5 6)
6 15
```

In describing a function as being *rank-n*, we mean that n is the lowest rank at which it works correctly. On higher rank arrays, the function application can be considered as partitioning the array into subarrays of rank n, then applying the

function to each subarray, and assembling the results. Thus both signum (a rank-0 function), and +/ (a rank-1 function) work fine on matrices, or arrays of higher rank.

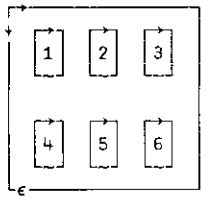
Ravel

It should be clear that the usual result of ravel is changed if it is applied with a rank less than the rank of its argument. For example, suppose we tried to apply it with rank 0 to *M23*, and used the first assembly scheme. The result of each function application to a scalar would be a vector of length 1, and we would have:

```
      ⍴ (, with-rank-0) M23
2 3 1
```

If we used the second assembly scheme, then each function application would be enclosed, and the result would have shape 2 3. This is, in fact, precisely what ,⍤⍤ does:

```
display ,⍤⍤M23
```



(In general, *f*⍤⍤ means apply *f* as a rank-0 function, opening each scalar before applying the function, and using the second assembly scheme.)

We say that ravel is a whole-array function, or equivalently, that it is *rank-infinity*.

Axis

Now let's look at what the axis operator gives us. If *f* is applied to an array *A* of rank n, and *f* is a rank-m function where 0 < m < n, then there will be more than one way of picking the rank-m subarrays of *A*. The axis operator lets you pick those subarrays. For example, if we apply the rank-1 function +/ to a matrix, then we might choose either the row-vectors or the column-vectors.

As another example, we could use axis to choose subarrays for ravel, as in:

```
      ,[2 3] A234
 1  2  3  4  5  6  7  8  9 10 11 12
13 14 15 16 17 18 19 20 21 22 23 24
```

This example means "apply ravel to axes 2 and 3 of *A234*", and is an example of the first assembly scheme described above (the individual results of ravel are not enclosed).

Also, as this example demonstrates, the axis operator not only allows you to pick the axes at which a function applies, but also, as a side effect, to *specify the rank at which the function applies.* Does this sound familiar – it should! The expression `,[2 3]A234` applies ravel *with rank*-2, to the subarrays along the last two axes.

We could also apply ravel with rank 1, as in `,[2]A234`, but since ravel applied to a vector is the identity, then `,[2]A234` is the identity on *A234* (this is an example where, when using the first assembly scheme, APL does not follow the result shape *sf*, *sr*). Hence ravel with rank 1 is also an identity function.

The notation for the axis operator also permits application with rank 0, as `,[0]`, though in practice this is not supported. If it were, we would expect that:

```
(⍴,[0]A)  ↔  (⍴A),1
```

APL Summary

At this stage, we have produced what seems to be a fairly simple and general model for function application in APL. We can classify functions according to the rank at which they are applied; we have two assembly schemes to put results together; and we can use the axis operator to specify axes where appropriate.

Unfortunately, this nice model does not always work! It has emerged from a consideration of how some functions actually behave, but does not describe how they were defined in the first place. There are simply too many exceptions for it to be consistent. We have seen one such already – the axis operator permits you to specify the rank at which a function is applied – except for rank 0 which has not been implemented! Here are some more exceptions:

Since `,[2 3]` applies ravel to the last two axes, we might expect that `⊂[2 3]` encloses the last two axes, which indeed it does:

```
      display ⊂[2 3] A234
┌→──────────────────────┐
│┌→─────────┐ ┌→──────────┐│
│└──────────┘ └───────────┘│
└∊─────────────────────────┘
```

1	2	3	4	13	14	15	16
5	6	7	8	17	18	19	20
9	10	11	12	21	22	23	24

However, try this with ⍉ or with ⍋:

```
      ⍉[2 3] A234
SYNTAX ERROR

      ⍋[2 3] A234
LENGTH ERROR
```

Not only do these not work, the error messages are wrong! The first shows a syntax error, but there is nothing wrong with the syntax, which is exactly that used earlier with ravel and enclose. The second shows a length error, but it is not possible for any monadic use of rotate to have a length error! When the APL interpreter itself gets error messages wrong, you know something strange is happening, and indeed the implementation of the axis operator is a patchwork of special cases.

Now let's try using axis with a defined function. Define sum as:

```
    ∇ r←sum m
[1]   r←+/m
    ∇
```

and note that it does not work with axis:

```
      sum[1] M23
NONCE ERROR
```

Next, consider ⎕fi, which is clearly a rank-1 function, and try applying it to a matrix. As with +/, we might expect it to apply to each row in turn and then assemble the results together, but it fails:

```
      ⎕fi ↓M23
RANK ERROR
```

We cannot complain that APL functions do not behave according to the model we have created, since no such model was used at outset. Yet there would obviously be a considerable benefit to programmers were a model of this type to be uniformly implemented.

Function Application in J

We start off by noting that the foregoing description of APL functions in terms of the rank at which they behave is identical in J. However, as mentioned earlier, J only follows the first assembly scheme (and therefore never does an implicit enclose of any result).

The big (and surprising) difference is that *J has no axis operator!* Whenever a function is applied to a set of axes in J, they are always taken from the end. Thus APL has first-axis functions, last-axis functions, and – using the axis operator – any-axis functions, whereas J has *only* last-axis functions. What J does have is a rank operator – which is just the side-effect noted earlier of APL's axis operator.

For example, compare:

```
,[2 3] A234          APL

,"2 A234             J
```

The APL expression means apply ravel along axes 2 and 3 of `A234`. It happens that these are the last two axes, and also that ravel with axis uses the first assembly scheme.

The J expression means apply ravel with rank 2 to `A234`. Since axes in J are always the last axes, then it must be that ravel is applied to the last two axes; also J only uses the first assembly scheme. Therefore, it may be seen that the two expressions are identical.

Now consider the APL expression `,[1 2]A234`. This ravels the first two axes of `A234`. There is no direct equivalent of this in J, since J has no way of specifying the first two axes. The only way to do it is to transpose the first two axes to the end, then ravel and transpose back:

```
|: (,"2) 0 1 |: A234
```

At this stage, the picture we have is that functions apply to their arguments the same way in APL and J, but the operators that modify the behaviour are different: APL has axis, which includes rank, while J has only rank. With this picture, J seems so limited, you wonder how it works at all! But J has a few cards in hand.

Consistency

First note that while J's model is simpler than APL's, unlike APL, it is applied consistently to all functions. Lets go through the earlier examples where APL fails:

In J, you can use rank to apply a function to scalars. For example, ravel with rank 0 appends a unit trailing axis:

```
   $ ,"0 A234
2 3 4 1
```

In J, all functions work with rank:

```
   |: "2 A234                    transpose rank-2
 1  5  9
 2  6 10
 3  7 11
 4  8 12

13 17 21
14 18 22
15 19 23
16 20 24

   |."2 A234                     rotate rank-2
 9 10 11 12
 5  6  7  8
 1  2  3  4

21 22 23 24
17 18 19 20
13 14 15 16
```

You can use rank with explicit definitions:

```
   sum=: 3 : '+/y.'
   sum"1 M23
6 15
```

You can apply any rank-1 function to an array of higher rank:

```
   fi=: 0&".                    equivalent of ⍎fi
   10 * fi ": M23               applied to a character matrix
10 20 30
40 50 60
```

Dyadic Use

Unlike axis, J's rank operator can be used to specify the left and right ranks of dyadic functions. In theory, axis could be used to do the same; for example, if a semicolon were used to delimit left and right axes, but this has not been implemented. For example:

```
      A=: 10 * B=: 1 2 3

    A +"0 1 B                    apply + with left rank 0, right rank 1
11 21 31
12 22 32
13 23 33
    A +"1 0 B                    apply + with left rank 1 right rank 0
11 12 13
21 22 23
31 32 33
```

Item Functions

On the face of it, J's lack of an axis operator seems incredibly restrictive. While the example mentioned earlier of applying ravel to the first two axes of a 3D array may not seem very important, it is easy to think of important examples where axis seems essential - for example in determining whether +/ of a matrix applies along the rows or along the columns. It turns out, however, that the extra facilities provided by axis are typically not required in J. This is because of *item functions*, and *prefix agreement*.

J defines many functions, including +/, to apply to the *items* of an array, which are the subarrays consisting of all axes but the first, for example the rows of a matrix.

Thus in J, +/M23 is *defined* to be:

```
      1 2 3 + 4 5 6
```

Similarly, +/M234 is *defined* to be:

```
 1  2  3  4          13 14 15 16
 5  6  7  8    +     17 18 19 20
 9 10 11 12          21 22 23 24
```

Note that this definition makes no use of axis, and indeed, +/ has rank infinity.

There is no concept as in APL, of splitting up the argument into vectors, applying +/ to each vector, then re-assembling the results.

It turns out that +/ in J behaves like +⌿ in APL. Moreover, when you change the rank of +/ in J, you get the same effect as using the axis operator in APL. For example, the following are equivalent:

J	APL
`+/ A234`	`+/[1] A234`
`+/"2 A234`	`+/[2] A234`
`+/"1 A234`	`+/ A234`

For example, +/"1 means apply +/ with rank 1, i.e. as a vector function. Since this is necessarily the last axis, it must be that +/"1 in J is the same as +/ in APL.

Prefix Agreement

J also minimizes the need for axis specification with an extension to APL's scalar agreement known as *prefix agreement*. Arguments to a dyadic function agree if the frame of one is a prefix of the frame of another. In most cases, for 'frame' you can use 'shape' to the same effect (for further discussion, see the first article in this series, Vector Vol.13 No.1 pp96-101). Thus the following works in J:

```
    M34 + 10*V3
11 12 13 14
25 26 27 28
39 40 41 42
```

Summary

In both APL and J, we can classify functions by the rank at which they apply. Function application is essentially the same in APL and J, except only that APL sometimes assembles results after first enclosing them. A major difference is that in APL you can modify function application using the axis operator, whereas in J you use the rank operator, which is a subset of the axis operator.

Although it appears that the APL model has more functionality, it is not fully implemented. Moreover, for a proper comparison, you must supplement the J model by taking into account the use of item functions and prefix agreement. In practice, the J model works well.

Reference

Iverson, K.E. *J Introduction and Dictionary*, Iverson Software, 1996

At Play With J: Stumping the Rocket Scientist

by Eugene McDonnell (eemcd@aol.com)

The Abstract Problem

This column concerns a statistical application, having to do with a rating problem involving five integer variables, related as follows:

```
a >: 0
c <: a
d < 100 * c
t <: c
i <: a - c
```

My interest in this application arose because the rating process is usually stated in quite a pedestrian way, yet has the reputation of being arcane and involved in the extreme. I'll give the pedestrian statement first, then an analysis of the statistical boundaries of the problem, next a J program following the statement as closely as possible, and lastly a J verb which is more concise and more efficient. In the second section I'll describe the physical situation giving rise to the statistical application.

To obtain the rating of a given system of these five variables proceed as follows:

> *Step 1: c divided by a. Subtract 0.3, then divide by 0.2.*
>
> *Step 2: d divided by a. Subtract 3, then divide by 4.*
>
> *Step 3: t divided by a, then divide by 0.5.*
>
> *Step 4: Start with 0.095, and subtract i divided by a. Divide the product by 0.04.*

The sum of each step cannot be greater than 2.375 or less than zero. Add the sum of steps 1 through 4, multiply by 100 and divide by 6. This is the rating.

We form the argument to the program as a five-item list:

```
a, c, d,. t, i
```

I'll write the program in J, Release 3.03, January 1997. The first line shows the change this release brings in the way of doing indirect assignment; one letter

names are now treated in the same way as multiple letter names, that is, with a space separating names.

```
Rating =: verb define
  'a c d t i' =. y.
  step1 =. ((c % a) - 0.3) % 0.2
  step2 =. ((d % a) - 3) % 4
  step3 =. (t % a) % 0.05
  step4 =. (0.095 - i % a) % 0.04
  (100*+/2.375<.0>.step1,step2,step3,step4)%6
)
```

The number of tokens in this program is easily found:

```
   #;:5!:5<'Rating'
79
```

The time required by `Rating` is 0.024. The four steps are roughly, but not exactly, the same. My impulse is to see whether I can make them exactly similar, for if we can we can take advantage of the array processing abilities of J. I take Step1 as the pattern. It has the form:

```
   ((v % a) - w) % z
```

Step2 follows the pattern exactly. Step 3 lacks the `- w` part, but that is easily fixed using the identity:

```
   x - 0
   x
```

Using this, we'll rewrite Step3 as: `step3 =. ((t % a - 0) % 0.05`

Step4 is only slightly more complicated. It reverses the minuend and subtrahend.

```
  step4 =. (0.095 - i % a) % 0.04
```

We can switch the two around by using the identity:

```
   (s - t) % u
   (t - s) % - u
```

To give us: `step4 =. ((i % a) - 0.095) % _0.04`

What I had in mind by putting them in the same form was to be able to take advantage of J's array processing abilities to get rid of the four local variables by writing something like:

```
   x =. (c, d, t, i) % a
```

or,

```
   x =. (}. % {.) y.    NB. behead divided by head
```

If we now form two lists, one of minuends and another of divisors, we can replace the four Step statements by:

```
   m =. 0.3 3 0 0.095
   n =. 0.2 4 0.05 0.04
   (x - m) % n
```

Next, reciprocate n to replace division by multiplication:

```
   ] b =. % n
5 0.25 20 25
   b * (x - m)
```

If now we distribute the multiplication within the parentheses we get:

```
   (b * x) - (b * m)
```

And, since the right limb is the product of constants, we can replace it by its product:

```
   ] q =. b * m
1.5 0.75 0 2.375
   (b * x) - q
```

I'm trying to arrive at an expression involving a linear polynomial, and am almost there. I have in mind using J's polynomial primitive (p.). For that I'll have to form a as the negate of q and reverse the order of the terms:

```
   a =. - q
_1.5 _0.75 0 _2.375
   a + (b * x)
```

Whew! We've got our linear polynomial (actually, four of them). This has been tedious, although eventually interesting. We now can replace all of the steps of `Rating` by:

```
   (100 * +/ 2.375 <. 0 >. a + (b * x)) % 6
```

or, using the polynomial primitive,

```
   (100 * +/ 2.375 <. 0 >. (a , b) p. x) % 6
```

Looking at this, we get irritated by that `100 *` and that `% 6`. We can use two identities:

```
   u * +/ v
   +/ u * v

   (+/ v) % w
   +/ v % w
```

And arrive, after a bit of algebra, at:

```
   ] e =. 100r6 * a
_25 _25r2 0 475r12
   ] f =. 100r6 * b
250r3 25r6 1000r3 _1250r3
   ] g =. e ,. f
    _25  83.3333
  _12.5  4.16667
     0   333.333
39.5833 _416.667
   ] h =. 100r6 * 2.375   NB. 39.5833 is 475r12
39.5833
```

Table `g` lists in its leading column the constant coefficients, and in the last column the linear coefficients for each of the four linear polynomials.

```
   Rtg=. [: +/ 0: >. h"_ <. g"_ p. }. % {.
```

In this verb, the trailing four items are divided by the leading item, and used as the right argument to the polynomial primitive, with the left argument table `g`. The four evaluations are constrained to lie in the interval from `0` to `475r12`, inclusive, and the constrained values are summed to give the rating.

The verb `Rtg` has 20 tokens and takes 0.007 units of time: about a quarter of the size, and less than one-third the time of the program `Rating`.

Having the four linear polynomial coefficients allows us to determine the meaningful boundaries of all systems.

```
      Table A
event    min     max
c % a    0.3     0.775
d % a    3       12.5
t % a    0       0.11875
i % a    0.095   0
```

Here's how to read this table: If, for example, the result of c%a is 0.3 or less, the rating will be 0 for the c%a event. If it is 0.775 or greater, the rating will be 475r12. Similarly for the next two rows. For the last row, a result for i%a of 0.095 or *greater* will give a rating of 0 for that event. A result of 0 (it can't be less) will give a rating of 475r12 for that event. Here are some numerical examples: The maximum rating can be obtained by the system of values:

```
   mxr =. 800 620 10000 95 0
   Rtg mxr
158.333
```

Recall that the ratings depend on the ratio of the trailing values to the leading value. When the leading value is 800, the list mxr produces the maximum rating of 158.333, since

```
   620 10000 95 0 % 800
0.775 12.5 0.11875 0
```

give the values in the column headed max in Table A. Changing the system to give the maximum values possible given the constraints listed at the beginning of this section does not give a greater result:

```
   Rtg 800 800 80000 800 0
158.333
```

Conversely, the minimum rating (zero) is obtained with the system:

```
   Rtg 800 240 2400 0 76 NB. result really 0
1e_5
```

And similarly, we can say that changing the system to:

```
   Rtg 800 0 0 0 800
0
```

will produce the same zero rating.

The Physical Problem

Now I have to apologize to readers outside the United States of America for imposing on your good nature for so long, when what I was describing derives from the parochial form of football popular in the USA but (I believe) not well-known outside that country. In that game there is a pre-eminent hero called the *quarterback*. He stands behind a line of seven myrmidons, the central one of which (called the *center*), hands the ball between his legs to the quarterback while in a crouching stance and facing away from the quarterback.

The quarterback can hand the ball in turn to one of the three other people behind the line like himself, or can run with the ball, or he can throw it forward, aiming it in the direction of one of his running team-mates. This is called a *forward pass*, and it is his ability to deliver forward passes so that they are caught by a team-mate before hitting the ground that is measured by the rating system described so laboriously above. The five variables so artfully abbreviated above are now made plain to you:

- a is the number of forward passes attempted.
- c is the number of passes caught by an eligible team-mate.
- d is the distance traversed from the line to the point of completion of the play, for all pass plays.
- t is the number of completed passes which result in a goal, or *touchdown*.
- i is the number of attempted passes which are ingloriously caught by a member of the opposing team — an *interception*.

As a sample piece of data I'll use the lifetime data of the quarterback George Blanda, who played professional football in the USA for a number of teams from 1949 through 1975. Before showing you this data, I'll interject some personal history. George Blanda and I were in the graduating class of 1949 at the University of Kentucky. George had been the successful quarterback of the college football team. He became a professional player immediately, and played for many years. When my job moved my family and me to Palo Alto, California, in the fall of 1974, I became aware that my old classmate George was still playing football for a living, and not only that, but he was a stellar performer. Week after week it was he who saved the day in the last minute for his team, the Oakland Raiders. Oakland is a large city across the bay from San Francisco, and about thirty-five miles north of Palo Alto. I was 48, but felt a resurgence of youth in seeing what my coeval Blanda was still doing on the football field. He played through the seasons of 1974 and 1975 before finally retiring (actually he was forced out by his management, who wanted to bring in younger players). George holds the career record for the total number of points scored by a football player, 2,002. The nearest player to him has scored 1,699 points.

Let us see then what George Blanda's lifetime statistics are:

attempts:	4007
completions:	1911
yards:	26920
touchdowns:	236
interceptions:	277

Applying our Rtg program gives us his career rating:

```
   Rtg gb =. 4007 1911 26920 236 277
60.6475
```

Blanda doesn't have a particularly good rating largely because of the great number of interceptions he threw. Quarterbacks with high ratings usually have many more touchdown passes than interceptions. The quarterback Joe Montana, for example, while playing for the San Francisco football team compiled a record of:

```
   jm =. 4600 2929 35124 244 123
   Rtg jm
93.4999
```

This was the highest career rating for any quarterback to have played the professional game. Ratings are also compiled during the football season, as well as for entire careers. Has anyone ever achieved the maximum rating? No one has ever done it for a career, or even for a season, but for a single game it has been done. The player John Taylor of the San Francisco team was called on in one game to throw the ball (he had never done this in a game before). It went for twenty yards, was completed, and scored a touchdown. So Taylor's rating for that game was:

```
   rtg 1 1 20 1 0
158.333
```

I got the title for this column from the fact that American sports writers and broadcasters are confident that the formula is so arcane it baffles even rocket scientists. We know better, of course. It really only baffles sports writers and broadcasters.

Bookbinders' Fun

by Jan Karman (karman@knoware.nl)

Causeway's NewLeaf is being used for highest quality PostScript™ printing. Most of business printing is just on single or multiple leaflets and gathered to bunches of A4-format reports when appropriate. Sometimes however we want to create a booklet of say 16 or more pages. Then we need to do one of the following:

1. either take a post academic course in Algebraic Group theory,
2. take a bookbinders course,
3. or just fold an A4 sheet once if you want A5 as a result, or an A3 format twice, and so on, put page numbers in the right place on the pages, e.g. bottom right on each page, unfold and watch the picture.

We want to print in the obtained order on a duplex printer in such a way that every successive bunch of say 4 or 5 sheets forms a complete booklet ready for folding and stapling.

The whole process is being done in two major steps:

1. Make detailed layout designs of each single page in the booklet, flow the text and send to spool.
2. Make one layout page with two frames (in the case of A4 to A5) without borders to speak of and flow each previously page made by 1. into it by Fetching & Including.

Adrian has provided `leaf.Fetch` and `leaf.Include` for that purpose.

A book consists of sections, a booklet has one section and 2000 booklets have 2000 sections. For making a book the sheets are folded into sections, sections are gathered, milled and melted to a book block; the book block is cut at three sides and put in a cover. A booklet may just be stapled.

APL does it like this:

```
....
[9]    ⍝ How many sections does it have?
[10]    →(0≠⎕NC'sn')+⎕LC+1 ⋄ sn←1
[11]   ⍝ How many pages does it have?
[12]    all←leaf.NumPages fi
[13]    ct←0 ⋄ Loop←sn⍴Begin
....
```

Now we need to fetch the pages for one booklet,

```
....
[15]  Begin:
[16]   np←⌊all÷sn
[17]  ⍝ Grab one booklet at a time into pre-set array
[18]   pages←(4×⌈0.25×np)⍴⊂''
[19]   pages[⍳np]←(⊂fi)leaf.Fetch¨(ct×np)+1+⍳np
[20]
[21]  ⍝ Round up to a multiple of 4 and count 2up
[22]   pg←⍳4×⌈0.25×np ⋄ np←0.5×⍴pg
....
```

Watch, the bookbinder's fun is coming!!

```
....
[24]  ⍝ ... and reorder into facing sets
[25]   set1←⍋(np⍴0 1)/(np↑pg),[0.5](⍋(-np)↑pg)
[26]   set2←⍒⍋(np⍴1 0)/(np↑pg),[0.5](⍋(-np)↑pg)
....
```

`set1` and `set2` are just matrices with page numbers in the order we need and are used as indices for leaf.Including pages from the stack.

```
....
[31]        leaf.Include(set1[1;this])⊃pages
....
```

So, not just fun, but practical and very gratifying `leaf.Include` applications for 'state of the art' printing and binding jobs.

Exercise

Interesting for the bookbinder's business, or just for fun, might be to generalize for layouts from A0, A1, ... to A4, A5, ..., or to find the most efficient layout (publishers are thinking in thousandths of cents!) for a complex book – one of my first dreams when I learned APL .

Acknowledgements

Mid sixties I was involved in a large bookbinder's progress control process: the Time Life series, tens of millions of books in seven languages, printed by Smeets Weert, Holland and bound and shipped by Proost & Brandt Amsterdam, Holland. Note that the bookbinder is responsible for the layout of the book.

I still have admiration for (the late) Mr Hangoor who designed the complex layouts for these books and from whom I learned the basic techniques.

Thanks, Adrian, for your ingeniously simple APL bookbinder's trick in line [25] and line [26]!

Function listing

```
      ∇ r←(sn)Booklet f1;np;pages;this;pg;set1;set2;sink;ct;all
[1]    ⍝ Adrian's single booklet function
[2]    ⍝ ... adjusted for multiple by JK
[3]    ⍝ Multiple (jk) booklet production. Will take any NewLeaf report
[4]    ⍝ and reduce to 2-up facing pages, ordering in the
[5]    ⍝ correct sequence for double-sided duplex mode (jk) printing.
[6]    ⍝ <f1> can be a spooled file name or just PG from the WS.
[7]
[8]     leaf.ChOrient'l'     ⍝ check and force orientation
[9]    ⍝ How many sections does it have?  ⍝ (jk)
[10]    →(0≠⎕NC'sn')↓⎕LC+1 ⋄ sn←1
[11]   ⍝ How many pages does it have?
[12]    all←leaf.NumPages f1
[13]    ct←0 ⋄ Loop←snpBegin                ⍝ (jk)
[14]
[15]   Begin:
[16]    np←⌊all÷sn
[17]   ⍝ Grab one booklet at a time into pre-set array
[18]    pages←(4×⌈0.25×np)⍴⊂''
[19]    pages[⍳np]←(⊂f1)leaf.Fetch¨(ct×np)+1+⍳np
[20]
[21]   ⍝ Round up to a multiple of 4 and count 2up
[22]    pg←⍳4×⌈0.25×np ⋄ np←0.5×≢pg
[23]
[24]   ⍝ ... and reorder into facing sets
[25]    set1←⌽(npρ0 1)/(np↑pg),[0.5](⌽(-np)↑pg)
[26]    set2←⌽⌽(npρ1 0)/(np↑pg),[0.5](⌽(-np)↑pg)  ⍝ first ⌽ for duplex
[27]                                              ⍝ without for 2 runs
[28]   ⍝ Run all through the printer ...
[29]    ⍞←'leaf.',(⊃(ct=0)⌷'Switch' 'Use'),' nob.ebs.booklet'
[30]    :For this :In ⍳¯1↑⍴set1
[31]        leaf.Include(set1[1;this])⊃,pages
[32]        leaf.Include(set1[2;this])⊃,pages
[33]        leaf.Include(set2[1;this])⊃,pages
[34]        leaf.Include(set2[2;this])⊃,pages
[35]    :End
[36]    ct+←1 ⋄ →Loop←1⌷Loop
[37]
[38]    sink←leaf.Spool'run1'
[39]    r←'PostScrp.Print ''run1'''
      ∇
```

On First Encountering OLE

by Adrian Smith (causeway@compuserve.com)

Background

It is much too soon to write a decent appreciation of the work Dyadic have put in since Lancaster to make Dyalog APL 8.1 a robust and effective OLE server. However it seems only fair to record my experiences to date, so that other readers can get a little further a little sooner into this strange new world.

The idea is very simple - you register a namespace as an OLE Server, with a list of the functions and variables that you want made public. Then users of other applications, such as Excel, can call any of the public functions, passing arrays and getting array results. As long as Excel declares the OLE server globally, with statements such as:

```
Public Const g_a_s = "dyalog.Adrian"
Public g_oobj_AdrianServer As Object

Function AdrianData() As Variant

    Set g_oobj_AdrianServer = CreateObject(g_a_s)
    AdrianData = g_oobj_AdrianServer.MakeData

End Function
```

... then Excel loads your server on the first function call and keeps it loaded until you quit Excel or close that workbook. This brings me to the first little warning:

- calls to Dyalog APs such as NFILES or XUTILS are probably a bad idea from an OLE server. The AP is left running after Dyalog is closed, and eventually you will stack enough of them up in memory to prevent it from loading.

The mapping from APL nested array to VBA 'Variant' seems very solid, with only the occasional lapse on Excel's part, such as its inability to store arrays where some (but not all) of the dimensions are zero.

Registering your Server

There is a hard connection between the act of saving the workspace and updating the Windows registry with the information about your APL server. This is to ensure that the entries in the registry are always in line with the saved workspace,

and in general it works pretty well, but again there a couple of snags to look out for. Having made our *OleServer* object:

```
  r←Make
∩ Make me a server please!
 ⎕WC'OleServer'
 r←⎕WG'ClassId'
 ⎕WS'ExportedVars'(,c'Public')
 ⎕WS'ExportedFns' Public

     Make
{B1B355CF-A765-11D0-B01F-00C04FDE82D1}
```

... and subsequently saved the WS, we can go exploring in REGEDIT, to see what the results were.

```
HKEY_CLASSES_ROOT
    dyalog.Adrian
       CLSID={B1B355CF-A765-11D0-B01F-00C04FDE82D1}
```

... so far so good. Now how does Excel know what to run when we invoke the function on the previous page?

```
HKEY_CLASSES_ROOT
    CLSID
      {B1B355CF-A765-11D0-B01F-00C04FDE82D1}=Adrian
        LOCAL_SERVER32=C:\APL\DYALOG81.EXE C:\DWS\ACDS.DWS
        TYPELIB={B1B355D0-A765-11D0-B01F-00C04FDE82D1}

    INTERFACE
      {B1B355D1-A765-11D0-B01F-00C04FDE82D1}

    TYPELIB
      {B1B355D0-A765-11D0-B01F-00C04FDE82D1}
         1.0
            0
               WIN32=C:\DWS\Adrian.tlb
```

.... note that there are actually three consecutive ids registered (the counter is at the end of the first block) of which Dyalog admits to the CLSID and the TYPELIB. At the moment, I have no idea what the INTERFACE is for, but the rest of it is essential. The LOCAL_SERVER entry starts the interpreter and the CLSID points to the namespace. The .TLB file catalogues the public functions and variables.

Now for the next little snag - if you get carried away and run that *Make* function again, Dyalog happily generates a new CLSID and duly registers it on saving the workspace - unfortunately it leaves the lower three entries in place (orphaned) so

everything will still work, but sooner or later you are going to have to roll up your sleeves and clean up the registry by hand! Now for the real gotcha ...

- do not under any circumstances run `⎕EX '.'`. This will wipe any OleServerness in the workspace, but still leaves all the servers registered. Now when you try starting your server from Excel it will kick off APL and load the WS. And wait ... and wait .....

Fortunately there is a Dyalog 'method' which you can use to unregister an existing server, but it is essential to run it before you accidentally wipe it out or re-create it, as you have no way of recovering that magic id string otherwise.

Debugging it

Off we go to Excel, crack open the VBA "Complete Reference" and write ourselves some toy functions. Everything is going fine until we get back a message like:

```
OLE ERROR 1004
SERVER REPORTED A LENGTH ERROR ON LINE 14
```

... and we want to track down the error. The problem with the Dyalog server is that it bounces the error back to Excel and simply clears the stack. Even if you are running it in a development interpreter, you cannot see what the function arguments were, nor check the values of any local variables at the point of the error. My current solution is to set an error trap (using `:Trap` in all functions as a global `⎕TRAP` currently has no effect in a server) to run a simple error analysis:

```
SeeError dm;av;nm;log;var
⍝ Simple error analysis
 log←''('ERROR ANALYSIS AT ',⍕⎕TS)
 :If 2<⍴⎕SI
   log,←(⊂'CALLING STACK'),(2↓⎕NSI,¨'.',¨⎕SI,¨'[',¨(⍕¨⎕LC),¨']'),' '
 :End
 log,←dm,' ' 'WORKING VARIABLES'
⍝ Take the line apart bit by bit and test each name
 dm←2⊃⎕DM ⋄ dm←(dm≠' ')⌿dm
 av←⎕AV/⍨0=⊃∘⎕NC¨⎕AV
Next:dm↓⍨←¯1+1⍳⍨dm∈av,'#'
 nm←dm↑⍨¯1+0⍳⍨dm∈av,⎕D,'#.'
 →(0∈⍴nm)↑Done ⋄ dm←(⍴nm)↓dm
 :Select ⊃⎕NC nm
   :Case 0
     log,←⊂nm,' - undefined'
   :CaseList 3 4
     log,←⊂'Subfunction - ',nm
   :Case 2
     var←⎕fmt↑nm
```

```
      :If 256<⍴var
        var←' ...',⍨250↑var
      :End
      log,←⊂nm,'←',var
    :Else
      log,←⊂'Invalid name class - ',nm
  :End
  →(0<⍴dm)↑Next
  Done:See↑log
```

... and then to throw this into an 'Ontop' window. Message boxes are not very useful (they tend to appear underneath Excel) so you need to force the log to get drawn with a trick such as:

```
'ff'⎕WS'Event' 1001 1 ⋄ ⎕NQ'ff' 1001 ⋄ ⎕DQ'ff'
```

Installing a Runtime Server

This is where the real fun begins. There are two key tricks needed to make it work; one is undocumented and the other is documented incorrectly! The first obvious problem is how to register the OleServer (a namespace in a workspace, remember) as part of a standard install routine. There are two possible attacks:

1. Write out a text file with the registry script required to add the correct entries, and call REGEDIT in your installer, passing the script on the command line.

2. Run up your workspace as the last step in the installation and get it to register itself. This is the undocumented bit!

It is hard to see how you could do (1) reliably, given that your users are unlikely to take any notice of your suggested directory, so the script file will need to be crafted by the installer on the user's machine. I was leaning towards the idea of ⎕*SAVE*, which feels very dangerous, when a chance phone call revealed that there is also a 'Register' method on the server. Now we are getting somewhere ...

```
 Shell;id
⍝ Start from here
⍝ Called on installation to register the NS as a server.
⍝ Else just falls in to server mode.

:If 0∈⍴2 ⎕NQ'.' 'GetEnvironment' 'Register'
  ⍙LX
 :Else
  id←Reg  ⍝ See below
  Win_msg'OLE Server registered as ...'id
  ⎕OFF
:End
```

... where *Reg* just queues the appropriate (undocumented) method. I really thought that I had cracked it, and even managed a successful install on my own machine under a different user id, in a completely new directory path. OK, make a disk set, walk across to Duncan's machine and away we go. Everything goes just fine until it attempts the registration, and gets an error saving **Adrian.tlb** – of course the workspace was saved on my machine, and the namespace thinks it knows where the typelib should be created. On my machine, the path exists, so my 'test with another user' trial was a complete fraud. Duncan has a much more rationally structured disk, and I naturally hit a file name error on trying to create the file in a non-existent directory. *Bother*, said Pooh.

At this point, I made the bad mistake of reading the instructions. If you look up the *'TypelibFile'* property in the Dyalog help file, it informs you that this is a read-only property of the server. Fortunately I have had sufficient experience of the Microsoft C++ documentation to know that the software is often ahead of the documentation department, so I gave it a try anyway ...

```
  r←Reg
⍝ Register the namespace with the right TypeLib
  r←'Adrian'⎕WG'ClassId'
  'Adrian'⎕WS'Typelibfile'('Adrian.tlb',⍨⎕WSID↓⍨¯1-'\'⍳⌽⎕WSID)
  2 ⎕NQ'Adrian' 530
```

... sorry for all the gratuitous commuting – this was written at home with Richard looking over my shoulder, and I do this sort of thing to confuse him. The important thing is that this works with no fuss and is very easy to build into an install script. Assuming your installer has a parameter for the installation directory, you will need to run something like:

```
$i\dyrt81.exe $i\acds.dws Register=Yes
```

... when all the files have been copied.

Conclusion

Considering the amount of new stuff, this first release of Dyalog's OleServer technology is remarkably solid and already feels as if it could be the vehicle for a serious set of runtime applications. They have also removed the requirement for all those pesky DOTs and DINs so you really can ship just a workspace and an interpreter. All we need now is a smaller interpreter!

Memories of Gérard Langlet

compiled by Anthony Camacho

The Vector working group decided that we could not do justice to Gérard Langlet in an obituary from our own resources. Maybe we will reprint something from *Les Nouvelles d'APL* or elsewhere when we see what people write. We agreed to collect mementoes and so sent out email messages asking people to contribute a paragraph to include in the collection.

We had many apologies from people who felt their English wasn't up to it or who didn't know Gérard well enough. Below are the contributions we have received so far that we feel able to include.

Dieter Lattermann (APL Club of Germany)

While preparing for APL96 at Lancaster I received a letter from Gérard from the hospital. He offered a talk on the famous "difference scan", which he considered to be the key to the "theory of everything". The programme committee initially was hesitant to accept his offer. There was a feeling that not much new information could be expected. However, it was eventually decided to listen once more to Gérard. I think, nobody attending his sessions regretted this. I had not taken his stay at he hospital too seriously and was glad to meet him again at Lancaster. There he indicated that this was probably his last APL conference. This utterance struck me and I refused to believe it. Unfortunately, as you all know, he was right.

I regret very much that I never managed to get him to perform at a meeting of the APL Club of Germany (my mastering of the English language is poor, but I think "performance" describes closely what we missed).

The APL community has lost a great, innovative promoter of APL and, at least as important, a human being who helped to make the life in this community so enjoyable. We will never forget him.

Anthony Camacho

Long convinced that Gérard had hold of the tail of the dragon that might lead to the secret of the universe, I did my best to get Vector to support his endeavours. (See the Editorial in Vector 11.2) While holding the tail of the dragon, he managed to be entertaining, engaging, witty and thoughtful. He gave me a T-shirt that displayed difference-scan for all the world to see. I still wear it, but less often than in 1994 (not because I am less convinced but for fear of wearing it out). I wish I had known him when I was younger and

cleverer. I wish I spoke French well enough to appreciate all he wrote (Sylvia and I have tried, in vain, to do him justice in translation).

Howard A. Peelle

I remember, with delight, Gérard's delight in watching his wife learn APL during the "boat conference" after APL92 in St. Petersburg.

Dick Holt

Thanks for doing a tribute to Gérard Langlet. Below is my email to c.l.a. I wrote it at a time when I was emotionally upset by his death. Upon reflection, I've decided not to change it.

The c.l.a. message below from Christian Scherer needs no translation.

The APL community deeply mourns the death of Gérard Langlet. His witty, learned, and creative work will long be treasured by his colleagues. Speaking personally, I'll remember, and miss, him for the rest of my life.

>Gérard Langlet est mort le 22 décembre.
>Il sera enterré au cimetière de Jouy-en-Josas le vendredi 27 décembre à
>10h45
>Christian Scherer

Eric Lescasse

Gérard was a researcher, a real researcher. Once Gérard had discovered APL, his life changed: he quickly became an expert in APL and used it ever since for his work, but also spent considerable efforts trying to communicate to others his passion for this language. And he did succeed, many of his fellow researchers at CEA having become APL users, following him. He had a rare and enormous work power, still finding the time to read a real lot of books. He became interested in seeing how APL could be applied to a lot of different areas crystallography, physics, biology, mathematics, poetry, etc. and published countless scientific articles and research papers in particular in the French APL Association review *"Les Nouvelles d'APL"* which he has been setting up himself for the last 2 years of his life. Through these articles he could expose important scientific discoveries he had made in all these domains, thanks to his particular use of APL and deep comprehension of some APL primitives and operators (not equal scan) on booleans. Many admire him for his researches and discoveries; all should have a real great respect for the courage he has shown throughout his illness, going on writing articles and publishing *"Les Nouvelles d'APL"* until the very end of his life and with never a complaint.

Michel Dumontier

I have just this evening to write a memory of Gérard Langlet and I shall try to be concise: I have so many memories of him as we have known each other for a long time...

> Gérard Langlet was better than a friend or a brother for me, our relations were different than with other people, because we had some affinity and we understood each other, which is not commonly the case for a friend, even for a brother. The chief thing that made a strong relationship between us was, (everybody has guessed it!) ... APL of course.
>
> But there is not only that: scientific understanding of biology, mathematics, languages, human behaviour, the way to feel things, to make relations between things, and we can add: our common understanding of things through APL, which is very different (I assert it strongly) and not comparable with the understanding of what I call a 'classic' [1] person who has not in mind the special tools, the only tools of thought cabled in the neurons by a long practice of APL.
>
> Gérard would not contradict me as he said sometimes: 'To practise APL you need to have your neurons aligned (orienté in French) in a particular way'... We did not have the same ideas on all the domains or matters but we naturally avoided speaking in these domains, so we remained good friends. (All persons are different even twins or clones; in nature all entities are different even electrons or smaller particles: a single reason is that they cannot be in the same place at the same time and here also, Gérard would not contradict me because of his belief in APL difference scan (≠\).)
>
> The first time that I met Gérard was either at an APL AFNOR meeting in Paris or an APL AFCET conference. Even though I practised APL since 1969, I cannot have met him before 1974 as it was only at this time that he began to be interested in APL, after having been transfixed by meeting a screen in a corridor which was displaying APL characters. That puzzled him and he was curious to know what it meant, because he had some natural curiosity about languages.
>
> He confided to me that he took six months to withdraw from his mind all the false things that they taught him in FORTRAN...to accept APL understanding. We have in common 11 APL congresses since 1985. I think that it was for him as for me, perhaps not with the same intensity, a kind of bliss to meet APL friends, generally for two weeks because we remained for APL ISO standard meetings.
>
> The week-end between APL congress and APL ISO meeting, he rented a car and we visited together or with other APL friends some beautiful countryside around and also various restaurants! I have kept many photos since 1985 and video records since 1988 of those precious moments.

I shall give just one anecdote to show the nature of our complicity and perhaps our behaviour (some would say French behaviour... I am not so sure!). During the week-end I spoke of above, in 1993, we visited a beautiful area near the Niagara falls where I remember there was a peach festival. During a walk in the streets of the village, we saw a prestigious hotel. I had no difficulty in convincing Gérard to visit the hotel where there was, if I remember it well, a sale by auction. In a great lounge, we saw a piano and naturally, we played one after the other some classical pieces of music and nobody said anything to us, we left as we came... we had also the love of music in common...

Naturally, we have not only exchanged many ideas but also much software, especially APL software. We wrote also papers in the same journals especially the French APL review: *'Les Nouvelles d'APL'* where he was the chief editor.

His last issue was that of December 1996. His last APL conference with me was that of Lancaster. As he had difficulties in walking, I brought him, his wife, my wife and my daughter in my car from his home to the conference. I think that it was a good memory for him, as it is for us.

He also said that nothing is purely haphazard, but rather what in English you call hazard (chance, coincidence), and he said for instance that it was not haphazard that we met each other.

I have surely a special memory of Gérard that I have never had for any other person and I think, as many people do, that Gérard will be missed in APL circles.

You will find at the following URL a biography of Gérard, two speeches at the cemetery, one from his boss Edgar Soulie, another from a common APL friend Sylvain Baron, a condolences book, the articles of the French APL review and other useful hypertext references: `http://www.ensmp.fr/~scherer/langlet/`

[1] I call also 'classic' mathematics, the mathematics (of the 17th century) that is only and still now taught at school. The mathematics of a person who doesn't know APL and has consequently no tool of thought.

Curtis Jones

My favourite memory of Professor Langlet was at his symposium at the NY/SIGAPL's Tool of Thought in January 1993. He kept a large crowd of us packed into a small meeting room interested and amused as he led volunteers through several exercises on successive differences in binary vectors. I'm still curious about how an edge of a triangle of bits from not-equal scan can be a Fourier transform. It was the call for papers with some relation to the parity scan that goaded me to write about inner-product scans for that Tool of Thought seminar. In my next paper I wanted to show "my" inner-product

scan generation of a Fibonacci series and cast it in a form that used Langlet's geniton. I looked at his paper in the APL94 proceedings [1] for a geniton, and found he had already demonstrated an inner-product scan to generate a Fibonacci series!

```
+.×\9⍴⊂2 2⍴1 1 1 0
```

I'm sorry that Gérard will not be able to write the book I think is needed to explain his work. It was good to see at APL96 in Lancaster that Michael Zaus is explaining and building on Langletian ideas of parity logic.

[1] Gérard A. Langlet, *"The APL Theory of Human Vision"*,
APL Quote Quad, Vol. 25, No. 1, pp. 105-121 (APL94, Sep. 1994)

Eugene McDonnell

Gérard Langlet was an inspiring and inspired teacher of APL. I have met many people who were introduced to APL in one of his classes. He was a theorist whose novel ideas have given us much to think about. And he was diligent in attending the meetings of the APL standards group, where he was a steady contributor to the work of that group.

Curtis Jones points out that Gene was the recording secretary of the APL standards group.

Norman Thomson

Gérard was entertaining, provocative, endlessly self-confident while at the same time warm in his enthusiasm for just about anything he put his hand to. He was undoubtedly one of the greatest characters who turned up at almost all the APL conferences, and will be missed there by many.

≠ \

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:

Duncan Pearson,
Causeway Graphical Systems Ltd
The Maltings, Castlegate
MALTON, North Yorks, YO17 0DP
Tel: +44 (0) 1653-696760
Email: 100265.1564@compuserve.com

Authors wishing to use Word for Windows may contact Vector Production for a copy of the APL2741 TrueType font, and a suitable Winword template.

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. Please also copy us with all electronically submitted material so that we have early warning of possible problems.

Tel: +44 (0) 1439-788385
Email: 100331.644@compuserve.com

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: _____

Category (please tick box) to run from: 1st May ❑ August ❑ Nov ❑ Feb ❑

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 or by Visa/Mastercard/JCB 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, Visa or JCB number.

I authorise you to debit my Visa/Mastercard/JCB 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, UK

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.

1996/97 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 0117-973 0036 100612.1057@compuserve.com	11 Auburn Road, Redland, BRISTOL, BS6 6LS
Treasurer	Nicholas Small 0181-980 7870 treas.apl@bcs.org.uk	8 Cardigan Road, LONDON E3 5HU
Journal Editor	Duncan Pearson 01653-696760 causeway@compuserve.com	Causeway Graphical Systems Ltd, The Maltings, Castlegate, MALTON, North Yorks YO17 0DP
Webmaster	Ray Cannon 01252-874697 100430.740@compuserve.com	21 Woodbridge Road, Blackwater, Camberley, Surrey GU17 0BS
Activities	Jon Sandles 01904-612882 jon_sandles@compuserve.com	138 Burton Stone Lane, YORK YO3 6DF
Education	Dr Ian Clark 01388-526803 100021.3073@compuserve.com	9 Hill End, Frosterley Bishop Auckland Co. Durham DL13 2SX
Projects and Publicity	Anthony Camacho 0117-973 0036 100612.1057@compuserve.com	11 Auburn Road, Redland, BRISTOL, BS6 6LS
Administration	Rowena Small 0181-980 7870 treas.apl@bcs.org.uk	8 Cardigan Road, LONDON E3 5HU

Journal Working Group

Editor:	Duncan Pearson	Causeway Graphical Systems Ltd (01653-696760)
Production:	Adrian & Gill Smith	Brook House, Gilling East, YORK (01439-788385)
Advertising:	Gill Smith	Brook House, Gilling East, YORK (01439-788385)

Support Team: Jonathan Barman (01488-648575), Richard and Adam Weber (01302-539761), Anthony & Sylvia Camacho, Ray Cannon (01252-874697), Marc Griffiths (0171-289 0471), David Ziemann (0181-348 4039), Jon Sandles (01904-612882), Ajay Askoolum

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.

Causeway Graphical Systems Ltd
The Maltings, Castlegate,
MALTON, North Yorks YO17 0DP
Tel: 01653-696760
Fax: 01653-697719
Email: causeway@compuserve.com
Web: www.causeway.co.uk

Compass Ltd
Compass House
60 Priestley Road
GUILDFORD, Surrey GU2 5YU
Tel: 01483-514500

Dyadic Systems Ltd
Riverside View, Basing Road,
Old Basing, BASINGSTOKE,
Hants, RG24 0AL
Tel: 01256-811125
Fax: 01256-811130
Email: sales@dyadic.com
Web: www.dyadic.com

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

Insight Systems ApS
Nordre Strandvej 119C
DK-3150 Hellebæk
Denmark
Tel: +45 49 76 20 20
Fax: +45 49 76 20 30
Email: info@insight.dk
Web: www.insight.dk

APL2000
6610 Rockledge Drive
Suite 502
Bethesda MD 20817 USA
Tel: +1 (301) 564-5020
Fax: +1 (301) 564-5021
Email: sales@apl2000.com
Web: members.aol.com/APL2000

MicroAPL Ltd
South Bank Technopark
90 London Road
LONDON SE1 6LN
Tel: 0171-922 8866
Fax: 0171-928 1006
Email: microapl@microapl.demon.co.uk
Web: www.microapl.co.uk

Soliton Associates Ltd
Groot Blankenberg 53
1082 AC Amsterdam
Netherlands
Tel: +31 20 646 4475
Fax: +31 20 644 1206
Email: sales@soliton.com

Dutch APL Association
Postbus 1341
3430BH Nieuwegein
Netherlands
Tel: +31 347 342 337